Journal of The Korea Society of Computer and Information
Vol. 23 No. 10, pp. 33-43, October 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.10.033

ShEx Schema Generator for RDF Graphs Created by Direct Mapping

Ji-Woong Choi*

Abstract

In this paper, we propose a method to automatically generate the description of an RDF graph structure. The description is expressed in Shape Expression Language (ShEx), which is developed by W3C and provides the syntax for describing the structure of RDF data. The RDF graphs to which this method can be applied are limited to those generated by the direct mapping, which is an algorithm for transforming relational data into RDF by W3C. A relational database consists of its schema including integrity constraints and its instance data. While the instance data can have been published in RDF by some standard methods such as the direct mapping, the translation of the schema has been missing so far. Unlike the users on relational databases, the ones on RDF datasets were forced to write repeated vague SPARQL queries over the datasets to acquire the exact results. This is because the schema for RDF data has not been provided to the users. The ShEx documents generated by our method can be referred as the schema on writing SPARQL queries. They also can validate data on RDF graph update operations with ShEx validators. In other words, they can work as the integrity constraints in relational databases.

▸Keyword: ShEx, SHACL, RDF, SPARQL, Direct Mapping, R2RML

I. Introduction

데이터의 웹 (the Web of Data)을 확산시키기 위해 2006년 W3C의 제안으로 진행되고 있는 LOD(Linking Open Data) 프로젝트는 2017년 2월 현재 1,146개의 데이터 셋을 LOD 클라우드에 연결하고 있다[1]. 이 데이터 셋들은 기본적으로 백과사전 데이터(DBpedia)[2]와 컴퓨터 과학 논문 데이터(DBLP)[3]는 물론 우리 정부를 포함한 각국 정부의 공공 데이터부터 지리, 소셜 네트워킹, 그리고 최근 들어 생명과학 데이터에 이르기까지 다양한 분야의 정제된 데이터를 RDF 포맷으로 제공하고 있다. 공개 데이터는 RDF 크롤링, RDF 덤프, SPARQL 엔드포인트를 통해서 취득 가능하다[1]. RDF 크롤링과 덤프는 데이터 셋 사본 전체를 일괄적으로 제공하는 방식임에 비해 SPARQL 엔드포인트 방식은 사용자가 SPARQL 질의를 통해 요구한 데이터만을 선별적으로 추출할 수 있도록 한다.

SPARQL 질의문을 효율적으로 작성하기 위해서는 사용자에게 데이터 셋 즉, RDF 그래프에 대한 구조 정보가 추가로 제공되어야 한다. 그러나 지금까지 이러한 요구가 충족되지 않고 있다. 이것은 마치 관계형 데이터베이스에 접근하여 스키마 정보 없이 SQL 질의문을 작성하는 상황에 비유할 수 있다. 이러한 문제의 원인은 RDF 모델 자체의 성격에 기인한다. RDF는 트리플을 단위로 하는 그래프 모델을 따른다. 트리플은 주어, 술어, 목적어로 구성되며 술어는 그래프의 간선에 대응되며 나머지 두 요소는 그래프의 정점으로 해석된다. 서로 다른 트리플에서 동일한 주어 혹은 목적어가 존재할 경우 이들을 매개로 두 트리플은 연결된 것으로 간주한다. 이러한 최소한의 제약으로 그래프를 유연하게 구성해 나가는 방식은 사용자로 하여금 구조적 혹은 반구조적 데이터 모델 모두 RDF 그래프로 표현할 수 있도록 하지만 이점이 바로 RDF 그래프에 대한 스키마 생성을 어렵게 만드는 요인이 된다[4]. 본 연구에서는 SPARQL 엔드포인트에서 사용자들에게 SPARQL 질의문 작성 시 참조할 수 있는 RDF 데이터 셋에 대한 스키마 정보 제공이라는 목표를 달성하기 위하여 최근 들어 연구되고 있는 RDF 그래프 형상에 대한 검증(validation)을 위한 언어인 SHACL(Shapes Constraint Language)[5]과 ShEx(Shape Expressions Language)[6]에 주

• 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: 2018. 08. 20, Revised: 2018. 10. 06, Accepted: 2018. 10. 15.

목하였으며 이들 중 ShEx를 RDF 데이터 셋에 대한 스키마 표현 언어로서 사용하고자 한다.

SHACL과 ShEx는 RDF 그래프가 준수해야 하는 RDF 형상에 대한 제약 조건을 기술할 수 있는 구문을 제공한다. 이들 언어로 표현된 그래프 형상에 대한 제약 조건은 다른 관점에서 바라보면 RDF 그래프의 구조에 대한 설명이다. 이들 언어에 기반한 검증은 SHACL 혹은 ShEx로 작성한 제약 조건들과 검증 대상인 RDF 그래프를 검증기(validator)에 입력하여 검증기가 RDF 그래프의 형상에 대한 제약 조건 만족여부를 판단한다. 검증은 RDF 그래프 전체 혹은 일부에 대해서 수행할 수 있는데 이것은 구조적 혹은 반구조적 데이터 모델을 표현한 RDF 그래프 모두에서 사용할 수 있도록 한 고려이다.

본 연구는 관계형 데이터베이스로부터 Direct Mapping 알고리즘[7]에 의해 생성된 RDF 데이터 셋에 대한 ShEx 스키마를 자동 생성하는 방법을 제안한다. Direct Mapping 알고리즘은 관계형 데이터베이스 데이터를 RDF 형식으로 표현할 수 있도록 하며 W3C에 의해서 표준화된 방법이다. 이 방법에 의해 생성된 RDF 데이터 셋은 관계형 모델의 의미를 그래프 모델로 표현한 것에 다름 아니기 때문에 그 데이터 셋에 대한 스키마 또한 관계형 모델의 의미를 기반으로 자동 생성이 가능하다.

본 연구에서 SHACL 대신 ShEx를 채택한 이유는 ShEx 구문이 SHACL과 비교하여 human-readability 측면에서는 확실히 보다 직관적이며 간결하기 때문이다. ShEx 자체가 SHACL의 대안으로서 설계되었으며 주요한 설계 철학 중 하나가 human-readability가 우수한 구문의 개발이었다[8].

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구와 배경 지식을 기술한다. 3장에서는 Direct Mapping 알고리즘에 의해 생성된 RDF 그래프에 대한 스키마를 자동적으로 생성하여 ShEx 문서로 출력해내는 과정과 방법을 상세히 기술한다. 4장에서는 3장의 내용을 구현한 시스템의 데모를 수행하며, 5장에서는 결론을 맺는다.

II. Related Works

1. Schema for an RDF Dataset

이 절에서는 RDF 데이터 셋에 대한 스키마를 생성하기 위한 기존 연구들을 분석하고 RDF 데이터 셋에 대한 스키마 표현 언어로 사용될 수 있는 SHACL과 ShEx를 비교한다. RDF 데이터 셋에 대한 스키마를 생성하기 위한 연구들은 스키마를 생성하고자 하는 목적에 따라 두 부류로 분류할 수 있다. 최근 연구 [9]로 대표할 수 있는 첫 번째 부류는 데이터 셋의 품질을 향상시키기 위한 목적으로 스키마를 생성한다. 생성된 스키마는 데이터 셋을 공개하기 사전 단계에서 수행되는 데이터 셋 구조 검증 수단으로 사용된다. 최근 연구 [10]으로 대표할 수 있는 두 번째 부류는 공개된 데이터 셋에 접근한 사용자들의 효율적인 SPARQL 질의문 작성 작업을 도울 목적으로 스키마를 생성한다. 생성된 스키마는 사용자들이 데이터 셋의 구조를 직관적으로 파악하도록 하는 수단으로 제공된다. 요약하면, RDF 데이터 모델에 대한 스키마 생성 연구들은 구조 검증과 구조 설명이라는 스키마의 두 가지 역할이 통합되지 못하고 나뉘어져 진행되어 왔다. 이러한 이유로 구조 검증 수단으로 생성한 스키마는 구조 설명 수단으로 적합하지 않으며 구조 설명 수단으로 생성한 스키마는 구조 검증에 적합하지 않은 한계를 가지고 있다. 구조 검증 목적의 스키마를 사용할 대상은 기계이며 구조 설명 목적의 스키마를 사용할 대상은 인간이기 때문이다. W3C는 이러한 문제를 극복하기 위한 수단으로서 SHACL과 ShEx라는 RDF 데이터 모델에 대한 스키마 표현을 위한 표준화된 언어들을 제안하였다.

Table 1. Comparing SHACL and ShEx 2.0

	SHACL	ShEx 2.0
common features	Shapes, Node Constraints, Property Constraints, RDF Syntax, Logical Operators, Extension Mechanism	
human-readable syntax	undefined	support
recursion	not support	support
SPARQL query containment	support	not support

표 1은 SHACL과 ShEx를 비교한다. 두 언어는 모두 RDF 데이터 모델 요소 즉, 노드, 아크, 서브 그래프에 대한 구조 검증 용도의 제약조건을 명세할 수 있는 사양을 공통적으로 갖추고 있다. 또한 두 언어 모두 RDF 신택스로 직렬화가 가능하다. 이것은 두 언어 모두 기계가 처리하기 용이한 문서 형식을 지원함을 의미한다. 다만, ShEx 사양은 RDF 신택스 외에도 인간이 RDF 데이터 셋의 구조에 대한 제약조건들을 구조에 대한 설명으로 빠르게 인식하도록 할 목적의 ShExC (ShEx Compact syntax) 신택스를 추가로 정의하고 있다. 다음으로 SHACL과 ShEx의 대표적인 사양상의 한계를 살펴보면 SHACL은 recursive shape 즉, cyclic subgraph 구조에 대한 검증 절차가 사양에 정의되어 있지 않다[11]. SHACL 사양은 이 문제를 SHACL 검증기 구현 측의 문제로 남겼다. 즉, 구현에 따라 사이클의 검출 여부와 검출의 범위 그리고 처리 성능은 달라질 수 있음을 의미한다. ShEx는 SPARQL 질의문을 제약조건으로 사용할 수 없다[12]. SPARQL 질의문은 그래프의 패턴 매칭을 통해 데이터를 추출한다. 즉, SPARQL에는 그래프의 구조에 대한 패턴을 정의하는 다양한 구문이 존재한다. SHACL의 경우 언어 설계 단계에서 SPARQL의 이러한 구문을 제약조건 명세로 사용하기 위한 고려가 존재했으나 ShEx는 그러하지 못했으며 다만, 표 1에 기술된 확장 메커니즘이라는 ShEx 하위 사양을 통해 구현 측의 문제로 남긴 상태이다.

2. Direct Mapping Algorithm

이 절에서는 Direct Mapping에 의해 생성되는 RDF 데이터 셋의 형상을 기술하고자 한다. 우선 RDF 데이터 모델을 간략히 기술한다. RDF 데이터 모델은 주어, 술어, 목적어 형식의 트리플

(tripple)이라고 명명된 문장들의 집합이다. 트리플의 주어로 사용될 수 있는 RDF 컴포넌트는 IRI 혹은 블랭크 노드(blank node)이다. IRI는 글로벌 식별자이며 블랭크 노드는 로컬 식별자이다. 로컬은 하나의 RDF 그래프를 의미한다. 트리플의 술어로 사용될 수 있는 RDF 컴포넌트는 IRI이다. 술어 자격의 IRI는 프로퍼티(property)라고 한다. 트리플의 목적어로 사용될 수 있는 RDF 컴포넌트는 IRI 혹은 블랭크 노드 혹은 리터럴(literal)이다. 목적어 IRI와 블랭크 노드가 다른 트리플에서 주어로 재사용되면 이들로 인해 서로 다른 두 트리플이 연결된다.

관계형 데이터베이스는 스키마와 인스턴스로 구성된다. 표 2와 3은 Direct Mapping 알고리즘이 데이터베이스 스키마와 인스턴스 구성요소들을 각각 RDF 데이터 모델의 어떤 구성요소로 매핑 하는지 그리고 매핑에 의해 생성된 RDF 구성요소들이 트리플에서 어떤 위치에 오는지를 요약한다. S는 주어, P는 술어, O는 목적어를 의미한다.

Table 2. Mapping of RDB Schema

Component in RDB Schema	Component in RDF	Part in Triple
table	IRI	O
column	IRI	P
foreign key	IRI	P

Table 3. Mapping of RDB Instance

Component in RDB Instance	Component in RDF	Part in Triple
row	IRI, blank node	S, O
column value	literal	O

Direct Mapping 알고리즘은 테이블의 1개의 행(row)씩 입력단위로 하여 그로부터 복수개의 트리플들을 생성해 나간다. 동일한 행으로부터 생성된 트리플들의 주어는 해당 행을 의미하는 IRI 혹은 블랭크 노드이기 때문에 늘 동일하다. 행이 소속된 테이블에 기본 키(primary key)가 정의되어 있으면 그 행은 IRI로 매핑되며 그렇지 않다면 그 행은 블랭크 노드로 매핑된다. 그림 1은 Direct Mapping 알고리즘의 적용 사례로서 1개의 행을 갖는 People 테이블을 입력으로 하여 5개의 트리플들을 출력해낸 모습을 묘사하고 있다. 그림 1에서 ①~⑤번 트리플의 공통된 주어인 <People/ID=7>는 IRI로서 People 테이블의 PK 컬럼인 ID의 값이 7인 행으로부터 생성되었으며 5개 트리플들의 공통된 주어임을 확인할 수 있다. 목적어 자리의 <People> IRI는 테이블명 People로부터 생성되었다. People 테이블의 각각의 행으로부터 생성된 주어 역할의 IRI들은 모두 타입이 <People>이 된다. 그림 1에서 ②~④번 트리플들은 모두 한 행에 포함된 각각의 컬럼으로부터 생성되었다. ②~④번 트리플들의 술어는 각각의 컬럼명으로부터 생성된 IRI들이며, 목적어는 각각의 컬럼값으로부터 생성된 리터럴들이다. 그림 1의 People 테이블에는 Addresses 테이블의 ID 컬럼을 People 테이블의 addr 컬럼이 참조한다고 정의한 1개의 참조 무결성 제약조건이 정의되어 있다. 이러한 참조 무결성 제약조건으로부터 생성된 IRI인 <People#ref-addr>은 그림 1의 ⑤번 트리플의 술어로 사용되었다. ⑤번 트리플은 People 테이블의 기본 키 컬럼인 ID 컬럼의 값이 7로 식별되는 행은 Addresses 테이블의 기본 키 컬럼인 ID 컬럼의 값이 18로 식별되는 행을 참조함을 표현한다.

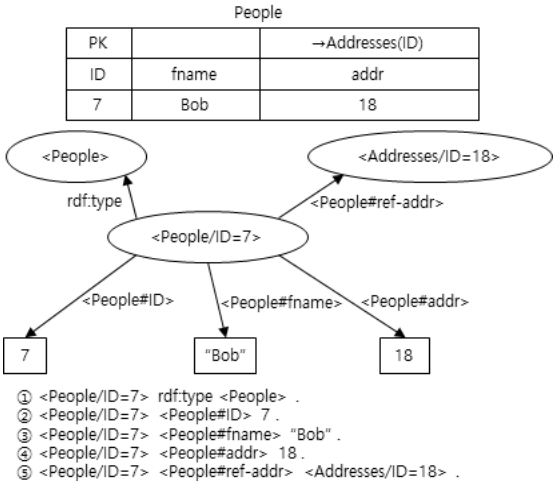

Fig. 1. Example of Direct Mapping Algorithm

3. ShEx (Shape Expressions)

이 절에서는 이 절 이후의 이해를 돕기 위하여 ShEx 모델을 간략히 소개한 후 ShEx의 본래의 용도인 ShEx를 사용한 RDF 그래프 구조 검증(validation) 절차에 대해 기술한다.

ShEx는 RDF 그래프의 구조를 기술하기 위한 언어이다. ShEx 스키마는 ShEx의 문법에 따라 작성된 모델을 지칭한다. ShEx 스키마에는 검증대상 RDF 그래프가 충족해야하는 구조에 대한 제약조건이 기술되어 있다. ShEx 스키마는 노드 제약조건(node constraint)과 셰이프 제약조건(shape constraint)의 컬렉션이다.

노드 제약조건은 검증대상 RDF 노드가 준수해야하는 제약조건을 기술해 놓은 것이다. ShEx에서는 검증대상 RDF 노드를 포커스 노드라고 명명했다. 여기에서, 노드라 함은 RDF 트리플의 주어 혹은 목적어를 지칭하며 포커스 노드가 될 수 있는 RDF 컴포넌트는 IRI, 블랭크 노드, 리터럴이다. 노드 제약조건은 다음 5가지 종류로 분류할 수 있다. Node kind constraint는 포커스 노드가 IRI, 블랭크 노드, 리터럴 중 어떤 종류이어야 하는지를 제약한다. Datatype constraint는 리터럴인 포커스 노드에 한해서 데이터타입을 제약한다. 리터럴의 데이터타입은 XML 스키마 데이터타입[13]을 의미한다. String facet constraint는 포커스 노드의 길이, 최소길이, 최대길이, 패턴을 제약한다. 이 제약조건은 모든 포커스 노드 종류에 적용할 수 있다. 즉, 모든 종류의 포커스 노드를 문자열로 해석하여 이 제약조건을 적용하는 것이다. 이 제약조건에서 문자열 패턴을 제약하기 위하여 [14]의 정규 표현식을 사용한다. Numeric facet constraint는 숫자형 리터럴인 포커스 노드에 한해서 숫자의 범위를 특정 값 이상, 이하, 초과, 미만으로 제약하며 숫자의 자릿수 또한 제약한다. Value constraint는 포커스 노드가 될 수 있는 후보 값들을 열거한 집합이다. 포커스 노드는 해당 집합의 원소이어야 한다. 이 제약조건은 모든 종류의 포커스 노드에 적용가능하다. 그림 2는 ShExC로 표현된 ShEx 스키마 의 한 예이다. 그림 2의

5번째 라인이 노드 제약조건이다. ex:age는 이 노드 제약조건의 식별자이다. 만약 이 노드 제약조건에 매핑되어 테스트되는 타겟 노드는 반드시 13이상 20이하의 정수이어야 한다.

```
01  PREFIX ex: <http://ex.example/#>
02  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
03  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
04
05  ex:age xsd:integer MinInclusive 13 MaxInclusive 20
06
07  ex:Enrollee {
08    foaf:age @ex:age ;
09    ex:hasGuardian IRI {1,2}
10  }
```

Fig. 2. ShEx Schema Example

```
01  inst:Alice foaf:age 15 ;
02    ex:hasGuardian inst:Person4 .
03
04  inst:Bob foaf:age 12 .
```

Fig. 3. RDF Graph Example

```
inst:Alice @ ex:Enrollee,
inst:Bob @ ex:Enrollee
{ FOCUS, foaf:age, _ } @ ex:Enrollee
```

Fig. 4. Shape Map Example

그림 2의 라인 7~10이 셰이프 제약조건이다. ex:Enrollee는 셰이프 제약조건의 식별자이다. 중괄호 파트는 트리플 표현식(tripple expression)이라고 한다. 트리플 표현식은 트리플 제약조건(tripple constraint)의 컬렉션으로 정의된다. 그림 2의 트리플 표현식은 2개의 트리플 제약조건으로 구성되어 있다. 셰이프 제약조건은 매핑된 타겟 노드의 형상을 제약한다. 예를 들어, ex:Enrollee에 매핑된 타겟 노드는 1개의 foaf:age라는 술어의 주어여야하며 그 술어의 목적어는 ex:age라는 노드 제약조건을 만족시켜야한다. 또한 그 타겟 노드는 1개 혹은 2개의 ex:hasGuardian이라는 술어의 주어여야하며 그 술어의 목적어는 반드시 IRI여야 함을 의미한다.

그림 3은 Turtle로 표현된 RDF 그래프 일부이다. inst:Alice 노드는 ex:Enrollee 셰이프 제약조건을 만족시키지만 inst:Bob 노드는 ex:hasGuardian 프로퍼티를 술어로 취하지 않을 뿐만 아니라 foaf:age의 목적어 12이기 때문에 ex:age 노드 제약조건을 위배하기 때문에 ex:Enrollee 셰이프 제약조건을 만족시키지 못한다.

RDF 그래프의 구조를 검증하기 위해서는 검증기에 ShEx 스키마 파일과 RDF 그래프 파일 그리고 셰이프 맵(shape map) 파일을 입력해야 한다. 셰이프 맵은 어떤 노드가 어떤 셰이프 제약조건의 타겟 노드가 되어야 하는지를 정의한다.

그림 4는 셰이프 맵의 예이다. 셰이프 맵은 그림 4의 상단 박스처럼 타겟 노드와 셰이프 제약조건의 쌍(pair)을 열거하는 방식으로 정의할 수 있다. 이러한 셰이프 맵을 고정 셰이프 맵(fixed shape map)이라고 한다. 또한 그림 4의 하단 박스처럼 중괄호에 트리플 패턴을 삽입하여 이 패턴을 만족시키는 RDF 노드를 타겟 노드로 정의하는 방식도 가능하다. 이러한 셰이프 맵을 질의 셰이프 맵(query shape map)이라고 한다. 이 질의 셰이프 맵의 의미는 목적어는 조건을 따지지 않고 단지 foaf:age를 술어로 하는 트리플들의 주어를 모두 타겟 노드로 선정하여 ex:Enrollee에 테스트하라는 것이다.

검증기는 상기한 3개의 파일을 입력으로 하여 검증과정을 거친 후 그 결과를 결과 셰이프 맵(result shape map) 파일에 출력해 준다. 결과 셰이프 맵의 출력 형식은 타겟 노드별로 셰이프 제약조건을 통과(pass)했는지 실패(fail)했는지 여부를 보고하며 실패 시 추가로 원인(reason)을 기술해 준다.

III. The Proposed Method

본 절에서는 Direct Mapping 알고리즘 방식으로 생성된 RDF 데이터 셋에 대한 ShEx 스키마를 생성하기 위한 절차들을 상세히 기술한다.

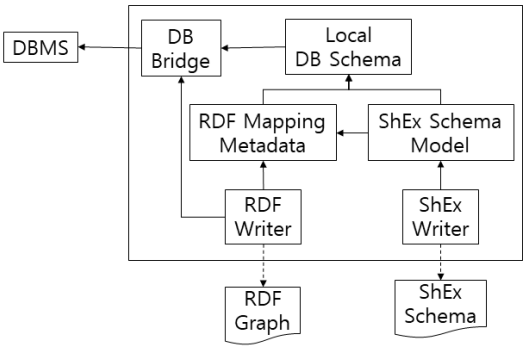

Fig. 5. Architecture for the Proposed Method

그림 5는 제안하는 방법을 위한 주요 모듈의 관계를 보여준다. 그림에서 실선 화살표는 모듈간의 종속성을 표현한다. 화살표 시작 측에 위치한 모듈은 화살촉 측에 위치한 모듈로부터 필요한 정보를 취득한다.

제안하는 방법은 3단계의 절차를 밟아 ShEx 스키마를 생성한다. 첫 단계에서는 데이터베이스 스키마 정보를 수집한다. 다음 단계에서는 ShEx 스키마 모델을 구축한다. 마지막 단계에서는 ShEx 스키마 모델을 파일 형식으로 출력한다.

1. Collecting Database Schema Information

ShEx 스키마 모델을 구축하기 위한 입력 데이터는 데이터베이스 스키마 정보이다. 따라서 본 연구에서는 ShEx 스키마 모델을 구축하기 전 단계로서 데이터베이스 스키마 정보를 일괄적으로 수집한다. 수집된 정보는 그림 5의 Local DB Schema 모듈에 저장된다. 이 모듈로 인해 이후 단계에서 DBMS에 접속할 필요가 없게 되며

DBMS의 종류와 무관하게 일관된 인터페이스로 타 모듈에서 DB 스키마 정보에 접근할 수 있게 된다.

직접적으로 DBMS로의 접근을 담당하는 모듈은 그림 5의 DB Bridge 모듈이다. DBMS 벤더별로 취득하고자 하는 정보에 따라 접근법이 상이한 부분이 존재하기 때문에 이 모듈은 그림 6과 같은 상속 구조를 취한다. DBMS 제품에 상관없이 동일한 접근법이 유효할 경우 그 기능은 DB Bridge 모듈에 위치하며 DBMS 제품별로 다른 방법을 사용해야 할 경우 그 기능은 해당 DBMS에 대응하는 하위 모듈에서 재정의(override)하였다. 이러한 구조는 DB Bridge의 하위 모듈들이 자신의 존재를 은폐하면서도 그 기능을 수행하도록 할 수 있다. 즉, 이 시스템에서 DBMS에 접근하고자 하는 모듈들은 모두 DBMS의 종류와 상관없이 DB Bridge라는 단 하나의 모듈을 통해 필요한 정보를 취득할 수 있다.

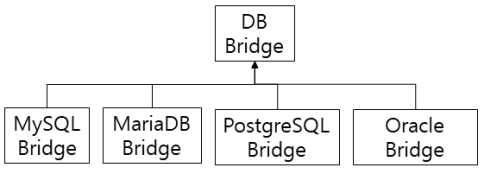

Fig. 6. Class Hierarchy of DB Bridge

Local DB Schema는 DB Bridge를 이용하여 데이터베이스 스키마 정보를 수집한다. 수집된 정보는 Local DB Schema 내에 그림 7의 구조로 관리된다.

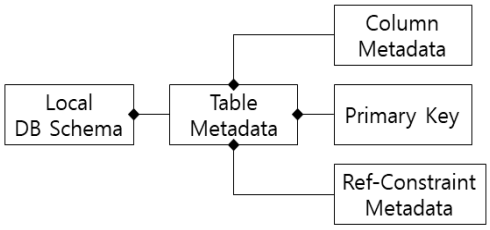

Fig. 7. Structure of Local DB Schema

Local DB Schema는 Table Metadata의 집합이다. Table Metadata는 테이블명을 포함한 1개의 테이블에 대한 정보를 담고 있다. Table Metadata는 대응하는 테이블의 컬럼수 만큼의 Column Metadata 리스트를 갖는다. Column Metadata는 1개의 컬럼에 대한 컬럼명, 데이터타입, 그리고 널(null) 허용 여부 등 해당 컬럼에 대해 정의한 무결성 제약조건 정보를 담고 있다. Table Metadata는 대응하는 테이블에 대해 정의한 기본 키와 외래 키 제약조건 정보도 포함한다. 외래 키 제약조건에 해당하는 Ref-Constraint Metadata가 담고 있는 정보는 참조하는 테이블과 참조 당하는 테이블, 제약조건에 포함된 컬럼들과 그들의 순서 그리고 그 각각의 컬럼이 참조하는 컬럼명이다. 기본 키 제약조건에 해당하는 Primary Key 속성이 담고 있는 정보는 기본키로 정의된 컬럼명과 그들의 순서이다. 기본키/외래 키 제약조건에 대해서 컬럼들의 순서 정보를 관리하는 이유는 Direct Mapping 알고리즘에서 기본 키를 갖는 테이블의 행으로부터 생성하는 노드에 대한 IRI와 외래 키 제약조건으로부터 생성하는 프로퍼티에 대한 IRI 작명 시 이들 컬럼명이 포함되는데 이 때 컬럼의 순서도 반영되기 때문이다.

Local DB Schema는 그림 7의 구조로 관리되는 내부 정보를 외부로 제공할 수 있는 인터페이스를 갖추고 있다. 따라서 데이터베이스 스키마 정보를 얻고자 하는 외부 모듈들은 Local DB Schema의 인터페이스를 통해서 필요한 정보를 획득하도록 설계되었다.

2. Constructing ShEx Schema Model

그림 8은 ShEx Schema Model의 내부 구조이다. ShEx 스키마는 노드 제약조건과 셰이프 제약조건의 컬렉션으로 정의되므로 ShEx Schema Model 또한 Node Constraint Model의 집합과 Shape Model의 집합으로 구성된다. 트리플 표현식은 셰이프 제약조건을 구성하는 한 파트이며 트리플 제약조건의 배열(arrangement)은 트리플 표현식으로 간주된다[6]. 이러한 정의를 반영하여 그림 8의 Shape Model의 구성요소로서의 Triple Constraint Model의 집합은 셰이프 제약조건의 트리플 표현식에 대응한다.

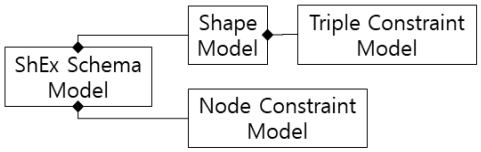

Fig. 8. Structure of ShEx Schema Model

그림 9는 데이터베이스 스키마 정보를 기반으로 ShEx 스키마 모델을 생성하는 과정을 보여준다. 이 알고리즘은 우선 라인 1에서 빈 ShEx 스키마 모델을 생성한다. 다음으로 3~33 라인의 반복문에서는 1회 반복 당 1개의 데이터베이스 테이블 정보로부터 노드 제약조건 모델들과 셰이프 모델을 생성하여 ShEx 스키마 모델에 추가해 나간다. 마지막으로 라인 34에서 완성된 ShEx 스키마 모델을 반환한다.

Table 4. Mapping between ShEx and RDB Schema

ShEx Schema		RDB Schema
Node Constraint		column
Shape		table
Triple Constraint	Type 1	table
	Type 2	column
	Type 3	(Referencing) Referential Constraint
	Type 4	(Referenced) Referential Constraint

표 4는 그림 9 알고리즘에서 ShEx 스키마의 각 요소를 생성하기 위해 제공되는 데이터베이스 스키마 요소들을 요약한다. 그림 9의 5~9 라인은 테이블의 개별 컬럼을 입력으로 하여 노드 제약조건 모델을 생성한 후 ShEx 스키마 모델에 추가한다. 라인 10에서 테이블을 입력으로 한 셰이프 모델을 생성한다. 이 셰이프 모델은 이후 코드의 실행에 의해 생성된 트리플 제약조건 모델들을 추가한 후 라인 32에서 스키마 모델에 추가된다.

```
function makeShExSchemaModel(db)
input: a Local DB Schema db
output: the ShEx Schema Model schema built from db
01  schema ← create an empty ShEx Schema Model.
02  tables ← get the set of tables from db.
03  for each table ∈ tables {
04    cols ← get the list of columns in table from db.
05    for each col ∈ cols {
06      nc ← create a Node Constraint Model from
07            table and col.
08      add nc to schema.
09    }
10    shape ← create a Shape Model from table.
11    tc ← create a Triple Constraint Model from table.
12    add tc to shape.
13    for each col ∈ cols {
14      tc ← create a Triple Constraint Model from
15            table and col.
16      add tc to shape.
17    }
18    refs ← get the set of referential constraints in
19            table from db.
20    for each ref ∈ refs {
21      tc ← create a Triple Constraint Model from
22            table and ref.
23      add tc to shape.
24    }
25    refs ← get the set of referential constraints
26            referencing table from db.
27    for each ref ∈ refs {
28      tc ← create a Triple Constraint Model from
29            table and ref.
30      add tc to shape.
31    }
32    add shape to schema.
33  }
34  return schema
```

Fig. 9 Algorithm for Constructing ShEx Schema Model

트리플 제약조건 모델은 표 4에서처럼 4가지 종류의 입력을 바탕으로 생성된 후 셰이프 모델에 추가된다. Type 1~4는 본 연구의 기술(記述) 편의를 위해 부여한 것이다. 테이블을 입력으로 하는 Type 1 트리플 제약조건 모델의 생성 및 추가는 라인 11~12에 해당한다. 테이블의 컬럼 당 생성되는 Type 2 트리플 제약조건을 위한 코드는 라인 13~17에 해당한다. Type 3과 Type4는 모두 외래키 제약조건을 기반으로 생성되지만, Type 3은 해당 반복의 테이블이 다른 테이블을 참조하는 외래키 제약조건을 기반으로 하는 반면 Type 4는 다른 테이블이 해당 반복의 테이블을 참조하는 외래 키 제약조건을 기반으로 한다는 차이가 있다. Type 3를 위한 코드는 라인 20~24이며 Type 4를 위한 코드는 라인 27~31이다.

2.1 Constructing Node Constraint Models

본 절에서는 데이터베이스 컬럼을 기반으로 노드 제약조건 모델이 생성되는 방법을 기술한다. 표 5는 노드 제약조건 모델을 구성하는 속성의 목록과 각 속성이 컬럼의 어떤 속성에 기반하여 생성되는지를 요약한다.

Table 5. Properties of Node Constraint Model

Node Constraint	Column
id	name
datatype	SQL datatype
xsFacet	SQL datatype
values	ENUM constraint

표 5의 노드 제약조건 모델의 4가지 속성은 ShEx 신택스에서 정의한 노드 제약조건의 구성요소이다. 이 단계에서의 노드 제약조건 모델은 각 속성값을 결정하기만 하며 이후 ShEx 스키마 출력단계에서 속성값들을 ShEx 신택스에 맞게 조립하여 문자열로 출력해준다. id 속성값은 컬럼명을 사용한다. id 속성값은 ShEx 스키마에서 유일해야하는데 컬럼명이 데이터베이스에서 유일하므로 이를 기반으로 결정한 id 속성값 또한 유일하다. datatype 속성값은 컬럼의 SQL 데이터타입에 매핑된 RDF 데이터타입으로 결정된다. 결정의 기준은 Direct Mapping에서 사용하는 데이터타입간 매핑의 기준 [15]를 따랐다. datatype 속성값의 의미는 Direct Mapping에서 컬럼값을 RDF의 리터럴로 변환했을 때 그 리터럴의 데이터타입이다.

Table 6. xsFacets Used in Node Constraint Model

RDF datatype	xsFacet
xsd:boolean	REGEXP
xsd:date	REGEXP
xsd:dateTime	REGEXP
xsd:decimal	TOTALDIGITS & FRACTIONDIGITS
xsd:double	MININCLUSIVE
xsd:hexBinary	MAXLENGTH
xsd:integer	MININCLUSIVE & MAXINCLUSIVE
xsd:string	MAXLENGTH
xsd:time	REGEXP

xsFacet 속성값은 datatype 속성값에 종속적이다. 어떤 노드 제약조건에 datatype 속성과 더불어 xsFacet 속성이 존재한다면 이 노드 제약조건을 만족하는 노드는 기본적으로 datatype에 속하는 리터럴이며 추가로 xsFacet 조건도 만족하는 리터럴로 제한된다. 즉, xsFacet 속성은 datatype 속성값에 따른 추가적인 제약이다. 표 6은 노드 제약조건의 datatype 속성에 따른 xsFacet 속성들이다. 표 6에서 REGEXP는 정규표현식을 의미한다. xsd:boolean 타입을 위한 정규표현식은 “/true|false/”이다. 이것은 true 혹은 false 둘 중에 하나만 가능하다는 의미이다. [15]에서 xsd:boolean 타입의 값으로서 이 두 가지만 허용하기 때문이다. 날짜/시간과 관련된 xsd:date, xsd:datetime, xsd:time의 경우에도 정규표현식 xsFacet 속성값을 갖도록 하였다. 그 이유는 날짜/시간에 대한 SQL 데이터타입들이 저장 가능한 값의 범위에 있어서 DBMS 별로 차이가 있다. 따라서 본 연구에서는 날짜/시간 관련 타입들에 대한 xsFacet 속성에 원천 DBMS가 저장할 수 있는 값의 범위를 정규표현식으로 표현한 문자열을 할당하였다. 본 연구에서 이러한 제약까지 ShEx 스키마에 포함시킨 이유는 SPARQL을 사용해 RDF 그래프에 데이터를 추가하는 상황까지 대비하기 위함이다. SPARQL 1.1 [16]는 RDF 그래프에 데이터를 추가할 수 있는 사양을 추가하였다. Direct Mapping에 의해 생성된 RDF 그래프는 원천 데이터베이스

데이터에 대한 RDF 표현이기 때문에 그 RDF 그래프에 추가된 데이터는 최종적으로 DBMS에 저장될 수 있으며 이 때 추가된 값이 DBMS가 저장할 수 있는 범위 밖에 있다면 문제가 발생할 수 있기 때문이다. xsd:decimal 타입을 위한 xsFacet 속성은 리터럴의 총 자리수를 의미하는 TOTALDIGITS와 소수점 이하 자리수를 의미하는 FRACTIONDIGITS으로 구성된다. xsd:double 타입에 대한 xsFacet 속성은 옵션으로서 음이 아닌 실수만 허용하는 컬럼일 경우 “MININCLUSIVE 0”이 속성값이 된다. xsd:hexBinary는 이진 데이터를 문자로 표현한다. 이 타입을 위한 xsFacet 속성은 컬럼에 정의된 최대 바이트 한도를 2배하여 MAXLENGTH 값으로 사용한다. xsd:String을 위한 xsFacet 속성은 컬럼에 정의된 최대 문자 한도를 MAXLENGTH 값으로 사용한다. xsd:integer를 위한 xsFacet 속성은 MININCLUSIVE와 MAXINCLUSIVE가 사용된다. DBMS들은 다양한 바이트 크기의 정수형들을 제공한다. 이들 SQL 정수형들이 모두 xsd:integer로 매핑된다. 또한 SQL 정수형 컬럼들은 unsigned로 한정되어 사용될 수 있다. 따라서 본 연구에서는 입력 컬럼의 정확한 정수 데이터 범위를 추출한 후 xsFacet 속성값에 할당하였다. 마지막으로 표 5의 values 속성은 ShEx에서의 열거형 제약조건 표현도구로서 ENUM 제약조건이 가해진 컬럼의 경우 이 제약조건을 values 속성으로 대응시킨다.

2.2 Constructing Shape Models

셰이프 모델은 테이블을 입력으로 하여 생성되며 id, nodeKind, xsFacet 속성과 트리플 제약조건 모델의 집합 속성으로 구성된다. id 속성값은 입력 테이블 이름을 기반으로 생성되며 ShEx 스키마 내에서 유일하게 결정된다. nodeKind 속성값은 입력 테이블에 기본 키 제약조건이 존재할 경우 “IRI”, 그렇지 않을 경우 블랭크 노드를 의미하는 “BNODE”가 된다. 그 이유는 Direct Mapping 알고리즘은 기본 키 제약조건이 있는 테이블의 행으로부터는 IRI를 생성하고 그렇지 않은 테이블의 행으로부터는 블랭크 노드를 생성하기 때문이다. 또한 셰이프 제약 조건에 테스트 될 RDF 노드는 테이블의 행으로부터 생성된 노드이기 때문이다. 셰이프 모델의 xsFacet 속성은 nodeKind 속성값이 “IRI”일 경우에 한하여 IRI의 패턴을 정규표현식으로 표현한 문자열을 그 값으로 한다. 그 이유는 Direct Mapping 알고리즘에 의해 동일한 테이블의 행들로부터 생성된 IRI들이 일정한 문자열 패턴을 갖기 때문이다.

2.3 Constructing Triple Constraint Models

트리플 제약조건 모델은 생성 후 셰이프 모델에 추가된다. 이것은 셰이프 제약조건을 만족하는 노드에 대해 형상에 관한 제약조건 또한 추가시키는 효과가 있다. 트리플 제약조건 모델의 속성은 predicate, valueExpr, cardinality, inverse이다. predicate 속성은 프로퍼티 IRI가 값이 된다. 이것은 셰이프 제약조건을 만족하는 노드에 연결되어 있어야 하는 프로퍼티를 지정하는 것이다. valueExpr 속성은 프로퍼티 IRI로 연결된 반대쪽 노드가 만족시켜야 하는 노드 제약조건이다. inverse 속성은 true 혹은 false의 값을 갖는다. inverse 속성이 false 일 때의 셰이프 제약조건을 만족하는 노드는 프로퍼티 IRI에 대해 주어, valueExpr 제약조건을 만족하는 노드는 목적어이며 true 일 때는 반대로 셰이프 제약조건을 만족하는 노드가 목적어, valueExpr 제약조건을 만족하는 노드는 주어이다. 본 연구에서는 Type 4일 경우만 true이다. cardinality 속성은 동일 구조의 트리플이 존재할 수 있는 개수를 명시적으로 제한하는 것이다.

Type 1 트리플 제약조건 모델의 사례는 그림 1의 ①번 트리플이다. Type 1의 경우 predicate는 항상 “rdf:type” 프로퍼티, cardinality는 1이다. valueExpr의 값은 입력 테이블로부터 유도된 IRI이다. 이 IRI는 그림 5의 RDF Mapping Metadata 모듈에 질의할 경우 얻어낼 수 있다. RDF Mapping Metadata와 RDF Writer 모듈은 Direct Mapping 알고리즘의 구현체이다. 특히 RDF Mapping Metadata는 RDB 스키마 요소와 RDF 요소 간 매핑정보를 유지하고 있으며 특정 RDB 스키마 요소에 매핑된 RDF 요소를 찾아주는 인터페이스 또한 갖추고 있다.

Type 2 트리플 제약조건 모델의 사례는 그림 1의 ②~④번 트리플이다. Type 2의 경우 predicate는 입력 컬럼으로부터 유도된 IRI이다. 이 IRI는 RDF Mapping Metadata 모듈에 질의하여 얻어낸다. valueExpr은 입력 컬럼으로부터 생성된 노드 제약조건 모델의 id값이다. ShEx는 valueExpr의 “@id값”을 노드 제약조건 참조로 해석한다. cardinality는 입력 컬럼이 null을 허용한다면 0 또는 1, 그렇지 않다면 1이다.

Type 3 트리플 제약조건 모델의 사례는 그림 1의 ⑤번 트리플이다. Type 3의 경우 predicate는 입력 외래 키 제약조건으로부터 유도된 IRI이다. 이 IRI 또한 RDF Mapping Metadata 모듈에 질의하여 구한다. valueExpr에는 입력 외래 키 제약조건에서 참조 당하는 테이블명을 우선 추출한 후 그 테이블을 입력으로 하여 생성된 셰이프 모델의 id값을 조회하여 할당한다. cardinality에는 입력 외래 키 제약조건에서 참조하는 컬럼들을 찾아낸 후 이 컬럼들 중 한 개라도 null을 허용하는 컬럼이 존재할 경우 0 또는 1, 그렇지 않으면 1을 할당한다.

Type 4 트리플 제약조건 모델은 Type 3의 역관계이다. 즉, 주어와 목적어가 뒤바뀐 관계이다. 따라서 Type 4 트리플 제약조건 모델은 Type 3의 valueExpr에 해당하는 셰이프 모델에 추가되며 Type 4 트리플 제약조건 모델의 valueExpr에는 Type 3 트리플 제약조건을 포함하는 셰이프 모델의 id가 할당된다. 당연히 predicate은 Type 3와 동일하다. 다만, cardinality는 별도의 계산 없이 0 혹은 그 이상이 된다.

3. Writing the ShEx Document

그림 10은 두 번째 단계에서 구축한 ShEx 스키마 모델을 파일로 출력하는 마지막 단계를 묘사한다.

```
function writeShExDocument(db, schema)
input: Local DB Schema db, ShEx Schema Model schema built from db
output: ShEx document shexDoc
01  write directive to shexDoc.
02  tables ← get the set of tables from db.
03  for each table ∈ tables {
04    cols ← get the list of columns in table from db.
05    for each col ∈ cols {
06      nc ← get the Node Constraint Model built from
07           table and col in shcema.
08      write nc to shexDoc.
09    }
10    shape ← get the Shape Model built from table in
11            schema.
12    write shape to shexDoc.
13  }
14  return shexDoc
```

Fig. 10. Algorithm to Write ShEx Document

그림 10 알고리즘은 라인 1에서 directive를 문서에 맨 처음 쓴다. directive는 네임스페이스들에 대한 prefix들을 선언하는 곳이다. directive에서 선언된 prefix들로 인해 IRI에 대한 표현을 간결하게 할 수 있다. directive는 다음과 같은 형식으로 출력된다. 아래 출력 형식에서 PN_PREFIX가 prefix이며 IRIREF가 네임스페이스이다.

("PREFIX" PN_PREFIX ":" IRIREF)*

그림 10의 3~13라인은 하나의 테이블로부터 생성된 ShEx 스키마 모델 요소들을 문서에 출력하는 코드이다. 우선 5~9라인에서 해당 테이블의 컬럼들로부터 생성된 각각의 노드 제약조건 모델을 문서에 출력한다. 다음으로 10~12라인에서 해당 테이블로부터 생성된 셰이프 모델을 출력한다. 이러한 출력을모든 테이블에 대하여 수행하면 ShEx 문서로의 출력이 종료된다. 이 과정에서 1개의 노드 제약조건 모델에 대한 출력 형식은 다음과 같다.

id valueSet | id datatype xsFacet*

위 출력 형식은 id를 우선 출력한 후 노드 제약조건 모델에 valueSet 속성값이 존재하면 valueSet 만을 출력하고 그렇지 않다면 datatype 속성과 부속 xsFacet 속성들을 출력함을 의미한다.

1개의 셰이프 모델에 대한 출력 형식은 다음과 같다.

id nodeKind REGEXP? "{" tripleConstraint+ "}"

위 출력 형식은 다음을 의미한다. 우선 id를 출력하고 nodeKind를 출력한다. nodeKind는 "IRI" 혹은 "BNODE"이다. REGEXP는 nodeKind가 "IRI"일 때만 출력된다. 마지막으로, 중괄호 안에 셰이프 모델 내에 있는 트리플 제약조건 모델을 모두 출력한다. 이때, 개별 트리플 제약조건 모델은 다음의 형식으로 출력된다.

"^"? predicate ("[" IRI "]" | "@"id) cardinality? ";"?

위 출력 형식은 다음을 의미한다. 문자 "^"가 출력되는 경우는 inverse 속성이 true일 때 만이다. 즉, Type 4 트리플 제약조건 모델일 때 만이다. 다음으로 predicate 속성의 값인 프로퍼티 IRI를 출력한다. 이어서 IRI 또는 "@id"가 출력될 수 있다. Type 1일 경우에 한하여 IRI가 출력되며 그 IRI는 현재 출력중인 트리플 제약조건이 포함된 셰이프를 생성케 한 테이블과 동일한 테이블을 기반으로 Direct Mapping 알고리즘에서 생성한 IRI이다. Type 2일 경우 "@" 문자 다음에 출현하는 id는 Type 2 트리플 제약조건을 생성케 한 컬럼과 동일한 컬럼으로부터 생성된 노드 제약조건의 id이다. Type 3와 4일 경우 "@" 다음의 id는 셰이프의 id이다. 다만, 그 id는 Type 3일 경우 현재 출력중인 트리플 제약조건을 생성케 한 외래 키 제약조건에 의해 참조 당하는 테이블로부터 생성된 셰이프의 id이며 Type 4일 경우 Type 3과 반대로 외래 키 제약조건이 정의된 테이블 즉, 외래 키를 갖게 되는 테이블로부터 생성된 셰이프의 id이다. cardinality 속성의 출력은 그 값이 1일 경우 디폴트로서 출력문자가 없으며 0 또는 1일 경우 "?", 0 이상일 경우 "*"문자를 출력한다. 마지막으로 세미콜론은 트리플 제약조건의 구분자로서 마지막에 출력되는 트리플 제약조건을 제외하고 항상 출력된다.

IV. Demonstration Details

본 장에서는 그림 11의 구조 하에 그림 12의 데이터로 구성된 데이터베이스로부터 본 연구에서 제안하는 방법으로 RDF와 ShEx 문서를 생성한 후 RDF 그래프의 구조를 ShEx 스키마로 검증하는 과정을 보이고자 한다.

```
addresses (ID int ,city char(10), state char(2))
people (ID int, fname char(10), addr int, deptName char(10),
        deptCity char(10), fk (addr) refs addresses(ID),
        fk (deptName, deptCity) refs department(name, city))
department (ID int, name char(10), city char(10),
            manager int,
            unique (name, city),
            fk (manager) refs people(ID))
projects (lead int, name varchar(50), deptName varchar(50),
          deptCity varchar(50),
          unique (lead, name),
          unique (name, deptName, deptCity),
          fk (lead) refs people(ID),
          fk (deptName, deptCity) refs department(name, city))
taskassignments (worker int, project varchar(50),
                 deptName varchar(50), deptCity varchar(50),
                 fk (worker) refs people(ID),
                 fk (project, deptName, deptCity) refs projects(name,
deptName, deptCity),
                 fk (deptName, deptCity) refs department(name, city))
tweets (tweeter int, when timestamp, text char(140),
        fk (tweeter) refs people(ID))
```

Fig. 11. Database Schema of direct_mapping

tweets

tweeter	when	text
7	2010-08-30T01:33	I really like lolcats.
7	2010-08-30T09:01	I take it back.

addresses

ID	city	state
18	Cambridge	MA

people

ID	fname	addr	deptName	deptCity
7	Bob	18	accounting	Cambridge
8	Sue	NULL	NULL	NULL

department

ID	name	city	manager
23	accounting	Cambridge	8

projects

lead	name	deptName	deptCity
8	pencil survey	accounting	Cambridge
8	eraser survey	accounting	Cambridge

taskassignments

worker	project	deptName	deptCity
7	pencil survey	accounting	Cambridge

Fig. 12. Database Instance of direct_mapping

그림 11은 direct_mapping이라고 명명한 데이터베이스 스키마이다. 이 스키마는 W3C에서 공개한 Direct Mapping 알고리즘 표준 문서인 [7]에서 소개된 것으로 Direct Mapping 알고리즘이 고려하는 구조적 특성을 모두 포함한다. 각각의 테이블에 대해서 컬럼의 이름과 타입, UNIQUE 그리고 외래 키 제약조건을 표기하였다. 밑줄이 있는 컬럼은 기본 키 컬럼이다. 본 장의 실험을 위하여 이 데이터베이스를 오픈소스 데이터베이스인 MariaDB[17]에 구축하였다.

그림 13은 direct_mapping 데이터베이스를 입력으로 하여 ShEx Schema Model과 ShEx Writer 모듈을 통해 생성된 ShEx 파일 내용이다.

```
BASE <http://foo.example/direct_mapping/>
PREFIX foo: <http://foo.example/direct_mapping/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

foo:addressesID xsd:integer MININCLUSIVE -2147483648
                            MAXINCLUSIVE 2147483647
foo:addressesCity xsd:string MAXLENGTH 10
foo:addressesState xsd:string MAXLENGTH 2

foo:addressesShape IRI
/^http:₩/₩/fooW.exampleW/direct_mappingW/addressesW/ID
=(.*)$/ {
  a [<addresses>] ;
  <addresses#ID> @foo:addressesID ;
  <addresses#city> @foo:addressesCity? ;
  <addresses#state> @foo:addressesState? ;
 ^<people#ref-addr> @foo:peopleShape*
}

foo:peopleID xsd:integer MININCLUSIVE -2147483648
                         MAXINCLUSIVE 2147483647
foo:peopleFname xsd:string MAXLENGTH 10
foo:peopleAddr xsd:integer MININCLUSIVE -2147483648
                           MAXINCLUSIVE 2147483647
foo:peopleDeptName xsd:string MAXLENGTH 10
foo:peopleDeptCity xsd:string MAXLENGTH 10

foo:peopleShape IRI
/^http:₩/₩/fooW.exampleW/direct_mappingW/peopleW/ID=(.*
)$/ {
 a [<people>] ;
 <people#ID> @foo:peopleID ;
 <people#fname> @foo:peopleFname? ;
 <people#addr> @foo:peopleAddr? ;
 <people#deptName> @foo:peopleDeptName? ;
 <people#deptCity> @foo:peopleDeptCity? ;
 <people#ref-addr> @foo:addressesShape? ;
 <people#ref-deptName;deptCity> @foo:departmentShape? ;
 ^<department#ref-manager> @foo:departmentShape* ;
 ^<projects#ref-lead> @foo:projectsShape* ;
 ^<taskassignments#ref-worker>
                          @foo:taskassignmentsShape* ;
 ^<taskassignments#ref-worker>
                          @foo:taskassignmentsShape* ;
 ^<tweets#ref-tweeter> @foo:tweetsShape*
}
```

```
foo:departmentID xsd:integer MININCLUSIVE -2147483648
                             MAXINCLUSIVE 2147483647
foo:departmentName xsd:string MAXLENGTH 10
foo:departmentCity xsd:string MAXLENGTH 10
foo:departmentManager xsd:integer
 MININCLUSIVE -2147483648 MAXINCLUSIVE 2147483647

foo:departmentShape IRI
/^http:₩/₩/fooW.exampleW/direct_mappingW/departmentW/I
D=(.*)$/ {
  a [<department>] ;
  <department#ID> @foo:departmentID ;
  <department#name> @foo:departmentName? ;
  <department#city> @foo:departmentCity? ;
  <department#manager> @foo:departmentManager? ;
  <department#ref-manager> @foo:peopleShape? ;
 ^<people#ref-deptName;deptCity> @foo:peopleShape* ;
 ^<projects#ref-deptName;deptCity> @foo:projectsShape* ;
 ^<taskassignments#ref-deptName;deptCity>
                                  @foo:taskassignmentsShape*
}

foo:projectsLead xsd:integer MININCLUSIVE -2147483648
                             MAXINCLUSIVE 2147483647
foo:projectsName xsd:string MAXLENGTH 50
foo:projectsDeptName xsd:string MAXLENGTH 50
foo:projectsDeptCity xsd:string MAXLENGTH 50

foo:projectsShape BNODE {
 a [<projects>] ;
 <projects#lead> @foo:projectsLead? ;
 <projects#name> @foo:projectsName? ;
 <projects#deptName> @foo:projectsDeptName? ;
 <projects#deptCity> @foo:projectsDeptCity? ;
 <projects#ref-lead> @foo:peopleShape? ;
 <projects#ref-deptName;deptCity>
                                  @foo:departmentShape? ;
 ^<taskassignments#ref-project;deptName;deptCity>
                                  @foo:taskassignmentsShape*
}

foo:taskassignmentsWorker xsd:integer
 MININCLUSIVE -2147483648 MAXINCLUSIVE 2147483647
foo:taskassignmentsProject xsd:string MAXLENGTH 50
foo:taskassignmentsDeptName xsd:string MAXLENGTH 50
foo:taskassignmentsDeptCity xsd:string MAXLENGTH 50

foo:taskassignmentsShape IRI
/^http:₩/₩/fooW.exampleW/direct_mappingW/taskassignment
sW/worker=(.*);project=(.*)$/ {
  a [<taskassignments>] ;
  <taskassignments#worker> @foo:taskassignmentsWorker ;
  <taskassignments#project> @foo:taskassignmentsProject ;
  <taskassignments#deptName>
                                  @foo:taskassignmentsDeptName? ;
  <taskassignments#deptCity>
                                  @foo:taskassignmentsDeptCity? ;
  <taskassignments#ref-worker> @foo:peopleShape? ;
  <taskassignments#ref-worker> @foo:peopleShape? ;
  <taskassignments#ref-project;deptName;deptCity>
                                  @foo:projectsShape? ;
  <taskassignments#ref-deptName;deptCity>
                                  @foo:departmentShape?
```

```
}

foo:tweetsTweeter xsd:integer MININCLUSIVE -2147483648
                              MAXINCLUSIVE 2147483647
foo:tweetsWhen xsd:dateTime
/(^(19[7-9][0-9]|20([0-2][0-9]|3[0-7]))-(0[1-9]|1[0-2])
-(0[1-9]|[12][0-9]|3[01])T(0[0-9]|1[0-9]|2[0-3]):([0-5]
[0-9]):([0-5][0-9])((.[0-9]{1,6}){0})$)|(^2038-01-(0[1-9]|
1[0-8])T(0[0-9]|1[0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9])
((.[0-9]{1,6}){0})$)|(^2038-01-19T0[0-4]:([0-5][0-9]):([0
-5][0-9])((.[0-9]{1,6}){0})$)|(^2038-01-19T05:(0[0-9]|1[
0-3]):([0-5][0-9])((.[0-9]{1,6}){0}))|(2038-01-19T05:14:0
[0-6]((.[0-9]{1,6}){0})$)|(^2038-01-19T05:14:07((.0{1,6}){
0})$)/
foo:tweetsText xsd:string MAXLENGTH 140

foo:tweetsShape BNODE {
  a [<tweets>] ;
  <tweets#tweeter> @foo:tweetsTweeter? ;
  <tweets#when> @foo:tweetsWhen ;
  <tweets#text> @foo:tweetsText? ;
  <tweets#ref-tweeter> @foo:peopleShape?
}
```

Fig. 13. ShEx Schema generated from direct_mapping

그림 13의 출력 형식을 설명하면 첫 세 줄이 directive에 해당한다. 그 나머지는 복수개의 노드 제약조건들이 출력된 후 한 개의 셰이프 제약조건이 출력되는 형식이 6회 반복되고 있다. 이 반복의 단위는 1개의 테이블로부터 생성된 ShEx 제약조건들이다. xsd:integer 타입의 노드 제약조건들이 제한하는 값의 범위는 모두 동일한데 이것은 MariaDB에서의 INT 타입의 범위이다. xsd:string 타입의 노드 제약조건들에 나타나는 최대 문자열 길이 제약은 이들 제약조건을 생성케 한 컬럼의 최대 문자열 길이 정보로부터 추출한 결과이다. foo:tweetsWhen 노드 제약조건은 정규표현식을 포함한다. 이 식은 MariaDB에서의 TIMESTAMP 타입의 범위인 1970-01-01 00:00:01부터 2038-01-19 05:14:07을 반영한다. projectsShape과 tweetsShape 셰이프 제약조건의 경우 노드 종류가 BNODE 인데 이것은 projects와 tweets 테이블이 기본 키가 없는 테이블이기 때문이다. 이 둘을 제외한 나머지 셰이프 제약조건들의 경우 노드 종류가 IRI이며 이어서 그 IRI에 대한 정규 표현식 제약이 정의되어 있음을 확인할 수 있다. 모든 셰이프 제약조건의 중괄호 내부는 Type1, 2, 3, 4 순으로 트리플 제약조건이 출력되어 있다. predicate이 “a”인 것이 Type 1이며 노드 제약조건을 참조하는 것들이 Type 2, 셰이프 제약조건을 참조하는 것들이 Type3과 4이다. 그 중에서 첫 문자가 “^”인 것이 Type 4 그렇지 않은 것이 Type 3이다.

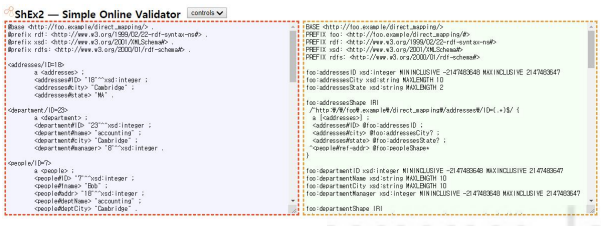

Fig. 14. RDF and ShEx Files Loaded on Validator

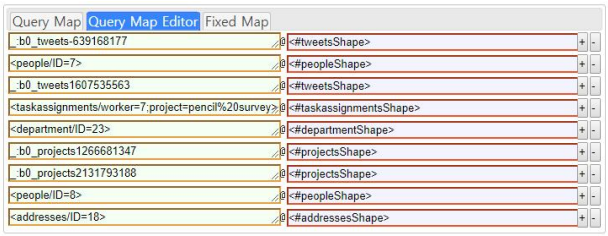

Fig. 15. A Snapshot of Query Map Editor

✓_:b0_tweets-639168177@<#tweetsShape>
✓<people/ID=7>@<#peopleShape>
✓_:b0_tweets1607535563@<#tweetsShape>
✓<taskassignments/worker=7;project=pencil%20survey>@<#taskassignmentsShape>
✓<department/ID=23>@<#departmentShape>
✓_:b0_projects1266681347@<#projectsShape>
✓_:b0_projects2131793188@<#projectsShape>
✓<people/ID=8>@<#peopleShape>
✓<addresses/ID=18>@<#addressesShape>

Fig. 16. The Result Snapshot of the Validation

그림 14~16은 direct_mapping 데이터베이스로부터 생성한 RDF 그래프의 구조를 그림 13의 ShEx 스키마를 기준으로 검증한 모습이다. 검증은 ShEx2 Simple Online Validator [18]에서 수행하였다. 그림 14의 좌측 박스에는 그림 13에 해당하는 ShEx 스키마 파일이 로딩되어 있으며 우측 박스에는 RDF 파일이 로딩되어 있다. 이 RDF 파일은 그림 5의 RDF Mapping Metadata와 RDF Writer 모듈을 사용해 생성하였다. 그림 15는 질의 맵 에디터 상에서 9개의 RDF 노드를 검증할 셰이프를 지정하고 있다. 이것은 본 데모를 위해 생성한 RDF 파일에 존재하는 모든 주어 노드를 검증하는 것이다. 그 이유는 direct_mapping 데이터베이스가 9개의 행을 포함하고 있기 때문이다. 그림 16은 9개의 노드에 대한 검증 결과 출력으로서 모든 노드가 제약조건을 만족함을 보고하고 있다.

V. Conclusion and Future Work

본 논문을 통해 데이터베이스로부터 Direct Mapping 알고리즘을 사용해 생성한 RDF 그래프에 대한 구조 검증이 가능한 ShEx 스키마를 생성하는 방법을 상세히 소개하였다. 이 연구의 목적은 DBLP[3]과 같이 Direct Maping 알고리즘에 의해 생성된 RDF 그래프를 사용하는 사용자들에게 자동 생성된 ShEx 스키마를 제공함으로써 ShEx 본연의 목적인 RDF 그래프에 대한 구조 검증을 용이하게 할 수 있도록 함과 동시에 RDF 그래프에 대한 구조 정보를 제공함으로써 SPARQL 질의문 작성을 보다 효율적으로 수행할 수 있도록 함이다. 이러한 목적을 보다 온전히 달성하기 위하여 본 연구를 기반으로 R2RML[19]를 통해 생성된 RDF 그래프에 대한 ShEx 스키마를 자동 생성하는 후속 연구를 진행할 것이다. 또한 ShEx 스키

마 정보를 텍스트가 아닌 GUI 인터페이스로 제공하여 SPARQL 질의 작성을 돕는 질의 도구 또한 개발할 계획이다.

본 연구의 구현은 다음의 URL에 공개되어 있다. https://github.com/jwchoi/OWL4RDB/tree/ShExForDirectMapping

REFERENCES

[1] Linking Open Data W3C SWEO Community Project, *https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData*

[2] DBpedia, *http://wiki.dbpedia.org/*

[3] DBLP, *https://dblp.uni-trier.de/*

[4] T. Tran, G. Ladwig and S. Rudolph, "Managing Structured and Semistructured RDF Data Using Structure Indexes," IEEE Trans. on Knowledge and Data Engineering, Vol. 25, Issue. 9, pp. 2076-2089, Sept. 2013.

[5] SHACL, *http://www.w3c.org/TR/shacl/*

[6] ShEx, *http://shex.io/shex-semantics/*

[7] A Direct Mapping of Relational Data to RDF, *http://www.w3c.org/TR/rdb-direct-maipping/*

[8] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva and D. Kontokostas, "Validating RDF Data" Morgan & Claypool Publishers, 2017.

[9] D. Arndt, B. D. Meester, A. Dimou, R. Verborgh, and E. Manneus, "Using Rule-Based Reasoning for RDF Validation," Proceedings of International Joint Conference on Rules and Reasoning, pp. 22-36, 2017.

[10] S. Han, L. Zou, J. X. Yu, and D. Zhao, "Keyword Search on RDF Graph - A Query Graph Assembly Approach," Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 227-236, 2017.

[11] A. Abbas, P. Genevès, C. Roisin and N. Layaïda, "SPARQL Query Containment with ShEx Constraints," Proceedings of the 21st European Conference on Advances in Databases and Information Systems, pp. 343-356, 2017.

[12] J. Corman, J. L. Reutter and O. Savković, "Semantics and Validation of Recursive SHACL," Proceedings of the 17st International Semantic Web Conference, pp. 318-336, 2018.

[13] XML Schema Part 2: Datatypes Second Edition, *http://www.w3c.org/TR/xmlschema-2/*

[14] XPath 3.1 Regular expression, *http://www.w3c.org/TR/xpath-functions-31/#regex-syntax*

[15] Natural Mapping of SQL Values, *https://www.w3.org/TR/2012/REC-r2rml-20120927/#natural-mapping*

[16] SPARQL 1.1 Update, *https://www.w3.org/TR/sparql11-update/*

[17] MariaDB, *http://mariadb.com*

[18] ShEx2 - Simple Online Validator, https://rawgit.com/shexSpec/shex.js/master/doc/shex-simple.html

[19] R2RML: RDB to RDF Mapping Language, http://www.w3.org/TR/r2rml/

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 Assistant Professor in the School of Computer Science and Engineering, Soongsil University. He is interested in information system.