

What Factors Influence the Design of a Linked Data Generation Algorithm?

Anastasia Dimou

anastasia.dimou@ugent.be

IDLab, Dep. of Electronics and Information Systems,
imec – Ghent University

Ben De Meester

ben.demeester@ugent.be

IDLab, Dep. of Electronics and Information Systems,
imec – Ghent University

Pieter Heyvaert

pieter.heyvaert@ugent.be

IDLab, Dep. of Electronics and Information Systems,
imec – Ghent University

Ruben Verborgh

ruben.verborgh@ugent.be

IDLab, Dep. of Electronics and Information Systems,
imec – Ghent University

ABSTRACT

Generating Linked Data remains a complicated and intensive engineering process. While different factors determine how a Linked Data generation algorithm is designed, potential alternatives for each factor are currently not considered when designing the tools' underlying algorithms. Certain design patterns are frequently applied across different tools, covering certain alternatives of a few of these factors, whereas other alternatives are never explored. Consequently, there are no adequate tools for Linked Data generation for certain occasions, or tools with inadequate and inefficient algorithms are chosen. In this position paper, we determine such factors, based on our experiences, and present a preliminary list. These factors could be considered when a Linked Data generation algorithm is designed or a tool is chosen. We investigated which factors are covered by widely known Linked Data generation tools and concluded that only certain design patterns are frequently encountered. By these means, we aim to point out that Linked Data generation is above and beyond bare implementations, and algorithms need to be thoroughly and systematically studied and exploited.

1 INTRODUCTION

Generating Linked Data remains a complicated and intensive engineering process, despite the significant number of existing tools. Most solutions primarily choose their own (often non-interoperable) approach. *Format-* and *source-specific* approaches were investigated as more generic alternatives [4]. In all cases, rules define how Linked Data should be generated. These rules often remain *implicit* and embedded in the implementation e.g., the DBpedia Extraction Framework, but more generic solutions distinguish them and turn them *explicit* and declarative, e.g., [R2]RML processors.

The rules cover a use case's context, whereas the *execution algorithm* covers its technical and functional requirements. On the one hand, each use case's context—how the Linked Data is modeled or which vocabularies are used to generate Linked Data—is described within the rules and does not influence the algorithm's design. For instance, the rules consider the adequate ontology terms to annotate the data. On the other hand, different *factors* related to technical and functional requirements of each use case determine how an

algorithm should be designed as well as how third-parties choose the most adequate tool. Potential alternatives for these *factors* affect how efficiently the rules are executed to generate Linked Data. For instance, “What is the *purpose*? Is the Linked Data consumed immediately or is it published for future use?” or “What *triggers* the generation? Is the Linked Data generated from a real-time data stream which needs to be immediately processed, or on demand?”.

Certain design patterns are noticed to be frequently applied across different tools, covering particular alternatives of these factors, whereas other alternatives or factors were never explored. For instance, if we lack tools whose algorithms support real-time data streams, Linked Data can be generated by storing the data in a database and using corresponding tools, e.g., Morph-streams. Thus, there are often no adequate tools for Linked Data generation for certain occasions or tools with inadequate and inefficient algorithms are chosen as best alternatives.

However, those factors were not thoroughly and systematically studied so far, nor were the algorithms' designs. Different solutions do not concretely describe the algorithms that drive their implementation while optimizations are applied according to specific use cases, decreasing a tool's chances to be reused. The algorithms are not designed considering different factors and the use case's technical and functional requirements are not matched to any factors.

In this work, we present a preliminary list of factors that could be considered. We do not aim for a complete list. This is a position paper whose goal is to provide insights and raise awareness, as Linked Data generation goes beyond and above bare implementations. We reviewed a few of the pioneering and broadly used tools that cover one or more of these factors and discuss the results of our observations. By these means, we aim to ensure that one can (i) choose or design the most adequate algorithm, and (ii) generate Linked Data without being restricted by tooling limitations.

The remainder of the paper is structured as follows: In Section 2, we discuss the preliminary list with different factors that we identified and in Section 3 we investigate which factors each tool covers in Section 4, we outline our conclusions.

2 EXECUTION FACTORS

Different factors determine how an algorithm should be designed for generating Linked Data. A *factor* can be any fact or circumstance that contributes to the envisaged result, i.e., the Linked Data

Table 1: Factors affecting Linked Data generation, when it occurs and which the associated elements are.

factor	element		generation's execution	
	data	rules	before	during
purpose			✓	
direction	✓		✓	
materialization			✓	
location	✓			✓
driving force	✓	✓	✓	
trigger			✓	
dynamicity	✓		✓	✓
diversity	✓		✓	✓
complexity	✓	✓	✓	

generation. Those factors are related to (and often dependent on) the elements involved in a Linked Data generation activity:

data both *raw data* to generate the desired Linked Data from, as well as existing Linked Data.

rules mapping rules that define how Linked Data are generated relying on available data.

tools tools that apply mapping rules to data and generate Linked Data.

Multiple factors determine how Linked Data is generated fulfilling different technical or functional requirements posed by different use cases. The different factors that we identified are outlined below and summarized in Table 1. For each factor, we outline the two furthest alternatives to shed light on the options, but intermediate or hybrid approaches may be adopted as well. The differences are determined depending on what the generation purpose is (Section 2.1), its direction (Section 2.2) and materialization (Section 2.3), where it occurs (location, Section 2.4), what drives (Section 2.5) and what triggers the execution (Section 2.6), how dynamic (Section 2.7) or diverse the data is (Section 2.8), and the data or rules complexity (Section 2.9).

Use case. Let us consider a use case that illustrates each factor with the help of an example. The use case is about an intelligent transportation search engine, which relies on Linked Data derived from heterogeneous data sources. The search engine obtains information about airports from an airline data source, about train stations from a train data source and about the location of countries, cities, and addresses from a data source with spatial data.

2.1 Purpose

Different purposes can prompt the Linked Data generation. On a high level, we identify: *production* and *consumption*. The purpose that drives the generation affects the design choices of the execution algorithm, but remains independent of the involved elements (data or rules). The fundamental difference lies on the extend of use cases that the Linked Data generation task aims to cover:

Production Linked Data generation can be driven by a production need, i.e., a *data owner* generates Linked Data to annotate and turn the data publicly available. *Production-driven*

generation remains independent of the data's potential consumption which should then be adjusted to the Linked Data as it becomes available.

Consumption Linked Data generation can occur due to certain consumption needs, namely a *data consumer* requires to process Linked Data which still need to be generated from raw data. Thus, the generated Linked Data is the response for a particular consumption need.

Example. NMBS, the Belgian train provider, has a legal obligation to publish information about train stations. Different data consumers can profit of this Linked Data, which is already *produced*, to build intelligent applications adjusted to the already generated Linked Data. The Belgian Airlines, Belgium's national airlines, want to identify all airports where its airplanes fly to. This *consumption* need leads to generating Linked Data specifically for this purpose.

2.2 Direction

Linked Data generation might follow different directions [10], which are determined by the available data:

Target-centric The execution is focused on describing a set of views over the data source(s). The approach is same as the Global-As-View (gav) formalism for data integration [6, 11]. When mapping among different data models, it is possible to define one of the data models as a view onto the other data model [10]. The target might be (i) a certain *graph pattern* derived from existing Linked Data, whose schema is desired to be replicated; (ii) a given query (*results-driven* editing approach [8]); (iii) a given *schema* (a combination of ontologies and vocabularies – *schema-driven* editing approach [8]); or (iv) a set of mapping rules (*model-driven* editing approach [8]). For instance, a *data owners* has a data source, while other data is already described as Linked Data. The data owner then generates its own Linked Data, considering a certain target.

Source-centric The execution is focused on describing the entities of each data source, independently of other data sources (*data-driven* editing approach [8]). The approach is similar to the Local-as-View (lav) formalism for data integration [6, 11], as a mapping occurs from the original data source(s) to the mediated schema (Linked Data or schema), making it easier to add and remove data sources. For instance, a *data owner* has two data sources. She defines rules to semantically annotate those data sources, without being concerned about similar or complementary data which is already available as Linked Data.

The direction is determined before the generation activity is triggered, and depends mainly on the available data and rules. Different execution algorithms may be designed that support either the one or the other, or both directions.

Example. Following our use case, the Belgian Airlines specify a set of SPARQL queries which act as the *target*. The rules are defined for each data source specifically, so the resulting Linked Data matches the SPARQL queries' graph patterns. NMBS specified a set of rules to generate its own Linked Data from its own available sources (*source-centric*).

2.3 Materialization

In relational databases, views simplify a database's conceptual model with the definition of a virtual relation [1]. A *materialized view* is a database that contains results, while the process of setting up a materialized view is called *materialization* [1]. To achieve this, different materialization strategies exist [7]. In the same context, the Linked Data generation materialization differs on when the consumption occurs, i.e., *dumping* or *on-the-fly* [10], affecting the corresponding algorithms. On the former case, long term consumption is expected, whereas, on the latter, direct. The materialization, as the purpose of execution, does not depend on the elements involved in the Linked Data generation, and it impacts before the Linked Data is generated.

Dumping A *data dump* is generated into a volatile or persistent triplestore, aiming to provide a view of the data (similar to a *materialized view* in relational databases).

On-the-fly This occurs when the Linked Data generation takes place *on-the-fly* (as a *non-materialized view*).

Example. NMBS *dumps* the train station Linked Data in a triplestore, which is used for storing and retrieving Linked Data, whereas the Belgian Airlines generates the airports Linked Data *on-the-fly* when a query is executed without storing it.

2.4 Location

The elements involved in Linked Data generation might reside on different sites. The fundamental difference lies in where the data and rules reside, and where the execution takes place. That is determined before the Linked Data generation is initiated and affects how the algorithms are designed. For instance, how the input data is retrieved or processed differs. We identify the following:

In-situ Linked Data generation is performed *in-situ* when it is addressed by the same site that holds both the tool and data. For instance, a *data owner* has the data and rules locally stored and in the same place as the tool that executes the rules to generate the Linked Data.

Remote Linked Data generation occurs *remotely* when the tool does not reside on the same site as the data and rules. For instance, the tool is a remote service, e.g., Software-as-a-Service (SaaS). To the contrary, the tool may reside locally, but the data and rules not.

Example. The train stations Linked Data is generated *in-situ*, as both the tool and data might be on the same site. To the contrary, the data for airports might reside *remotely* from the site where the tool to generate the Linked Data is.

2.5 Driving force

The rules to generate Linked Data can be executed using alternative driving forces [4], namely *rules* and *data*, or any combination of the two (*hybrid*), and algorithms are affected depending on the element that drives the Linked Data generation. Which approach is followed depends either on the data or rules.

Mapping-driven The processing is driven by rules which prompt the Linked Data generation and adequate data is employed. For instance, a *data consumer* poses a query that

is translated to rules or directly provides rules based on which Linked Data is generated.

Data-driven The execution is driven by data which prompts the Linked Data generation and adequate rules are executed. Once this data reaches a Linked Data generation tool, a new execution is triggered to generate Linked Data according to rules associated to this data.

Example. Once an updated version of the train stations is available, the data might be sent to a Linked Data generation tool and prompts a new generation round (*data-driven*). The airports Linked Data generation is triggered by the rules which specify the corresponding data sources (*mapping-driven*).

2.6 Trigger

Linked Data generation can occur *real-time* or *ad-hoc* [9]. While it is independent of the elements involved in Linked Data generation, as it occurs with the purpose and materialization, it affects the algorithms design, e.g., *real-time* execution requires timely generation.

Real-time *Real-time* execution is related to the notions of event, i.e., “any occurrence that results in a change in the sequential flow of program execution” and response time, i.e., “the time between the presentation of a set of inputs and the appearance of all associated outputs”.

On-demand *On demand* execution occurs if agents trigger the execution to generate Linked Data when desired.

Example. The train stations generation occurs *real-time*, as every time the data is updated, new Linked Data is generated. If the train stations Linked Data is not generated every time a new version is available, its generation occurs *on-demand*.

2.7 Dynamicity

A data source's dynamicity might differ, influencing how the Linked Data generation occurs. Thus, it affects how an algorithm is designed and a corresponding tool is implemented. For instance, the memory allocation is influenced. This factor depends on the data, but not on the rules, and affects the generation both before and while it is executed.

Static data A static data structure refers to a data collection that has a certain size.

Dynamic data A dynamic data structure refers to a data collection that has the flexibility to grow or shrink in size. For instance, it might not be possible to obtain all data, as the data can be infinite in size.

Example. The train stations original dataset is *static*: when the Linked Data generation is triggered, the original raw dataset's size is known. The airport's dataset is *dynamic*: its returned size is not foreseen, as it depends on a query's answers.

2.8 Diversity

Linked Data generation may occur based on a single or multiple data sources. The different data sources might be homogeneous or heterogeneous with respect to their *structure*, e.g., tabular, hierarchical or attribute-value pairs, their *format*, e.g., CSV, ML, JSON, or their *access interface*, e.g., database connectivity, Web APIs or local

files [5]. The diversity factor influences the Linked Data generation both *before*, e.g., what is supported, and *during* the execution, e.g., how heterogeneous data sources are aligned.

Homogeneity Data with same data structure and format.
Heterogeneity Data with different structures and formats.

2.9 Complexity

Data or rules complexity affects the algorithm's design.

Data The original dataset's size or e.g., the depth of a data source which is hierarchically-structured can influence how the Linked Data generation is accomplished. For instance, big datasets require to be treated differently than smaller, as parallelization or distribution might be preferred which might be an overhead for smaller datasets.

Rules The rules complexity might be affected by e.g., the desired transformations and (cross-sources) joins.

All in all, the *purpose*, *direction*, *materialization*, *driving force*, and *trigger* affect the Linked Data generation *before* the execution occurs, whereas the *location*, and *complexity* affect during execution, while the *dynamicity* and *diversity* influence both before and during. All these should be taken into consideration when designing the corresponding algorithms.

3 TOOLS

We outline the pioneering and broadly used open source rule-based tools for Linked Data generation which support the W3C recommended R2RML language [3] or its extension for heterogeneous data sources, RML [4]. We investigate which factors each tool covers and we discuss the results.

DB2triples. DB2Triples¹ is a tool for extracting data from relational databases, semantically annotating the data extracts according to R2RML rules and generating Linked Data. It implements the two W3C specifications for generating Linked Data from databases, i.e., R2RML [3] and Direct Mapping [2]. It is an open-source system released under GNU Lesser General Public License, version 2.1².

DB2Triples is adequate for generating Linked Data for *production*, but not for *consumption*. It is a command-line tool that *dumps* the generated Linked Data to a file. It only considers *local* (*in-situ*), *homogeneous* and *static* databases. Its function is prompt by the *mapping* and occurs *on-demand*, while it does not address neither data nor rules complexity.

Morph. Morph³ is a tool for Linked Data generation from data residing in relational databases. It supports (i) *data upgrade*, which generates Linked Data from a relational database, according to certain R2RML mapping rules; and (ii) *query translation*, which evaluates SPARQL queries over virtual Linked Data, by rewriting those queries into SQL. Morph employs a query translation algorithm from SPARQL to SQL with different optimizations during the query rewriting process, to generate more efficient SQL queries. It is an open-source system released under Apache License, Version

2.0. Morph-streams⁴ is an extension over Morph for evaluating SPARQL-Stream queries over a range of data streams. It allows to register SPARQL-Stream continuous queries over an R2RML-wrapped data source, apply query-rewriting and receive updated results as soon as the queries are evaluated.

Morph allows to generate Linked Data for both *production* and *consumption* by both *dumping* the Linked Data, when they are generated for production, and consuming *on-the-fly* them, when they are generated for consumption. Similarly to DB2triples, Morph may only be used with *in-situ* (except for CSV files which can be *remote* and get accessed via HTTP) and *homogeneous* raw data, but it can support both *dynamic* and *static* data. Morph functions *mapping-driven* and *on-demand*. To a certain extend, Morph tries to address the *query translation* (SPARQLtoSQL) complexity. Morph-streams generates Linked Data from both *static* and *dynamic* data, *on-demand* and *real-time* from heterogeneous data sources imported though in a *homogeneous* database. Morph-streams tries to address complexity with respect to *query-rewriting*.

Ontop. Ontop⁵ is another tool that allows to query relational databases as Virtual RDF Graphs using SPARQL, as Morph does too. It translates SPARQL queries into Datalog rules before transforming them into SQL queries (query-translation). Similarly to Morph, Ontop covers the same factors and tries to address complexity with respect to *query translation*.

R2RMLParser. The R2RMLParser⁶ is a tool that relies on R2RML mapping rules to generate Linked Data from relational databases. The R2RML Parser deals in principle with incremental Linked Data generation. In more details, each time a Linked Data generation task is executed, not all of the input data should be used, but only the one that changed (so-called incremental transformation). The R2RMLParser is released under the Creative Commons Attribution-NonCommercial 4.0⁷ license.

The R2RMLParser can be characterized as *mapping-driven* and *on-demand*. The generation occurs for *production* reasons and the Linked Data is *dumped* when generated. As the aforementioned tools which are focused on relational databases, the R2RML parser focuses on *local*, *homogeneous* raw data. However, it seems to address to a certain extend, the *dynamicity* (both *static* and *dynamic*) and *complexity* of the data with respect to time.

XSPARQL. XSPARQL⁸ performs dynamic query translation to generate Linked Data from different sources. XSPARQL primarily provides a query-driven approach that combines XQuery [17] and SPARQL [34, 47]. This way, it allows to query data in XML and RDF using the same framework, and supports both the generation of RDF from XML (lifting), and XML from RDF (lowering). XSPARQL was extended to also support Linked Data generation from databases combining SQL and SPARQL via R2RML rules, but Linked Data cannot be generated from both XML and databases.

XSPARQL does support *heterogeneous* data to a certain extend, but it is limited to data in XML format and relational databases.

⁴Morph-streams, <https://goo.gl/FYr9Lc>

⁵Ontop, <https://github.com/ontop/ontop>

⁶R2RMLParser, <http://github.com/nkons/r2rml-parser>

⁷Creative Commons Attribution-NonCommercial 4.0, <http://creativecommons.org/licenses/by-nc/4.0/>

⁸XSPARQL, <http://xsparql.derri.org/>

¹DB2triples, <https://github.com/antidot/db2triples>

²GNU LGPL, version 2.1, <https://goo.gl/Wi7qbV>

³Morph, <https://goo.gl/JtAyFL>

Extending it to support other *heterogeneous* data requires new pipelines, as each format is separately addressed and combination of heterogeneous data is not feasible. Otherwise, XSPARQL is a *consumption-driven* tool which generates Linked Data *on-the-fly*, relying on *local* data and is prompt *on-demand* by the rules.

RMLMapper. The RMLMapper⁹ is an RML Engine, i.e., a rule-based Linked Data generator for data sources accessed using different protocols containing data in various structures, formats, and serializations, e.g., CSV, XML and JSON. It is written in Java and can be used on its own via a command-line interface or its modules separately in different interfaces, e.g., as a library or remote service. It is released under MIT license.

In contrast to the tools mentioned above, the RMLMapper focuses on *heterogeneous* and both *local* and *remote* data to generate Linked Data. However it still deals with *static* data and it does not optimize neither data or rules complexity. It follows an *on-demand* and *mapping driven* approach and *dumps* the data in a file or any other triplestore.

CARML. CARML¹⁰ is also an RML Engine. It is developed as a Java library that transforms (semi-)structured sources to RDF based on rules declared in RML. More precisely, it supports data in CSV, JSON and XML format. It is an open-source system released under MIT license¹¹ that takes a static input and streams it to generate the corresponding Linked Data.

CARML follows the same principles as the RMLMapper. It also focuses on *heterogeneous* data, follows the *mapping-driven* approach, and generates Linked Data *on-demand*. It functions with *static* that streams them to generate Linked Data. Nevertheless, as most of the other tools do not optimize the *data* or *rules* complexity.

4 CONCLUSIONS

Overall, we observe patterns, i.e., correlations among different factors or certain of their alternatives repeat over different tools.

Tools which are *consumption-driven* typically function both with *static* and *dynamic* data but only with *homogeneous* data. *Consumption-driven* tools may be used for *production* purposes but they are not optimized for that purpose and cannot handle *heterogeneity*.

The *dynamicity* is only addressed by *consumption-driven* implementations in the form of dynamic data that answer a certain query. Even though it is not obvious from the aforementioned, the extend to which the *consumption-driven* tools address dynamicity do not adhere well with the *complexity*, in particular of data, e.g., its size.

Production-driven tools support *heterogeneous* data but typically do not support *real-time* generation. Only the most recent, CARML, focuses on Linked Data generation from *dynamic* data. However, even then, the dynamicity is caused from otherwise *static* data.

In general, there are no tools which support the *data-driven* approach, as there are no tools which support *real-time* data.

Moreover, none of the tools put effort into optimizing the complexity of the data or rules nor do they optimize their generation algorithms. The *consumption-driven tools*, i.e., Morph and Ontop,

are the only tools which optimize the *query translation*, when generating Linked Data, while Morph-streams optimizes the *query rewriting*. Query rewriting and translation may be considered as partially handling the *rules and data complexity*.

Among the *production-driven* tools, none addresses *complexity*. The R2RMLparser is the only one that aims to address to a certain extend the *data complexity*, in its case with respect to time.

Even though data complexity is studied in data mining, neither results from these studies are applied to Linked Data generation algorithms nor such algorithms are investigated in this context.

Overall, lack of in-depth understanding of Linked Data generation complexity and the many degrees of freedom in designing algorithms to generate Linked Data prevents human and software agents from effortless generating and directly profiting of large amounts of Linked Data for use with Semantic Web technologies.

With this position paper, which is not meant to be complete with respect to the factors or tools, we aim to show the diversity of factors that influence the Linked Data generation and the limited spectrum that is covered by current tools. We intent to raise awareness that the algorithms which drive the Linked Data generation should be more systematically studied so as human and software agents to be able to effortlessly and efficiently generate Linked Data.

In the future, we aim to study more thoroughly these factors and their alternatives. We hope that the factors and their alternatives will be exploited, more diverse algorithms will be designed and more efficient tools for Linked Data generation will be developed.

REFERENCES

- [1] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden. Materialization strategies in a column-oriented DBMS. In *Data Engineering, 2007. IEEE 23rd International Conference on*, 2007.
- [2] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. A Direct Mapping of Relational Data to RDF. W3C Recommendation, W3C, Sept. 2012.
- [3] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Rec, Sept. 2012.
- [4] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Workshop on Linked Data on the Web*, 2014.
- [5] A. Dimou, R. Verborgh, M. Vander Sande, E. Mannens, and R. Van de Walle. Machine-interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval. In *Proceedings of the 11th International Conference on Semantic Systems*, 2015.
- [6] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. 2012.
- [7] E. N. Hanson. *A performance analysis of view materialization strategies*. 1987.
- [8] P. Heyvaert, A. Dimou, R. Verborgh, E. Mannens, and R. Van de Walle. Towards Approaches for Generating RDF Mapping Definitions. In *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, volume 1486, 2015.
- [9] N. Konstantinou, D.-E. Spanos, D. Kouis, and N. Mitrou. An approach for the Incremental Export of Relational Databases into RDF Graphs. *International Journal on Artificial Intelligence Tools*, 24, 2015.
- [10] A. Langegger and W. Wöß. XLWrap – querying and integrating arbitrary spreadsheets with SPARQL. In *The Semantic Web - ISWC 2009: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009*, 2009.
- [11] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246, 2002.

APPENDIX

⁹RMLMapper, <http://github.com/RMLio/RML-Mapper>

¹⁰CARML, <https://github.com/carmil/carmil>

¹¹MIT license, <https://opensource.org/licenses/MIT>

Table 2: Linked Data generation tools, mapping language and supported input formats

	carml	DB2triples	Morph	Ontop	R2RMLparser	RMLMapper	XSPARQL
language							
R2RML	-	✓	✓	✓	✓	✓	✓
RML	✓	-	-	-	-	✓	-
input							
relational database	-	✓	✓	✓	✓	✓	✓
CSV	✓	-	✓	-	-	✓	-
JSON	✓	-	-	-	-	✓	-
XML	✓	-	-	-	-	✓	✓

Table 3: Linked Data generation tools and factors

	carml	DB2triples	Morph	Ontop	R2RMLparser	RMLMapper	XSPARQL
factor purpose							
production	✓	✓	✓	✓	✓	✓	-
consumption	-	-	✓	✓	-	-	✓
materialization							
dumping	✓	✓	✓	✓	✓	✓	✓
on-the-fly	-	-	✓	✓	-	-	n/a
location							
in-situ	✓	✓	✓	✓	✓	✓	✓
remote	✓	-	-	-	-	✓	n/a
driving force							
mapping	✓	✓	✓	✓	✓	✓	✓
data	-	-	-	-	-	-	-
trigger							
real-time	-	-	-	✓	-	-	-
on demand	✓	✓	✓	✓	✓	✓	✓
dynamicity							
static	✓	✓	✓	✓	✓	✓	✓
dynamic	-	-	✓	✓	-	-	✓
diversity							
homogeneity	✓	✓	✓	✓	✓	✓	-
heterogeneity	✓	-	-	-	-	✓	✓
complexity							
data	-	-	-	-	(✓)	-	-
rules	-	-	(✓)	-	-	-	-