JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 11, pp. 67-80, November 2022
https://doi.org/10.9708/jksci.2022.27.11.067

Automatic Creation of ShEx Schemas for RML-Based RDF Knowledge Graph Validation

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 the ShEx schemas to describe and validate RDF knowledge graphs constructed by RML mapping. ShEx schemas consist of constraints. The proposed system generates most of the constraints by converting the RML mapping rules. The schemas consisting only of constraints obtained from mapping rules can help users to figure out the structure of the graphs generated by RML mapping, but they are not sufficient for sophisticated validation purposes. For users who need a schema for validation, the proposed system is also able to provide the schema with added constraints generated from metadata extracted from the input data sources for RML mapping. The proposed system has the ability to handle CSV, XML, JSON or RDBMS as input data sources. Testing results from 297 cases show that the proposed system can be applied for RDF graph validation in various practical cases.

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

[요 약]

본 논문에서는 RML 매핑 방식으로 생성된 RDF 지식 그래프의 구조를 묘사하고 검증할 용도의 ShEx 스키마를 자동으로 생성하는 시스템을 제안한다. ShEx 스키마는 제약 조건들로 구성된다. 제안된 시스템은 대부분의 제약 조건을 RML 매핑 규칙을 변환하여 생성한다. 매핑 규칙에서 유도된 제약 조건만으로 구성한 스키마는 사용자가 RML 매핑으로 생성한 그래프의 구조를 파악하는 데 도움을 주는 용도로는 부족함이 없지만 정교한 검증 용도로 사용하기에는 충분치 않다. 검증 용도에 부합하는 스키마가 요구될 경우, 제안된 시스템은 RML 매핑의 입력 데이터 소스에서 추출한 메타데이터를 사용해 만든 제약 조건이 추가된 스키마를 생성할 수 있다. 제안된 시스템이 지원하는 입력 데이터 소스 유형은 CSV, XML, JSON, RDBMS다. 297개의 테스트 케이스로 구성된 실험에서 보인 결과는 제안된 시스템이 RML 매핑으로 생성된 RDF 그래프 검증을 위해 범용적으로 사용될 수 있음을 보여준다.

▸ **주제어**: RML, ShEx, 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. 10. 27, Revised: 2022. 11. 10, Accepted: 2022. 11. 16.

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

I. Introduction

RDF(Resource Description Framework) 지식 그래프는 도메인을 묘사하는 사실들을 그래프 모델로 구성하고 RDF 형식으로 표현한 것이다[1]. 최근 들어 Google, Facebook, Microsoft, Amazon과 같은 인터넷 기업들에서부터 제조, 유통 등 다양한 분야의 기업들까지 지식 그래프 구축에 힘쓰고 있다[2]. 그 목적은 지식 그래프를 사용해 챗봇, 검색, 추천 등의 지능형 서비스에 문맥 정보를 제공하기 위해서다[2-3].

RDF 지식 그래프는 주로 데이터베이스 혹은 CSV, JSON, XML 등 다양한 포맷의 파일에 산재되어 있던 기존 데이터를 RDF 포맷으로 변환하여 통합하는 방식으로 구축된다[4]. RML(RDF Mapping Language)은 이러한 방식에서 사실상 표준으로 사용되는 매핑 규칙 정의 언어다[5]. 매핑 규칙은 기존 데이터가 RDF 그래프의 무슨 요소로 어떻게 변환되어야 하는지를 기술해 놓은 변환 규칙을 말한다. RML 매핑 규칙은 RML 프로세서에 의해 실행된다. 즉, RML 프로세서는 매핑 규칙에 따라 기존 데이터에서 RDF 그래프를 생성해주는 시스템이다. 최근에는 Amazon 클라우드 플랫폼에서 동작하는 Data Lens[6]라는 RML 프로세서가 공개되어 주목받고 있다.

본 논문은 RML 매핑으로 생성되는 RDF 그래프의 구조를 묘사할 수 있고 검증할 수 있는 ShEx(Shape Expressions)[7] 언어로 표현한 스키마를 자동으로 생성해주는 시스템을 제안한다. 제안된 시스템은 매핑에 의해 생성되는 그래프에 대한 스키마의 부재로 인하여 겪는 작업자의 다음 두 가지 어려움에 대한 해결책을 제공할 수 있다. 첫째, 매핑 규칙 작성 시 작업자는 작성한 규칙이 본래 의도한 구조의 그래프를 생성케 하는 것인지 확신하기 어렵다. 둘째, 생성된 그래프를 검증기로 검증하고자 할 때 작업자는 수작업으로 스키마를 작성해야 하며 이로 인해 검증 도중 오류가 발생하더라도 그 원인이 스키마에 의한 것인지 그래프에 의한 것인지 판단하기 어렵다. 각각의 문제 상황에 적합한 스키마가 제공될 수 있도록 제안된 시스템은 입력으로서 RML 문서만을 사용하거나 RML 문서 뿐 아니라 매핑의 소스 데이터도 함께 사용할 수 있도록 설계되었다. RML 문서만을 사용하여 생성한 스키마는 사람이 그래프 생성 전에 그 구조를 미리 파악하고자 하는 상황에 적합하다. 그래프 구조 묘사에 해당하는 제약 조건들은 RML 문서 내 매핑 규칙에서 유도되기 때문이다. 검증에 사용할 스키마는 RML 문서와 매핑의 입력 데이터 모두를 사용하여 생성한 스키마가 적합하다. RML 매핑 규칙에서는 획득할 수 없고 매핑의 입력 데이터 분석을 통해서만 획득되는 정보로 만들 수 있는 제약 조건들 중 검증기가 검증을 위해 필수로 요구하는 제약 조건이 존재하기 때문이다.

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 우선 RDF 그래프의 구조 검증을 위해 고안된 언어인 SHACL(Shapes Constraint Language)[8]과 ShEx를 비교하고 ShEx를 선택한 이유를 밝힌다. 그다음 RDF 그래프를 매핑 방식으로 생성하고자 할 때 사용하는 대표적인 기술인 Direct Mapping[9], R2RML(RDB to RDF Mapping Language)[10], RML과 더불어 이들로 구축한 RDF 그래프에 대한 스키마를 생성하는 기존 연구를 간략히 소개한다. 특히, RML 기반의 기존 연구가 보이는 한계를 자세히 분석하여 본 연구에서 해결하고자 한다.

1. Comparison of SHACL and ShEx

SHACL과 ShEx는 모두 RDF 지식 그래프 구축이 대규모화됨에 따라 RDF 질의 언어인 SPARQL[11] 혹은 웹 온톨로지 언어인 OWL[12]을 본래의 용도와는 달리 RDF 그래프 구조 검증을 위해 대용했던 기존 방식의 불편을 해소하기 위해 개발된 RDF 그래프 구조 검증 전용 언어다[13]. 두 언어 모두 셰이프(shape)라는 것을 검증 단위로 사용한다. 셰이프는 문자 그대로 트리플의 형상을 말한다. 트리플은 텍스트로 표현한 RDF 그래프에서의 한 문장을 의미하며 주어, 술어, 목적어 순서의 세 요소로 구성되기에 트리플이라 부른다. 주어와 목적어는 그래프의 정점에 대응되며 술어는 그래프의 간선에 대응된다. RDF 문서는 다른 구문 요소 없이 단순히 트리플들의 나열일 뿐이지만 서로 다른 트리플에서 주어 자리든 목적어 자리든 동일한 것이 있다면 정점이 중첩된 것으로 기계에 의해 해석된다. 따라서 셰이프는 한 정점을 중심으로 그 정점을 공유하는 트리플들의 요소별 제약 조건들로 구성된다. 검증은 스키마에 구비된 셰이프들에 RDF 그래프의 특정 정점을 타겟 노드라는 자격으로 대응시킨 뒤 그 타겟 노드를 포함한 트리플들의 요소별 형상이 짝을 이룬 셰이프의 제약 조건들에 부합하는지를 검증기로 확인하는 작업이다.

SHACL은 2017년 발표되었으며 W3C 표준 언어다. SHACL의 가장 큰 특징은 SHACL의 논리 모델을 표현할 신택스로서 RDF 모델을 표현하기 위한 신택스를 차용했다는 것이다. 이것의 의미는 SHACL 스키마가 기계에 의해 RDF 그래프로 취급될 수 있다는 것이며 이로 인한 장점은 RDF를 지원하기 위해 개발된 제반 시스템들을 SHACL이 그대로 사용할 수 있다는 것이다. 그러나 RDF 신택스는 기계가 그 신택스로 작성된 텍스트를 그래프 모델로 용이하게 해석할 수 있도록 설계된 것이라서 사람에게 정보를 효율적으로 전달할 수 있는 체계는 아니다. RDF 신택스의 한 문장을 세 요소로만 구성해야 하는 제한으로 인해 셰이프처럼 여러 속성으로 이루어져 하나의 의미 단위를 형성하는 대상을 사람에게 시각적으로 간결하게 전달하기 어렵다. 실제로 SHACL은 하나의 트리플을 할애해 제약 조건 하나를 기술하기 때문에 셰이프가 여러 개의 분절된 트리플로 기술될 수밖에 없다. 이러한 구조는 사람이 기계와 같은 방식으로 셰이프의 의미를 파악해 나가게 하는 불편을 초래한다.

ShEx는 2018년 W3C Community Groups에 의해 발표되었다. ShEx는 인간 가독성(human-readability)을 위해 고유 개발한 ShExC(ShEx Compact Syntax)라는 신택스와 기계 가독성을 고려하여 JSON-LD[14] 기반으로 정의한 ShExJ(ShEx JSON Syntax)라는 두 개의 신택스를 제공한다. W3C 표준인 JSON-LD는 최근 들어 웹 애플리케이션에서 사실상 표준처럼 사용되는 데이터 교환 포맷인 JSON으로 RDF 데이터를 표현하기 위해 정의한 RDF 신택스 중 하나다. ShExJ로 인해 ShEx 스키마도 RDF 그래프로 취급될 수 있으며 ShExC는 RDF 신택스 중 가독성이 가장 우수한 Turtle[15]보다 간결하다[16]. SHACL과 ShEx를 신택스 관점이 아닌 언어 사양에서 정의한 제약 조건들로 가할 수 있는 그래프 형상의 폭인 표현력 관점에서 비교하면 두 언어는 완전히 동등하지는 않으나[17] 일반적인 사용 사례들을 위해서는 비등하다[16].

본 논문은 제안된 시스템이 생성한 스키마로 RML 매핑 전후에 사람이 신속하게 그래프의 구조를 파악하게 하는 것이 목적이기에 인간 가독성 측면에서 우수한 ShEx를 스키마 언어로 선택했다. ShEx를 선택함에 따른 표현력 측면에서의 이득 또한 존재한다. RDF 그래프에서는 정점의 중첩에 의한 트리플의 연결이 연쇄적으로 발생할 수 있다. 이러한 구조를 반영하기 위해 두 검증 언어 모두 셰이프 간 참조를 통해 연관을 표현할 수 있도록 한다. 그러나 SHACL에서는 셰이프 간 참조가 사이클을 이루는 구조에 대한 검증기의 동작은 정의되어 있지 않다[16]. 즉, SHACL에서는 셰이프 간 참조를 단방향으로만 기술해야 안전하다. 이와는 달리 ShEx에서는 양방향 참조가 문제 되지 않는다. 이로 인해 한 셰이프가 다른 셰이프에 의해 참조되고 있다는 것도 표현할 수 있다. 이러한 표현의 장점은 사용자가 주목하고 있는 셰이프와 연관을 맺고 있는 주변 셰이프를 빠르게 파악할 수 있게 한다는 것과 그래프 전체 구조 파악을 어느 셰이프에서나 시작할 수 있게 한다는 것이다.

2. Direct Mapping, R2RML and RML

매핑에 의해 RDF 그래프를 생성하는 접근법은 RDB 데이터를 RDF 포맷으로 공개하고자 하는 노력에서 시작되었다. 이러한 목적의 다양한 시도는 2012년 발표된 Direct Mapping이라는 W3C의 표준 알고리즘으로 수렴되었다. Direct Mapping은 하나의 RDB 전체 데이터가 변환 대상이며 변환된 RDF 그래프는 관계형 모델의 의미를 반영한다. 즉, Direct Mapping 사양은 하나의 RDB가 주어졌을 때 그 RDB 데이터 사이에 존재하는 관계형 모델에 기반한 의미를 왜곡 없이 RDF 포맷으로 표현하기 위한 매핑 규칙 그 자체이다. R2RML은 Direct Mapping과 같은 해에 W3C의 표준으로 발표되었다. R2RML과 Direct Mapping의 공통점은 매핑의 소스 타입이 RDB로 한정된다는 것이다. 다만, R2RML의 성격이 매핑 규칙 정의 언어이기 때문에 R2RML을 사용해 사용자가 매핑 규칙을 자유롭게 정의할 수 있음은 물론 매핑에 가용할 소스 데이터의 범위 또한 조정할 수 있다는 차이점이 있다. 2020년 발표된 RML은 매핑의 소스가 RDB는 물론 CSV, XML, JSON 포맷도 가능하도록 R2RML을 확장한 매핑 규칙 정의 언어다.

본 논문은 RML을 기반으로 하지만 Direct Mapping 혹은 R2RML을 기반으로 본 논문처럼 매핑의 결과인 RDF 그래프에 대한 스키마를 자동 생성하기 위한 목적으로 수행된 기존 연구들을 먼저 살펴보고자 한다. [18-21]은 Direct Mapping에 기반한 연구들이다. [18-19]에서는 스키마를 SHACL로 출력하며 [20-21]에서는 ShEx로 스키마를 출력한다. [18]과 [20]는 스키마 생성을 위한 알고리즘만을 제안했으며 [19]와 [21]은 스키마를 생성해주는 시스템을 제안했다. [18-21]은 모두 스키마를 구성하는 제약 조건들을 생성하기 위한 입력으로 Direct Mapping을 구성하는 매핑 규칙들뿐만 아니라 매핑 소스 RDB의 스키마도 함께 사용함으로써 검증기를 통한 그래프 구조 검증이 가능한 수준의 스키마를 제공한다. [22-24]는 R2RML에 기반한 연구들이다. [22]는 스키마 생성을 위한 알고리즘만을 제안했다. [22]에 의하면 제안된 알고리즘은 스키마 표현 언어로 SHACL를 선택하든 ShEx를 선택하든 모두 적용 가능하다고 밝히고 있다. [23]과 [24]는 스키마 생성 시스템을 제안했다. [23]은 SHACL로 스키마를 출력하며

[24]는 ShEx로 스키마를 출력한다. [22-24]는 모두 그래프 구조 검증이 가능한 수준의 스키마를 생성하며 이를 위해 R2RML 스크립트에 기술된 매핑 규칙들과 RDB 스키마를 함께 입력으로 사용한다.

[25]는 현재 발표된 RML 기반의 유일한 연구로서 RML을 개발한 조직에 의해 수행된 것이다. [25]는 스키마를 SHACL로 출력하는 시스템을 제안했다. [25]가 생성하는 스키마는 사람에게 RML 매핑으로 생성될 그래프에 대한 구조 정보를 제공할 목적에 적합하다고 판단된다. 그 근거는 자동화된 검증을 위한 쓰임새까지는 고려하지 않은 다음 네 가지의 스키마 구성을 보이기 때문이다.

첫째, [25]는 검증기가 필수로 요구하는 제약 조건인 cardinalty의 값을 기본값인 '0 이상'만 스키마에 사용하고 있다. cardinality는 타겟 노드를 중심으로 형성될 수 있는 트리플 구조마다 부여해야 하고 각각의 구조가 몇 회 출현 가능한지를 제한한다. [25]는 스키마 생성을 위한 입력으로 RML 문서의 매핑 규칙만 사용하기 때문에, 입력 소스의 양태를 반영한 cardinality 값을 구할 수 없으므로 대신 기본값을 할당한 것이다. cardinality가 모두 '0 이상'인 셰이프로 검증을 수행하면 타겟 노드 그 자체만 검증 대상이 되며 타겟 노드를 둘러싼 트리플 구조는 검증에서 제외되는 부작용을 낳는다. 각각의 트리플 구조가 모두 0회 발생하는 것을 허용하기 때문이다.

둘째, [25]는 "open" 방식의 검증을 발동시키는 스키마를 생성한다. SHACL과 ShEx는 모두 "open" 방식과 "closed" 방식의 검증을 지원한다. "open" 방식은 타겟 노드가 셰이프에 명시된 구조만 만족하면 셰이프에 명시되지 않은 다른 구조의 트리플에 추가적으로 속해 있더라도 이를 허용하는 반면 "closed" 방식은 이를 허용하지 않고 오류를 발생시킨다. 특정 RML 매핑에 의해 그래프가 올바르게 생성되었는지를 검사할 목적으로는 "closed" 방식이 적합하다.

셋째, [25]는 "repeated properties" 구조의 셰이프가 검증기에서 오류를 유발하지 않도록 하기 위한 추가적인 구문 처리가 되어있지 않다. "repeated properties"는 똑같은 술어 제약 조건이 여러 번 사용된 셰이프를 일컫는다. 이러한 셰이프는 타겟 노드가 자신에 연결된 술어는 동일하나 그 술어에 연결된 나머지 노드의 형상만 상이한 여러 트리플들에 함께 포함되어 있는 구조를 제약하고자 할 때 만들어진다. 따라서 이러한 목적의 셰이프에는 술어 제약 조건은 같지만 나머지 노드에 대한 제약 조건이 상이하게 기술된 여러 개의 제약 조건 쌍이 정의되어 있다. SHACL은 "repeated properties" 구조의 셰이프에 대해 술어 제약 조건이 반복되지 않은 일반적인 셰이프들과는 다른 구문 형식을 요구한다. SHACL 검증기의 기본적인 동작 방식이 술어 제약 조건이 반복되지 않은 셰이프에 맞추어져 있어서 요구된 구문 형식을 준수하지 않으면 동작 방식이 변경되지 않아 "repeated properties" 구조의 셰이프는 늘 검증에 실패하게 된다. SHACL 검증기의 기본적인 동작 방식에서는 하나의 술어·나머지 노드 제약 조건 쌍을 만족해야 하는 트리플을 타겟 노드에 연결된 술어가 술어 제약 조건을 만족하는 모든 트리플들로 선정해 버리기 때문에 술어 제약 조건은 같지만 나머지 노드에 대한 제약 조건이 제각각인 "repeated properties" 구조의 셰이프에서는 이런 방식으로 선정된 트리플들이 하나의 나머지 노드 제약 조건을 모두 만족시킬 수는 없으므로 검증은 실패하게 된다.

넷째, [25]는 서로 다른 triples map으로 생성한 트리플들이 합쳐진 구조에 대한 대비가 취약하다. RML 문서는 tripls map이라는 것을 단위로 구성된다. 하나의 triples map에 의해 생성된 트리플들은 동일한 구조를 갖는다. 이러한 이유로 [25]는 하나의 triples map으로부터 하나의 셰이프를 생성한다. 그러나 RML 문서 작성자의 의도에 의해서건 아니면 입력 데이터의 양태로 인한 우연에 의해서건 서로 다른 triples map으로 생성한 트리플들에 동일한 노드가 동시에 존재하게 되어 이 트리플들이 합쳐지는 구조가 발생할 수 있다. [25]는 이러한 확장된 구조를 검증하기 위한 추가적인 셰이프를 제공하지 않는다. [25]의 스키마는 "open" 방식의 검증 방식을 발동시키므로 일반적인 경우에서는 타겟 노드에 새로운 구조의 트리플이 추가되더라도 검증이 실패하지는 않는다. 하지만 추가된 트리플로 인해 "repeated properties" 구조의 셰이프를 요하게 되는 상황에서는 검증이 실패하게 되는 한계가 존재한다.

III. The Proposed System

1. System Overview

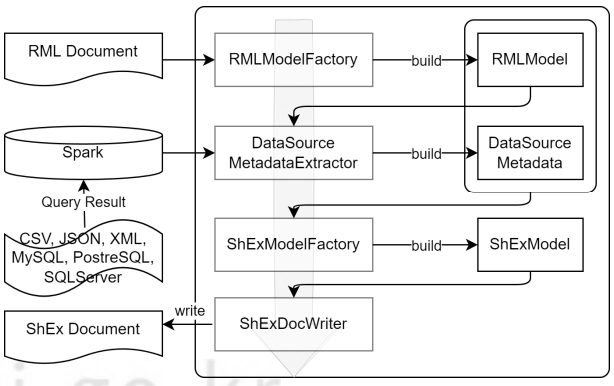

Fig. 1. System Overview

그림 1은 제안된 시스템의 동작 흐름을 요약한다. 동작 흐름은 4단계 처리로 나뉜다. 1단계에서는 RML 문서를 파싱하여 메모리에 RML 모델을 구축한다. 이 작업은 1단계 처리기인 RMLModelFactory가 수행한다. RML 모델은 RML 문서의 내용을 그대로 객체 모델화시킨 것이다. RML 모델이 제공하는 API로 인해 이후 단계의 처리기들은 RML 문서의 탐색과 필요한 요소값 획득을 용이하게 수행할 수 있게 된다. 2단계에서는 매핑에 사용될 입력 데이터의 양태를 분석하여 제약 조건 생성을 위해 필요한 메타데이터를 추출한다. 성공적인 2단계 처리를 위해서는 RDB, CSV, JSON, XML에서 온 대용량 데이터를 일관된 방식으로 함께 연관 지어 분석할 수 있는 환경이 필요하다. 오픈 소스 분산 데이터 분석 엔진인 Apache Spark는 이러한 요구를 충족하기 때문에 2단계 처리기인 DataSourceMetadataExtractor는 Spark에 이기종 입력 데이터를 모두 로딩시킨 후 Spark가 제공하는 API를 사용해 로딩된 이기종 데이터를 일관된 방식으로 분석을 수행하는 구조를 취한다. 3단계에서는 1~2단계의 출력인 RML 모델과 데이터 소스 메타데이터를 입력으로 취하여 ShEx 스키마를 프로그래밍 객체 모델 형태로 생성한다. 이 작업은 3단계 처리기인 ShExModelFactory가 수행한다. 4단계에서는 ShEx 모델을 단순히 ShExC 신택스로 표현한 ShEx 문서를 생성한다. 4단계 처리는 ShExDocWriter가 담당한다. 4단계를 위하여 ShEx 모델을 구성하는 각각의 객체에는 모두 자신에 대응하는 ShExC 구문을 출력하는 기능이 내장되어 있으며 구문 출력 기능들은 ShExC의 구문 구조 규칙에 따라 조직되어 있다. 따라서 4단계 처리기는 최상위 레벨 구조에 해당하는 구문 출력 기능만을 호출하여 전체 스키마를 출력한다.

2. RML Model Details

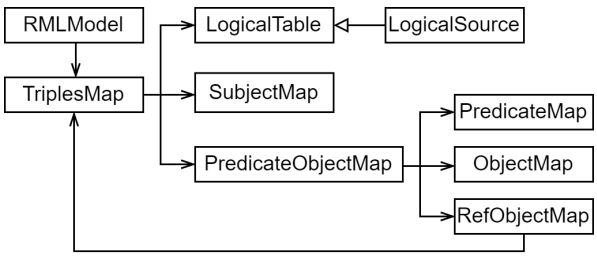

Fig. 2. Major Classes of RML Model

그림 2는 RML 모델을 구성하는 주요 클래스들의 관계를 보여준다. RML 모델을 구성하는 클래스들의 이름과 그들 간의 관계는 RML 사양에서의 그것과 동일하다. 매핑을 위해 작성한 RML 문서에는 RML 사양이 정의한 클래스들에 대한 인스턴스들이 기술되어 있는 것으로 볼 수 있으며 1단계 처리기는 이 인스턴스들을 RML 모델의 객체로 1:1 대응시켜 옮긴다.

RML 모델은 triples map들의 모음이다. 하나의 triples map이 동일한 구조의 트리플들을 생성한다. triples map 하나는 트리플 생성 규칙과 이 규칙을 적용할 입력 데이터 정의로 이루어진다. 데이터는 행렬 구조로 정의된다. logical table은 RDB로부터만 데이터를 정의할 수 있지만 logical source는 RML이 지원하는 모든 포맷으로부터 정의할 수 있다. logical table은 R2RML 사양의 것이지만 이를 RML 사양에서 데이터 소스 종류를 다양화하기 위하여 확장한 것이 logical source기 때문이다. 트리플 생성 규칙은 주어 생성 규칙(subject map)과 술어·목적어 생성 규칙(predicate object map)이 분리돼 정의된다. 술어·목적어 생성 규칙은 다시 술어 생성 규칙(predicate map)과 목적어 생성 규칙으로 구성되며 목적어 생성 규칙은 object map 혹은 referencing object map 중 하나다. object map은 새로운 목적어 생성을 위한 규칙을 담고 있으며 referencing object map은 다른 triples map에 의해 생성된 주어를 목적어로 취하기 위한 규칙을 담고 있다. 술어·목적어 생성 규칙은 한 triples map에 여러 개 정의될 수 있다. 트리플 생성 규칙은 입력 데이터의 행마다 적용된다. 각 행에서 1개의 주어가 생성되며 이 주어는 같은 행에서 술어·목적어 생성 규칙 수만큼 생성된 술어·목적어 쌍의 공통된 주어가 된다. 어떤 행에서 주어 생성에 필요한 열 데이터가 null이면 그 행에서는 주어뿐 아니라 모든 술어·목적어 쌍이 생성되지 않으며 술어 혹은 목적어 생성에 필요한 열 데이터만 null이면 해당 술어·목적어 쌍만 생성되지 않는다.

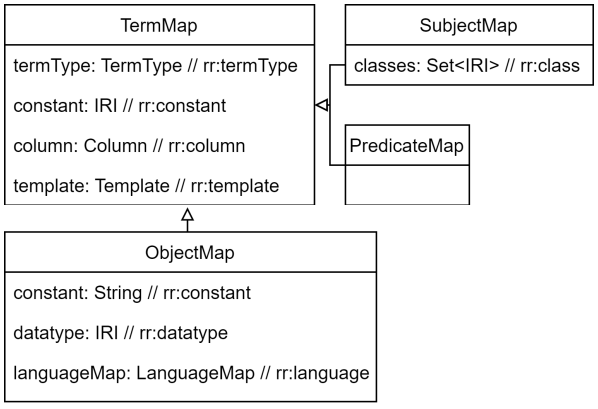

Fig. 3. TermMap and its subclasses

그림 3을 통해 주어, 술어, 목적어를 새로 생성하기 위한 규칙이 어떤 속성들로 구성되는지를 설명하고자 한다. 그림 3의 클래스 간 관계 또한 RML 사양에서 정의한 관계를 RML 모델로 옮긴 것이다. 그림 3에서 클래스의 속성은 '속성명: 타입' 형식으로 표현되었으며 그 우측 주석은 대응하는 RML 문서에서의 속성명이다. 속성 타입으로 사용된 TermType, IRI, Column, Template, LanguageMap은 그림 2에 표시되지 않았으나 모두 RML 모델의 클래스들이다. 그림 3을 보면 SubjectMap, PredicateMap, ObjectMap이 모두 TermMap 이라는 클래스를 상속하고 있다. 이것은 TermMap에 정의된 4개의 속성이 주어, 술어, 목적어 생성 규칙 어디서나 사용될 수 있는 공통 속성임을 뜻한다. 속성 termType의 값은 생성될 항이 어떤 종류에 속하는지를 명세한 결과다. RDF 문법에 따라 이 속성의 값으로 주어는 IRI 혹은 블랭크 노드, 술어는 IRI, 목적어는 IRI 혹은 블랭크 노드 혹은 리터럴이 가능하다. 각 항의 구체적 생성 방법은 세 가지 중 하나를 선택할 수 있으며 그 선택에 따라 constant, column, template 중에서 하나의 속성만 값을 갖게 된다. 첫 번째는 생성될 항의 구체적인 값을 명시하는 방법이다. 이 방법은 입력 데이터를 항 생성에 사용하진 않으나 생성 규칙이 행마다 적용되기 때문에 행 수만큼 모두 같은 명시된 값을 생성한다. 두 번째는 입력 데이터의 특정 컬럼명 하나를 명시하는 방법이다. 이 방법은 각각의 행에 속한 명시된 컬럼의 값을 그대로 생성될 항의 값으로 취한다. 세 번째는 템플릿 문자열을 명시하는 방법이다. 이 문자열에는 입력 데이터의 컬럼명이 1개 이상 포함될 수 있다. 이 방법은 템플릿 문자열 내 컬럼명을 각각의 행에 속한 대응하는 컬럼의 값으로 치환하여 항으로 취한다. ObjectMap의 3가지 속성은 목적어 종류가 리터럴일 경우에만 값을 가질 수 있다. 목적어만이 항의 종류가 리터럴일 수 있기 때문이다. ObjectMap의 constant는 리터럴 상수가 명시되었을 때 값을 갖는다. datatype은 생성될 리터럴의 데이터 타입을 의미한다. languageMap은 언어 태그가 값이다. 언어 태그는 RDF의 데이터 타입이 문자열인 리터럴 표기법 중 하나인 'language-tagged 리터럴'에 사용되며 해당 문자열이 어느 언어인지를 표시하는 역할로 사용된다. SubjectMap의 classes 속성은 rr:class의 값으로 나열된 여러 IRI를 집합으로 저장한다. 이 속성이 사용되면 해당 subject map으로 생성된 모든 주어에 대하여 술어로서 'rdf:type', 목적어로서 속성값으로 명시된 각각의 클래스 역할 IRI를 취하는 트리플들이 추가된다.

3. Metadata Extraction from Data Sources

Table 1. Metadata Map

Key (RML Model Object)	Value (Metadata)
predicate-object pair	cardinality
object map generating literals with rr:column	data type
object map generating numeric literals with rr:column	value range
object map generating string literals with rr:column	length range
term map with rr:template	length range

(a)
```
{ 'User': 'Rachel',
  'Info': { 'ID': 'RML2ShEx', 'Email': 'who@foo.com' },
  'Friends': [ 'John', 'Emily' ] }
```

(b)

User	Info	Friends
Rachel	{ 'ID': 'RML2ShEx', 'Email': 'who@foo.com' }	['John', 'Emily']

(c)

User	Info.ID	Info.Email	Friends
Rachel	RML2ShEx	who@foo.com	John
Rachel	RML2ShEx	who@foo.com	Emily

Fig. 4. Example of Flattening for JSON Data Set

2단계 처리에서는 표 1의 두 번째 열에 표시된 4가지 종류의 메타데이터를 입력 데이터로부터 얻어낸다. 이를 위해 2단계 처리기는 우선 사용자가 logical table 혹은 logical source에 명시한 질의문을 실행시켜 획득한 행렬 구조의 데이터를 Spark에 로딩한다. CSV는 질의문 대신 파일명이 명시되므로 명시된 파일을 로딩한다. CSV는 파일 자체가 이미 행렬 구조기 때문이다. RDB는 RDBMS에서 처리된 SQL 질의 결과를 로딩한다. JSON과 XML을 위한 질의어인 JSONPath와 XPath는 Spark가 지원하지 않기 때문에 제안된 시스템에서 두 질의어에 대한 질의문 처리를 직접 수행한 후 그 결과를 로딩한다. JSON과 XML 데이터는 로딩 후 평탄화 처리를 거친 다음 분석에 사용된다. 그림 4는 평탄화 과정을 요약한다. 그림 4의 (a)는 JSONPath 질의 결과 셋의 한 행에 해당하고 (b)는 이 데이터가 Spark에 로딩된 모습이며 (c)는 (b)를 평탄화시킨 결과다. RML 매핑에서 컬럼으로 취급되는 JSON과 XML 데이터의 속성은 그 값으로 Info 속성처럼 객체, Friends 속성처럼 배열일 수 있다. 심지어 속성값인 객체 혹은 배열 속에 다시 객체 혹은 배열이 내포될 수 있으며 내포의 깊이에 제한은 없다. 그림 4는 JSON에 대한 사례지만 XML도 동일한 내포 특성을 보인다. 평탄화는 (a)처럼 객체 혹은 배열이 내포된 입력 데이터의 한 행이 RML 프로세서에 의한 매핑 시 내포가 제거된 (c) 구조의 여러 행으로 취급되어 여러 번의 매핑 규칙 적용이 발생하기 때문에

정확한 cardinality 메타데이터 추출을 위해 수행한다. 평탄화는 컬럼의 값이 객체면 그 컬럼을 객체의 각각의 속성에 대응시켜 새로 생성한 컬럼들로 치환시키며 컬럼의 값이 배열이면 각각의 행이 배열 요소 개수가 되도록 복제한 후 배열 요소 각각을 해당 컬럼의 한 행의 값으로 배치한다. 이 과정은 내포의 깊이에 제한이 없는 점을 고려하여 재귀적인 논리로 구현되어 있다.

cardinality는 술어·목적어 생성 규칙마다 구한다. cardinality는 해당 술어·목적어 생성 규칙으로 생성한 술어·목적어 쌍이 하나의 주어에 최소·최대 몇 개 연결되는지를 뜻한다. 목적어 생성 규칙은 object map 혹은 referencing object map이다. 목적어 생성 규칙이 무엇인가에 따라 cardinality를 구하는 절차에 차이가 있다. object map인 경우의 cardinality를 얻는 절차를 간략히 기술하면 다음과 같다. 데이터를 트리플 생성에 사용될 컬럼들로만 축소하고 주어 생성 컬럼들에 null이 포함된 행과 중복된 행은 제거한다. 주어 생성 컬럼들을 기준으로 그룹을 형성한 다음 그룹별 행 수를 구한다. 이때 주어 생성 컬럼이 아닌 컬럼들에 null이 포함된 행이 있다면 그 행의 수만큼 구해진 행 수에서 빼준다. 그룹별 행 수의 최솟값, 최댓값을 해당 술어·목적어 생성 규칙에 대한 cardinality로 취한다. referencing object map에 대한 cardinality를 구할 때는 해당 referencing object map이 소속된 triples map의 입력 데이터뿐만 아니라 referencing object map이 참조하는 triples map의 입력 데이터도 함께 사용한다. RML에서는 전자의 데이터를 child 후자의 것을 parent라 한다. cardinality는 child와 parent를 조인하여 하나의 테이블로 합친 후 구한다. 조인에 사용할 컬럼들은 referencing object map에 명시된 것을 사용한다. child와 parent는 조인 전 각각 자신의 주어 생성에 사용될 컬럼들과 조인에 사용될 컬럼들만으로 구성되며 중복된 행과 주어 생성 컬럼들에 null이 포함된 행은 제거된다. child의 행들이 조인 후에도 소실되지 않도록 하는 outer join을 수행한 다음 child의 주어 생성 컬럼들을 기준으로 그룹을 짓고 그룹별 카운트를 계산한다. 이때도 child의 주어 생성 컬럼이 아닌 컬럼에 null이 포함된 행은 카운트에서 빼준다. 이후 그룹별 카운트의 최솟값·최댓값을 cardinality로 취한다. 상기한 child를 중심으로 한 조인과 그룹별 카운팅를 기반으로 구한 cardinality는 child의 한 행이 parent의 몇 개의 행을 참조하는지에 대한 범위 정보다. 제안된 시스템은 parent의 한 행이 child의 몇 개의 행에 의해서 참조되는지에 대한 범위 정보도 함께 산출한다. 이를 위해 상기한 조인과 그룹별 카운팅을 parent를 중심으로 한 번 더 수행한다. cardinality를 제외한 세 가지 메타데이터는 표 1의 1열에 표시된 바와 같이 리터럴을 생성하는 object map 혹은 템플릿 문자열을 갖는 term map 각각에 대해 생성된다. 이 메타데이터들은 rr:column에 명시된 혹은 rr:template 내에 포함된 컬럼 각각을 대상으로 하여 구한다. rr:column에 명시된 컬럼의 경우 데이터 타입을 값 탐색에 의해 결정한 다음 숫자형이면 최솟값·최댓값을 추가적으로 취하고 문자형이면 길이의 최솟값·최댓값을 추가적으로 취한다. rr:template에 포함된 컬럼의 경우 본래의 데이터 타입과 상관없이 문자형으로 간주하여 길이의 최솟값·최댓값만을 취한다. 표 1은 이 단계에서 구한 메타데이터가 대응하는 RML 모델 객체를 키로 하는 맵 구조로 저장됨을 보여준다. 이후 단계의 처리기인 ShEx 모델 생성기가 RML 모델 객체로 메타데이터를 용이하게 획득할 수 있도록 하기 위함이다.

4. ShEx Model Details

```
ShExModel = { declarableShapeExprs: Set<DeclarableShapeExpr> }

ShapeExpr = DeclarableShapeExpr | ShapeExprRef
DeclarableShapeExpr = { id: IRI }
ShapeExprRef = { shapeExprLabel: IRI }
DeclarableShapeExpr = NodeConstraint | Shape | ShapeAnd | ShapeOr
NodeConstraint = { nodeKind: NodeKinds, values: Set<ValueSetValue>,
                   datatype: IRI, xsFacets: Set<XSFacet> }
Shape = { closed: Boolean, expression: TripleExpr}
ShapeAnd = { shapeExprs: Set<ShapeExpr> }
ShapeOr = { shapeExprs: Set<ShapeExpr> }

ValueSetValue = Language | ObjectValue
Language = { languageTag: String }
ObjectValue = IRIREF | ObjectLiteral
IRIREF = { value: IRI }
ObjectLiteral = { value: String }

XSFacet = StringFacet | NemericFacet
StringFacet = { pattern:String, flags:String,
                stringLength:StringLength, INTEGER: Integer }
NumericFacet = { numericRange:NumericRange, numericalLiteral: String }


TripleExpr = DeclarableTripleExpr | TripleExprRef
DeclarableTripleExpr = { id: IRI }
TripleExprRef = { tripleExprLabel: IRI }
DeclarableTripleExpr = TripleConstraint | EachOf
TripleConstraint = { inverse: Boolean, predicate: IRI, valueExpr: ShapeExpr,
                     min: Integer, max: Integer }
EachOf = { expressions: Set<TripleExpr> }

NodeKinds = { IRI, BNODE, LITERAL }
StringLength = { LENGTH, MIN_LENGTH, MAX_LENGTH }
NumeicRange = { MIN_INCLUSIVE, MIN_EXCLUSIVE,
                MAX_INCLUSIVE, MAX_EXCLUSIVE }
```

Fig. 5. Structure of ShEx Model

그림 5에는 ShEx 모델을 구성하는 모든 클래스들의 속성 그리고 클래스 간 상속 관계가 텍스트로 표현되어 있다. 그림 5는 ShExJ와 ShExC의 구문 요소 간 포함관계를 분석하여 객체 모델로 변환한 결과다. ShEx 사양은 단지

두 신택스에 대한 문법만 제공할 뿐 객체 모델 형태로 구문 요소 간 관계를 제시해 주진 않기 때문이다. 그림 5의 모든 클래스명과 속성명은 문법에서 사용된 명칭을 따랐다. 그림 5에서 좌변은 모두 클래스다. 좌변 클래스를 상속하는 클래스들이 존재하면 이들을 우변에 기호 '|'으로 구분해 나열하였다. 좌변 클래스의 속성은 중괄호 안에 '속성명: 타입' 형식으로 나열하였다. 열거형인 좌변 클래스는 우변 중괄호 안에 대문자로만 구성된 상수들을 나열하였다. 캐주얼한 용어로서의 셰이프에 대응하는 클래스는 ShapeExpr이고 셰이프들의 집합인 ShEx 모델은 싱글톤인 ShExModel 객체에 대응한다.

1. TriplesMap 객체 클러스터링
2. TriplesMap 객체와 참조 id 쌍 생성
3. SubjectMap 객체로부터 NodeConstraint 객체 생성
4. SubjectMap 객체의 classes 속성 / Predicate-ObjectMap 객체 쌍 / Predicate-RefObjectMap 객체 쌍 } 으로부터 TripleConstraint 객체 생성
5. NodeConstraint 객체와 TripleConstraint 객체들을 ShapeAnd 객체로 결합
6. 클러스터 멤버 간 조합당 ShapeAnd 객체 생성
7. 참조 용도의 ShapeOr 객체 생성

Fig. 6. Process to Build a ShEx Model

그림 6은 ShEx 모델을 구축하기 위한 처리 과정을 요약한다. 첫 번째 처리는 서로 다른 triples map에서 동일한 주어 노드가 생성되는 경우에 대한 대비의 일환으로서 주어 생성 규칙이 동일 혹은 동등하거나 포함관계에 있는 TriplesMap 객체끼리 군집을 형성시킨다. 같은 군집에 속하기 위해서는 TriplesMap의 termType은 반드시 같아야 한다. constant가 같으면 동일 관계로 판단한다. template의 문자열 패턴이 같으면 동등 관계로 판단한다. template의 문자열 패턴은 컬럼값으로 치환되는 부분을 제외한 나머지 고정된 문자열을 뜻한다. 한 TriplesMap의 constant가 다른 TriplesMap의 template의 문자열 패턴에 매칭되면 포함관계로 판단한다. 한 군집에 여러 TriplesMap들이 속하게 되면 이들은 같은 주어를 생성할 가능성이 있는 것들로 다루어진다.

두 번째 처리에서는 각각의 TriplesMap마다 참조 id란 것을 임의로 생성하여 짝지어 준다. 참조 id는 각각의 TriplesMap이 RefObjectMap의 부모 TriplesMap으로 사용될 경우를 대비하여 만든 것이다. RefObjectMap을 ShEx 모델로 변환할 때 부모 TriplesMap은 그에 짝지어진 참조 id로 변환된다. 참조 id는 그림 6의 3, 5, 7번 처리에서 생성할 NodeConstraint 혹은 ShapeAnd 혹은 ShapeOr 객체의 id가 된다. TriplesMap이 주어 생성 규칙만 있고 다른 TriplesMap과 같은 주어를 생성할 가능성이 없는 것이면 이 TriplesMap에 대응된 참조 id는 해당 TriplesMap의 주어 생성 규칙의 ShEx로의 변환 결과인 NodeConstraint 객체의 id가 될 것이다. TriplesMap이 다른 TriplesMap과 같은 주어를 생성할 가능성은 없으나 최소 하나 이상의 술어·목적어 쌍을 생성케 하는 규칙을 보유한 것이라면 이 TriplesMap에 대응된 참조 id는 해당 TriplesMap의 트리플 생성 규칙의 ShEx로의 변환 결과인 ShapeAnd 객체의 id가 될 것이다. TriplesMap이 다른 TriplesMap과 동일한 주어를 생성할 가능성이 있는 것이라면 이 TriplesMap에 대응된 참조 id는 해당 TriplesMap으로 생성된 트리플이 다른 TriplesMap으로 생성된 트리플과 합쳐질 수 있는 모든 가능한 구조 각각에 대응하는 셰이프를 OR 연산자로 결합한 ShapeOr 객체의 id가 될 것이다.

세 번째 처리는 RML 모델의 TriplesMap 객체마다 1개씩 존재하는 SubjectMap 객체 각각으로부터 1개씩의 NodeConstraint 객체를 생성한다. NodeConstraint 객체는 주어 혹은 목적어 노드를 위한 제약 조건이다. 물론 SubjectMap 객체로부터 생성한 NodeConstraint 객체는 주어 제약 조건이 된다. 이 과정에서 SubjectMap의 termType, constant, template 속성값은 각각 NodeConstraint의 nodeKind, values, xsFacets 속성값으로 사용된다. 구체적으로 termType 값에 대응하여 nodeKind에는 IRI 혹은 BNODE가 할당되며 constant 속성값은 IRIREF의 value 속성에 할당되어 values의 한 원소로 추가되며 template 속성값은 ShEx에서 사용하는 정규식으로 변환되어 StringFacet의 pattern 속성에 할당된 후 xsFacets의 한 원소로 추가된다. 이때 템플릿 문자열 내 컬럼명 부분은 메타데이터 중 하나인 해당 컬럼값의 문자열 길이 범위로 치환된다.

네 번째 처리는 술어·목적어를 생성케 하는 그림 6에 제시된 세 종류의 RML 모델 요소 각각으로부터 ShEx의 술어·목적어 제약 조건인 TripleConstraint 객체를 생성한다. TripleConstraint의 predicate 속성에는 술어 IRI가 할당되며 목적어 제약 조건인 valueExpr 속성에는 새로 생성한 NodeConstraint 객체가 할당된다. 목적어 제약 조건이 SubjectMap 객체의 classes 속성에서 비롯된 경우는 classes 속성값이 모두 IRI 상수이므로 이들은 각각 IRIREF 객체의 value 속성에 할당된 뒤 NodeConstraint 객체의 values 원소로 등록된다. 목적어 제약 조건이 ObjectMap 객체에서 비롯됐다면 NodeConstraint의 모든 속성이 사용될 수 있다. ObjectMap의 termType이 IRI 혹은 블랭크 노드일 때는 주어 제약을 위한

NodeConstraint 객체를 생성할 때와 같지만 리터럴일 때는 datatype 속성도 사용된다. 이 속성에는 RML 문서에 명시된 데이터 타입을 우선으로 할당하나 그것이 없다면 메타데이터로 획득한 데이터 타입을 할당한다. 그리고 리터럴 목적어인 경우 다양한 ValueSetValue 하위 타입 객체들이 values 원소로 등록되며 다양한 XSFacet 하위 타입 객체들이 xsFacets 원소로 등록된다. ValueSetValue 하위 타입으로는 언어 태그가 있다면 Language, 상수가 있다면 ObjectLiteral이 사용되며 XSFacet 하위 타입으로는 문자형 리터럴이면 StringFacet, 숫자형 리터럴이면 NumericFacet이 사용된다. XSFacet 하위 타입의 값은 숫자 범위, 문자열 길이 범위 메타데이터를 가져와 할당한다. 술어·목적어 생성 규칙이 RefObjectMap 객체인 경우에는 2개의 TripleConstraint 객체를 생성한다. 생성될 TripleConstraint 객체의 predicate 속성값은 둘 다 술어 생성 규칙이 명시한 술어 IRI로 같고 inverse 속성값은 하나는 true이고 다른 하나는 false이다. inverse가 true인 것의 valueExpr 속성에는 부모 TriplesMap에 대응하는 참조 id가 할당되며 inverse가 false인 것의 valueExpr에는 자식 TriplesMap에 대응하는 참조 id가 할당된다. inverse가 false인 것은 자식 triples map이 생성한 트리플의 주어가 목적어로서 부모 triples map이 생성한 트리플의 주어를 참조할 수 있음을 뜻하고 inverse가 true인 것은 반대로 부모 triples map이 생성한 트리플의 주어가 자식 triples map이 생성한 트리플에서 목적어로 참조될 수 있음을 뜻한다. TripleConstraint의 min과 max는 cardinality에 대한 속성이다. 술어·목적어 생성 규칙으로 생성된 경우는 메타데이터 저장소에서 검색된 값을 사용하며 SubjectMap의 classes 속성으로 생성된 경우는 classes의 원소 개수가 cardinality 값으로 결정된다. classes 원소별로 트리플이 하나씩 생성되기 때문이다.

다섯 번째 처리는 같은 TriplesMap 객체로부터 생성된 주어 검증용 NodeConstraint 객체와 술어·목적어 검증용 TripleConstraint 객체들을 하나의 ShapeAnd 객체로 결합시킨다. 이 ShapeAnd들은 주어·술어·목적어 검증을 위한 모든 제약 조건이 구비된 셰이프가 되며 이 처리로 RML 모델에 등록된 각각의 TriplesMap에 대응된 하나씩의 셰이프 생성은 완료된다. TripleConstraint 객체들은 NodeConstraint 객체와의 결합 전에 먼저 EachOf 객체의 expressions 속성의 원소로 모두 등록되며 이 EachOf 객체는 다시 Shape의 expression에 할당된다. 결국, ShapeAnd 객체는 NodeConstraint 객체 하나와 이 Shape 객체 하나를 결합한 것이다. EachOf는 검증 노드와 연결된 구조가 EachOf 원소의 제약 조건 각각을 모두 만족시켜야 한다는 의미의 구문을 첨가하게 한다. 그리고 EachOf를 품는 Shape의 closed 속성은 true로 세팅되며 이로 인해 EachOf에 대한 'closed' 방식의 검증을 발동시킨다.

여섯 번째 처리에서는 서로 다른 triples map에서 같은 주어가 생성되어 트리플들이 합쳐진 구조를 검증할 수 있는 셰이프들을 생성한다. 합쳐지는 경우 각각을 구하기 위해 앞서 생성해 둔 TriplesMap들의 군집들 중에서 멤버가 복수인 각각의 군집을 대상으로 멤버간 가능한 모든 조합을 구한다. 구해진 조합별로 조합을 구성하는 멤버 TriplesMap에 대응하여 생성한 셰이프들을 결합시킨 새로운 ShapeAnd 객체를 하나씩 생성한다. 이로써 트리플이 합쳐지는 가능한 모든 구조 각각을 대비한 셰이프 생성이 완료된 것이다. 결합된 셰이프도 결국 주어 제약 조건과 술어·목적어 제약 조건들의 결합이다. 결합된 셰이프의 주어 제약 조건은 조합 내 NodeConstraint들 중 포함 관계를 고려하여 가장 넓은 범위의 것이 선택된 것이고 술어·목적어 제약 조건은 조합 내 셰이프의 중복이 제거된 모든 TripleConstraint들의 모음이다.

일곱 번째 처리에서는 TriplesMap들의 군집들 중에서 멤버가 복수인 군집에 속한 TriplesMap 각각에 대응하여 ShapeOr 객체를 하나씩 생성한다. ShapeOr는 해당 TriplesMap으로 생성한 트리플이 다른 TriplesMap이 생성한 트리플과 합쳐질 수 있는 각각의 구조에 대비하여 생성한 셰이프들을 논리 연산자 OR로 결합한 것이다. 결합될 셰이프는 해당 TriplesMap에 대응하여 생성된 NodeConstraint 혹은 ShapeAnd 객체와 해당 TriplesMap이 속한 군집에서 구한 조합 중 해당 TriplesMap이 속한 조합에 대응하여 생성된 ShapeAnd들이다. ShapeOr는 셰이프들의 id만을 취하여 OR로 결합한다. 이는 구문의 간결성을 고려한 선택이다. ShapeOr는 자신에 대응하는 TriplesMap으로 생성된 주어가 다른 트리플에서 목적어로 참조될 때 이 주어가 어떠한 구조로 합쳐졌든 결합된 구조 중 하나기 때문에 검증 시 오류를 유발하지 않게 한다.

5. Example of RML-to-ShEx Transformation

그림 7~9는 하나의 RML 매핑 사례를 구성한다. 그림 7~9는 순서대로 매핑의 입력 데이터, RML 매핑 규칙, 매핑의 결과인 RDF 그래프다. 이 사례는 RML 매핑 규칙으로부터 SHACL 스키마를 생성해주는 연구 [25]가 갖는 한계를 집약적으로 보이기 위해 고안된 것이다. 그림 10은 이 사례의 매핑 규칙과 입력 데이터로부터 제안된 시스템이 생성한 ShEx 스키마다. 본 절에서는 이 스키마를 통해 제안된 시스템이 기존 연구 [25]의 한계를 극복했음을 보이고자 한다.

```
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. 7. Input Data

그림 7은 세 개의 CSV 파일의 내용을 보여준다. 각 파일의 첫 번째 행은 헤더 행이다. 8은 세 개의 triples map으로 구성되어 있다. 7~19행, 20~26행, 27~33행이 순서대로 TriplesMap1, TriplesMap2, TriplesMap3로 명명된 triples map이다. 그림 8의 9행, 22행, 29행에서 각각의 triples map이 자신의 매핑 규칙을 적용할 데이터 소스를 지정하고 있다.

```
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: @prefix foaf: <http://xmlns.com/foaf/0.1/>.
06: @base <http://example.com/base/>.

07: <TriplesMap1> a rr:TriplesMap;

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

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

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

14:   rr:predicateObjectMap [
15:     rr:predicate <http://ex.com/practises> ;
16:     rr:objectMap [
17:       a rr:RefObjectMap ;
18:       rr:parentTriplesMap <TriplesMap2>;
19:       rr:joinCondition [ rr:child "Sport" ; rr:parent "ID" ; ] ] ] .

20: <TriplesMap2> a rr:TriplesMap;

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

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

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

27: <TriplesMap3> a rr:TriplesMap;

28:   rml:logicalSource [
29:     rml:source "sport_es.csv"; rml:referenceFormulation ql:CSV ];

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

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

Fig. 8. RML Document

```
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. 9. Result RDF Graph by Fig. 8

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

05: my:S0 @my:S4 OR @my:S5

06: my:S1 @my:S3 OR @my:S5

07: my:S2 IRI /^http:\/\/ex\.com\/student(.{2,})$/ AND CLOSED {
08:   ex:practises @my:S0 ?;
09:   foaf:name LITERAL MINLENGTH 10 MAXLENGTH 14
10: }

11: my:S3 IRI /^http:\/\/ex\.com\/sport(.{3,})$/ AND CLOSED {
12:   $my:T0 rdfs:label [@es] LENGTH 5
13: }

14: my:S4 IRI /^http:\/\/ex\.com\/sport(.{3,})$/ AND CLOSED {
15:   $my:T1 (
16:     ^ex:practises @my:S2;
17:     rdfs:label [@en] LENGTH 6
18:   )
19: }

20: my:S5 IRI /^http:\/\/ex\.com\/sport(.{3,})$/ AND CLOSED {
21:   &my:T0;
22:   &my:T1
23: }
```

Fig. 10. ShEx Document From Fig. 7~8 For Fig. 9

그림 9의 4~6행 트리플들은 TriplesMap1으로 생성된 것이다. 4~6행 트리플의 주어는 그림 8의 10행 주어 생성 규칙에 따라 ID 컬럼값을 사용해 생성되었다. 4행과 6행 트리플의 술어·목적어는 그림 8의 11~13행 술어·목적어 생성 규칙을 stduent.csv의 각각의 행에 적용해 생성된 것이다. 5행 트리플의 술어·목적어는 그림 8의 14~19행 술어·목적어 생성 규칙을 stduent.csv의 1행에 적용해 생성된 것이다. stduent.csv의 2행은 조인 컬럼인 Sport 값이 null이기 때문에 그림 8의 14~19행 술어·목적어 생성 규칙에 의한 트리플이 생성되지 않았다. 7행 트리플은 TriplesMap2로 생성되었으며 8행 트리플은 TriplesMap3로 생성되었다. 7~8행 트리플의 목적어는 둘 다 'language-tagged 리터럴'이다.

그림 10의 7~10행은 TriplesMap1에 대응하여 생성된 ShapeAnd 구문이다. my:S2는 id이고 7행의 id와 AND 사이는 주어 검증을 위한 NodeConstraint 구문이고

CLOSED부터 중괄호 부분은 술어·목적어 검증을 위한 Shape 구문이다. NodeConstraint 구문은 주어의 종류로 결정된 IRI와 문자열로서의 주어가 준수해야 하는 정규식으로 구성되어 있다. Shape 구문의 중괄호 내부는 EachOf 구문이며 세미콜론으로 구분된 각각의 행은 TripleConstraint 구문이다. my:S2와 같은 구조로 my:S3와 myS4는 각각 TriplesMap3과 TriplesMap2에 대응하여 생성된 ShapeAnd 구문이다. my:S5는 TriplesMap2와 TriplesMap3이 같은 주어를 생성할 수 있기 때문에 my:S3와 myS4를 결합하여 생성한 ShapeAnd 구문이다. my:S5의 &my:T0과 &my:T1은 my:S3와 my:S4의 EachOf 구문의 id 참조다. my:S0과 my:S1은 각각 TriplesMap2와 TriplesMap3에 대응하여 생성한 ShapeOr구문이다. 이 중 TriplesMap2는 TriplesMap1에서 부모 tripels map으로서 사용되었기 때문에 my:S0은 8행에서 ex:practises 술어의 목적어 제약 조건으로 참조되어 쓰였다.

그림 9의 트리플들을 그림 10의 검증한 결과는 ex:student10과 ex:student20이 주어인 트리플들은 my:S2의 주어·술어·목적어 제약 조건을 모두 만족시켜 검증에 성공하였고 ex:sport100이 주어인 트리플들은 my:S5의 주어·술어·목적어 제약 조건을 모두 만족시켜 검증에 성공하였다. ex:student10과 ex:student20이 주어인 트리플들은 연구 [25]가 생성하는 스키마에서도 검증에 성공하는 사례다. 하지만 주어가 제약 조건을 만족시켰기 때문에 성공한 것이다. 왜냐하면, 연구 [25]가 생성하는 스키마는 술어·목적어 제약 조건의 cardinality가 모두 “0 이상”이고 “open” 방식으로 검증을 수행하기 때문이다. 즉, 해당 주어에 연결된 술어·목적어가 아예 없거나 제약 조건에 명시되지 않은 술어·목적어가 연결되었어도 검증에 성공할 수 있었다는 뜻이다. ex:sport100이 주어인 트리플들은 연구 [25]에서는 검증에 실패하는 사례로서 TriplesMap2와 TriplesMa3이 같은 주어를 생성해 트리플이 합쳐진 경우에다가 각각의 triples map에서 생성한 트리플의 술어가 rdfs:label로 같으므로 “repeated properties” 구조에 해당한다. 이로 인해 “repeated properties” 구조에 대비하여 SHACL에서 요구한 별도의 구문 처리를 하지 않은 연구 [25]가 생성한 스키마에서는 이 두 트리플이 술어가 같다는 이유로 항상 함께 검증 대상으로 묶이기 때문에 언어 태그 “es”가 첨가된 목적어는 언어 태그가 “en”이어야 한다는 제약 조건을 만족시킬 수 없고 반대로 언어 태그 “en”이 첨가된 목적어는 언어 태그가 “es”여야 한다는 제약 조건을 만족시킬 수 없으므로 검증에 반드시 실패하게 된다. ShEx는 SHACL과 달리 술어가 같은 이유로 검증 대상을 하나로 묶지 않기 때문에 “repeated properties”를 위한 별도의 구문이 존재하지 않는다. 즉, 연구 [25]의 “repeated properties”에 대한 문제는 ShEx를 사용한 것 그 자체로 해결된 것이다. ex:sport100을 my:S5의 타겟 노드로 설정하면 그림 9의 7행 술어·목적어는 my:T1의 참조를 따라가 그림 10의 17행 조건으로 검증되고 그림 9의 8행 술어·목적어는 my:T0의 참조를 따라가 그림 10의 12행 조건으로 검증된다. 이에 더하여 그림 10의 16행도 my:T1에 속하기 때문에 ex:sport100이 목적어로 사용된 그림 9의 5행 트리플도 함께 검증된다. 그림 10의 16행의 첫 문자인 ‘^’은 이 행이 inverse triple constraint임을 표시하며 16행의 의미는 “my:S5를 준수하는 주어는 my:S2를 준수하는 주어에 의해 술어 ex:practises의 목적어로 반드시 1번 참조되어야 한다.”이므로 그림 9의 5행이 이 조건을 만족함을 확인할 수 있다. ShExC의 triple constraint 구문의 가장 마지막 요소는 cardinality이고 ShEx에서 디폴트 cardinality는 ‘1’이다. 그림 10의 8행을 제외한 모든 triple constraint의 cardinality가 ‘1’로 계산되었기에 따로 표시되지 않았다. 그림 10의 8행에 있는 ‘?’는 cardinality ‘0 혹은 1’을 뜻한다. ex:student10은 그림 10의 8행 제약 조건에 해당하는 술어·목적어가 있으나 ex:student20은 없음을 확인할 수 있다. 그림 10의 8행과 16행은 그림 8의 14~19행 술어·목적어 생성 규칙에서 비롯된 역관계에 있는 triples constraint들로서 이 둘의 cardinality가 child 입장에선 ‘0 혹은 1’로 parent 입장에선 ‘1’로 입력 데이터의 양태를 반영하여 올바르게 계산되었음도 이 사례에서 확인된다.

IV. Conformance Test

본 장에서는 제안된 시스템이 다양한 매핑 사례에서 어느 정도의 정확한 스키마를 생성하는지를 평가하고자 한다. 이러한 목적의 테스트 케이스 셋은 아직 개발되지 않았기 때문에 본 연구에서는 RML 프로세서의 정확성을 평가하고자 개발된 테스트 케이스 셋 [26]을 대용하고자 한다. [26]의 각각의 테스트 케이스는 데이터 파일, RML 문서, RDF 문서로 구성된다. 데이터 파일과 RML 문서는 RML 프로세서의 입력이며 RDF 문서는 평가 대상 RML 프로세서가 출력해야 할 정답에 해당한다. 본 평가에서도 데이터 셋과 RML 문서를 제안된 시스템의 입력으로 사용

하나 RDF 문서는 제안된 시스템이 생성한 스키마로 검증할 RDF 그래프가 된다. 케이스별로 스키마와 RDF 문서를 검증기에 입력한 후 타겟 노드 자격이 될 수 있는 모든 노드 각각에 대해 검증을 수행한 결과 검증 오류를 출력하지 않는다면 그 케이스는 성공으로 판단한다.

[26]은 총 297개의 테스트 케이스가 60개 카테고리로 나뉘어 정의되어 있다. 카테고리별로 매핑 규칙 구성을 달리한다. 같은 카테고리로 묶인 테스트 케이스들은 입력 데이터 소스 종류만 달리한다. 테스트 케이스에서 사용하는 입력 데이터 소스는 CSV, JSON, XML, 그리고 RDBMS인 MySQL, PostgreSQL Microsoft의 SQLServer다. 카테고리마다 모든 입력 데이터 소스에 대한 테스트 케이스가 정의된 것은 아니다. 본 평가에서는 각각의 테스트 케이스가 정한 입력 데이터 소스를 그대로 사용했으며 모든 테스트 케이스를 평가에 사용하였다. 아래 표 2는 평가 결과를 요약한다. 결과는 4가지 유형으로 정리된다.

Table 2. Test Result Summary

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

첫 번째는 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 경우에 대한 제안된 시스템의 반응은 RML 모델 생성 단계에서 RML 문서의 구문 오류를 감지하고 이를 화면에 출력하고 처리를 종료했다.

두 번째는 스키마 생성에 성공했으나 검증할 그래프가 제공되지 않은 경우로서 8개의 카테고리에서 발생했다. 이 8개의 카테고리 또한 테스트 사양에서 의도적으로 존재하지 않는 데이터로의 접근을 유도하여 RML 프로세서가 그래프 생성에 실패하도록 만든 케이스들이다. 이 경우 제안된 시스템은 데이터 소스 메타데이터 수집 단계에서 존재하지 않는 데이터로의 접근을 인지한 후 메타데이터 사용 없이 RML 매핑 규칙만을 사용하여 스키마를 생성했다.

세 번째는 스키마 생성에 성공했으나 주어진 그래프를 성공적으로 검증하지 못한 경우로서 6개의 카테고리에서 발생했다. 이 6개의 카테고리에 속한 RML 문서는 모두 RDF 그래프가 N-Quads 신택스[27]로 생성되도록 설계되었다. N-Quads는 한 문장을 주어, 술어, 목적어, 그래프 IRI로 구성한다. 그래프 IRI는 트리플이 속한 그래프명을 의미한다. 이것은 트리플마다 그 트리플이 속한 그래프도 한 문장 안에 함께 표현하기 위해 고안된 것이다. 제안된 시스템이 N-Quads로 표현된 그래프를 검증할 수 없는 이유는 ShEx에는 트리플이 특정 그래프에 속해야 한다고 제한할 수 있는 제약 조건이 존재하지 않기 때문이다. 이것은 SHACL도 마찬가지다.

나머지 모든 카테고리의 테스트 케이스에서는 제안된 시스템이 생성한 스키마로 주어진 그래프의 모든 트리플을 성공적으로 검증하였다. 요약하면 제안된 시스템은 의도적으로 오류를 유발한 경우와 ShEx 사양의 한계로 인하여 검증이 애초에 불가능한 경우를 제외하면 모든 테스트 케이스에서 정확한 스키마를 생성했다고 평가할 수 있다.

V. Conclusions

본 논문을 통해 RML 매핑으로 생성될 RDF 그래프의 스키마를 자동 생성해주는 시스템을 제안했다. 본 연구는 동일 목적의 유일한 기존 연구인 [25]가 생성하는 스키마가 그래프 검증 시 보이는 한계의 네 가지 원인을 다음과 같이 해소했다. 첫째, 기존 연구는 스키마 생성에 RML 문서만을 사용하지만 본 연구는 그것에 더하여 매핑의 입력 데이터도 스키마 생성에 사용한다. 이로 인해 검증을 위해 반드시 명시되어야 함에도 기존 연구에서는 구체적인 값을 명시할 수 없었던 제약 조건의 값을 본 연구의 스키마에서는 입력 데이터의 양태를 분석한 결과를 바탕으로 명시할 수 있게 되었다. 둘째, 본 연구는 기존 스키마가 취했던 "open" 방식이 아닌 "closed" 방식의 검증을 발동시키는 스키마를 생성한다. 이로 인해 타겟 노드의 구조가 셰이프가 묘사하는 구조와 다름이 없을 때만 검증을 통과하므로 보다 엄격한 검증이 가능해졌다. 셋째, 기존 연구는 SHACL로 스키마를 생성하지만 본 연구는 ShEx로 스키마를 생성한다. SHACL의 경우 특별한 구문 처리를 요구하나 기존 연구에서 대비하지 않았던 "repeated properties" 구조는 ShEx에서는 문제가 되지 않는 구조이기 때문에 ShEx를 선택한 자체로 해소되었다. 넷째, 본 연구는 기존 연구와 달리 서로 다른 triples map에서 생성된 트리플이 병합되는 상황을 고려한 스키마를 생성한다. 이로 인해 본 연구의 스키마는 검증에 있어서 보다 넓은 적용성을 확보하게 되었다. 실무적 측면에서 제안된 시스템은 확장성을 고려하여 데이터 분석을 Spark에 의뢰하

는 구조를 선택했다. 이로 인해 소위 빅 데이터 규모의 데이터로부터 생성하는 그래프라 하더라도 제안된 시스템은 이를 위한 스키마를 생성할 수 있도록 설계되었다. 제안된 시스템의 한계는 테스트에서 드러났듯이 N-Quads로 표현된 RDF 그래프의 경우 검증이 불가능하다는 점이다. 이는 ShEx 사양의 개정 사항을 추적하여 ShEx 사양이 N-Quads 신택스에 대한 처리가 가능해지면 제안된 시스템에도 반영할 예정이다. 향후 연구로는 각각의 셰이프를 SPARQL로 변환하여 타겟 노드를 자동으로 찾아주는 알고리즘을 개발하고자 한다.

ACKNOWLEDGEMENT

The source code of this work is hosted on https://github.com/jwchoi/RML2ShEx.

REFERENCES

[1] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. G., 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. 71, 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] Y. Leng, H. Wang and F. Lu, "Artificial Intelligence Knowledge Graph for Dynamic Networks: An Incremental Partition Algorithm," IEEE Access, Vol. 8, pp. 63434-63442, March 2020. DOI: 10.1109/ACCESS.2020.2982652

[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] R. Waterson, Build a knowledge graph in Amazon Neptune using Data Lens, https://aws.amazon.com/blogs/database/build-a-knowledge-graph-in-amazon-neptune-using-data-lens/

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

[9] 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/

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

[11] S. Harris, and A. Seaborne, SPARQL 1.1 Query Language, http://www.w3.org/TR/sparql11-query/

[12] B. Motik, P. F. Patel-Schneider, and B. Parsia, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), http://www.w3.org/TR/owl-syntax/

[13] K. Rabbani, M. Lissandrini, and K. Hose, "SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption," Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), pp. 260-263, Lyon, France, April 2022. DOI: 10.1145/3487553.3524253

[14] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P. Champin, and N. Lindström, JSON-LD 1.1, http://www.w3.org/TR/json-ld11/

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

[16] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," Springer Nature, pp. 233-266, 2022.

[17] D. Fernandez-Álvarez, J. E. Labra-Gayo, and D. Gayo-Avello, "Automatic extraction of shapes using sheXer," Knowledge-Based Systems, Vol. 238, No. 107975, pp. 1-9, February 2022. DOI: 10.1016/j.knosys.2021.107975

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

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

[27] 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.