RDFAdaptor: Efficient ETL Plugins for RDF Data Process

Jiao Li^{1*}, Guojian Xian^{1,2*}, Ruixue Zhao^{1,2}, Yongwen Huang¹,
Yuantao Kou^{1,2}, Tingting Luo¹, Tan Sun^{2,3†}

¹Agricultural Information Institute of CAAS, Beijing 100081, China

²Key Laboratory of Agricultural Big Data, Ministry of Agriculture and Rural Affairs, Beijing 100081, China

³Chinese Academy of Agricultural Sciences, Beijing 10081, China

Citation: Li, J., Xian, G.J., Zhao, R.X., Huang, Y.W., Kou, Y.T., Luo, T.T., & Sun, T. (2021). RDFAdaptor: Efficient ETL plugins for RDF data process. *Journal of Data and Information Science*, 6(3), 123–145. <https://doi.org/10.2478/jdis-2021-0020>

Received: Dec. 18, 2020

Revised: Feb. 26, 2021

Accepted: Mar. 9, 2021

Abstract

Purpose: The interdisciplinary nature and rapid development of the Semantic Web led to the mass publication of RDF data in a large number of widely accepted serialization formats, thus developing out the necessity for RDF data processing with specific purposes. The paper reports on an assessment of chief RDF data endpoint challenges and introduces the RDF Adaptor, a set of plugins for RDF data processing which covers the whole life-cycle with high efficiency.

Design/methodology/approach: The RDFAdaptor is designed based on the prominent ETL tool—Pentaho Data Integration—which provides a user-friendly and intuitive interface and allows connect to various data sources and formats, and reuses the Java framework RDF4J as middleware that realizes access to data repositories, SPARQL endpoints and all leading RDF database solutions with SPARQL 1.1 support. It can support effortless services with various configuration templates in multi-scenario applications, and help extend data process tasks in other services or tools to complement missing functions.

Findings: The proposed comprehensive RDF ETL solution—RDFAdaptor—provides an easy-to-use and intuitive interface, supports data integration and federation over multi-source heterogeneous repositories or endpoints, as well as manage linked data in hybrid storage mode.

Research limitations: The plugin set can support several application scenarios of RDF data process, but error detection/check and interaction with other graph repositories remain to be improved.

Practical implications: The plugin set can provide user interface and configuration templates which enable its usability in various applications of RDF data generation, multi-format data conversion, remote RDF data migration, and RDF graph update in semantic query process.

* These authors have contributed equally to this work.

† Corresponding author: Tan Sun (E-mail: suntan@caas.cn).

Research Paper

Originality/value: This is the first attempt to develop components instead of systems that can include extract, consolidate, and store RDF data on the basis of an ecologically mature data warehousing environment.

Keywords RDF ETL solution; RDF data processing; Linked data; Portable plugins

1 Introduction

The fundamental standard for the Semantic Web is Resource Description Framework (RDF) (Cyganiak, Wood, & Lanthaler, 2014), a machine-understandable data modeling language built upon the notion of statements about resources with triple form <subject, property, object> (SPO), in which the subject represents the resource being described, the property denotes the predicate, and the object contains the value associated with the property for the given subject or a simple value (called literal). RDF is designed to use among resources mapped at hierarchic levels in a graph-based representation, and now is widely used in Semantic Web applications for managing large bases and data ontologies. With the exponential growing of information, the Web has involved from a network of linked documents to one where both documents and data are linked, the advent of Linked Data (Bizer, Heath, & Berners-Lee, 2009) and its underlying technologies make data reuse and federation possible, and in particular RDF data seems promising for the process. In fact, the complexity of data processing is increasingly challenging, data consumers often deal with various triple data or even non-RDF data, e.g. internal or open data in relational databases or CSV, Excel, JSON, and XML files, so the tools or services support for multiple-source data processing which covers the whole life-cycle from data extraction & integration, storage, and querying to applications, is needed.

There are numerous tools used for Linked data preparation. RDF generators which support extracting RDF data from relational database (RDB): R2RML Parser^①, exports RDB contents as RDF graphs based on an R2RML mapping document. D2RQ^② offers an integrated environment with multiple ways to access relational data, including RDF dumps, SPARQL terminals, Sesame API access, and D2SQ servers. Triplify^③ is a PHP Web plugin, which reveals the semantic structures encoded in RDBs by making database content available as RDF, JSON or Linked Data. Many other tools for RDF data processing have also been proposed: Silk (Volz et al., 2009), a link discovery framework to interlink RDF data on the Web based on the declarative algorithms. OpenLink Virtuoso^④, a SQL-ORDBMS and

^① https://www.w3.org/2001/sw/wiki/R2RML_Parser

^② <https://www.w3.org/2001/sw/wiki/D2RQ>

^③ <https://www.w3.org/2001/sw/wiki/Triplify>

^④ <https://virtuoso.openlinksw.com/>

Web Application Server hybrid for storing RDF data and executing SPARQL queries (Erling & Mikhailov, 2009). Apache Jena[®], an open source Java framework, provides RDF API and triple store—TDB. It has also Fuseki which serves for exposing triples as a SPARQL endpoint accessible over HTTP and supports REST-style interaction with RDF data. Furthermore, to address the challenges of increasing alternative syntaxes for publishing RDF content, there exist a group of tools or services for the conversion between serialization formats for RDF: Triplr[®] is one of the earliest RDF format converters capable of guessing the format of input data. Any23[®] (Anything to triples) can be featured as a public REST-style Web Service for cool uniform resource identifiers (URIs), content negotiation. RDF2RDFa[®] (no longer maintained and kept available only for archival purposes) is used to turn RDF/XML content to RDFa schema. RDF translator[®], a RESTful Web service, aims at translating between the most popular serialization formats. Nevertheless, those above-mentioned tools/services are separated in different configurations and require a prepared script or programming task for executing data transformation/conversion or conflict resolution with limited source and storage supported.

Suppose a data consumer wants to build an RDF repository that integrates information from various non-RDF and RDF sources, or merge RDF data in different formats from multiple repositories with less or even no coding, a packaged and multifunctional RDF data process method will be much easier to use and perform more effective. None of the tools or services presented so far support such comprehensive and convenient RDF data processing. Therefore, we developed the RDFAdaptor—a set of Extract-Transform-Load (ETL)[®] plugins, which offers solutions for RDF data processing with straightforward models in good connectivity and scalability. It contains groups of steps or hop algorithm modules, which can be arbitrarily combined into specific functional tasks.

The rest of this paper is structured as follows: Section 2 gives an overview of the existing literature or of significance for the study area, and summarizes the current problems. Section 3 outlines the overall design process of the plugin architecture, and gives the implementation details. Section 4 is devoted to describe the main functionalities of the plugin set and application scenarios. In section 5 we describe some cases/experiments of the RDFAdaptor and discuss the experiences and

[®] <http://jena.apache.org/>

[®] <http://triplr.org/>

[®] <http://any23.apache.org/>

[®] <http://www.ebusiness-unibw.org/tools/rdf2rdfa/>

[®] <https://rdf-translator.appspot.com/>

[®] https://en.wikipedia.org/wiki/Extract,_transform,_load

lesson learnt. Finally, Section 6 concludes the paper and provides the directions for future work.

2 Related work

Since the publication and exponential growing of RDF data, maintaining data processing and management tasks of increasing complexity is challenging. Many researchers or data solution providers have attempted to fulfill the required outcomes of Semantic Web. In this section a brief overview of both types of existing frameworks for RDF data transformation, processing, and management are given.

2.1 Extract-Transform-Load frameworks

Extract-Transform-Load (ETL) is the common paradigm by which data from multiple systems is combined to a target database, data store, or data warehouse, with the task of dealing with data warehouse homogeneity, cleaning and loading problems. ETL arises from the fact that modern computing business data resides in various locations and in many compatible formats. It is a key process to bring all the data together in a standard, homogeneous environment, managing and processing such huge collection of interconnected data costs 55% of the total data warehouse runtime (Inmon, 1997). There exist a multitude of open-source ETL frameworks / tools, most of them are non-RDF frameworks—for example Apache Camel[®], Apatar[®], or GeoKettle[®], which are mainly supporting stream data.

Pentaho Data Integration[®] (also named Kettle) is a leading open-source ETL application on the market, and has been slightly modified to four elements—ETTL, performing: (1) extraction of data from original data sources, (2) transport of the data, (3) transformation of the data to the proper format and structure, and (4) loading of the incoming data into the target data source to an operational data store, or data warehouse/data mart. It is a set of Java-based tools and applications, and comprises of main components like Spoon (a graphical tool), Pan (data transformation application), Chef (job create tool), Kitchen (an application helps execute jobs in a batch mode), and Carte (a web server for remote monitor). The data sources and supported databases of Pentaho Data Intergration include any database using ODBC on Windows, Oracle, AS/400, MySQL, MS SQL Server, et al. Parallel processing is implemented in the software, which enables the possibility to improve overall efficiency performance of ETL when dealing with data in large scale.

[®] <https://camel.apache.org/>
[®] <http://www.apatar.com/>
[®] <http://www.spatialytics.org/projects/geokettle/>
[®] <https://etl-tools.info/en/pentaho/kettle-etl.htm>

Non-RDF ETL frameworks are the key approaches to perform ETL in relational databases or other data formats (e.g. CSV/Excel, XML files), while not able to process RDF data, such as exchange RDF data among systems, extract RDF data from external SPARQL endpoints, transform RDF data from/to other format, or load RDF data to external SPARQL endpoints. As RDF data gains traction, the proper support for its processing and management is more important than ever. Linked Data Integration Framework (LDIF) (Schultz et al., 2011) is an open-source Linked data integration framework aiming at Web data transformation with a predefined set of *data processing units* (DPUs), but not user-friendly for new DPUs cannot be added easily. UnifiedViews, an RDF-enabled ETL tool, allows users to define, execute, monitor, debug, schedule, and share linked data processing and publishing tasks with plenty of simple-create DPUs (Knap et al., 2015; Knap et al., 2018). It contains three main components: Graphical user interface, Pipeline execution engine, and REST API administration service. However, the shortcoming of inappropriate mapping needs to be addressed in practice. Furthermore, LinkedPipes ETL (LP-ETL) (Klímeck, Škoda, & Nečaský, 2016), a lightweight Linked Data preparation tool, is developed based on the work of UnifiedViews and focuses on APIs and configurations, while does not support relational data units.

2.2 RDF data management

2.2.1 RDB-to-RDF technologies

RDB (Relational Database) is the most traditional and popular storage model and performs well in almost all application scenarios. However, it only guarantees the syntax and structure of the stored data regardless of semantic meaning, and has little effect on data exchange and sharing among the Web users (Elmasri & Navathe, 2003). Moreover, storing RDF data into relational databases, namely extending relational databases to support semantic data poses serious limitations on query and raises all kinds of issues, e.g. results in self- joins, and the openness of RDF data mode brings challenges to define a fixed relational schema (Heese & Znamirowski, 2011). To preserve the re-usability of published relational data and maximize the profit of utilizing Semantic Web, there have been a number of efforts transforming data from RDB schema to RDF schema (RDB-to-RDF) (Stefanova & Risch, 2013), such as direct mapping or domain semantic-driven mapping. The processes can be broadly classified into four categories:

- **Ontology matching:** Concepts and relations are extracted from relational schema or data by using data mining, and then mapped to a temporal established ontology or specific database schema (Shvaiko & Euzénat, 2013).

- Direct mapping: In this W3C recommended approach, tables of RDB are mapped to classes defined by an RDFS vocabulary, and attributes are mapped to RDF properties (Kyzirakos et al., 2018) (Arenas et al., 2012).
- Mapping Language: This involves cases of low similarity between database and target RDF graph, as exemplified by R2RML (Das, Sundara, & Cyganiak, 2012), which enables users express the desired transformation by following chosen structure or vocabulary.
- Search Engine-based: Transformation process is based on the SPARQL query of search engines with capability in supporting large collection of concurrent queries (Roshdy, Fadel, & EIYamany, 2013)

2.2.2 RDF storage

There are various RDF storage solutions such as triple store, vertically partitioned store, row store or column store. For efficient query processing in semantic-oriented environments, plenty of RDF stores which interpret the predicate as a connection between subject and object have been developed—for example Apache Jena—TDB, Amazon Neptune[®], or AllegroGraph[®]. RDF stores can be considered as a subclass of graph DBMS while they present specific approaches exceeded that of general graph DBMS such as query language SPARQL[®]. Here gives the most prominent instances:

MarkLogic[®] is a multi-model and massively scalable database being developed for storing and querying data in documents, graph data, or relational data, with a similar set of NoSQL features. Compared to other databases, MarkLogic has a built-in search engine, which makes it easier to build and configure indexes for standard queries. It has several advantages such as agile data integration, low cost and high data security, and therefore, widely used in managing regulated data, curating data for advanced analytic and advanced search and discovery.

GraphDB[®], a semantic repository with efficient reasoning, cluster and external index synchronization support, allows linking of text and data in large knowledge graphs (Bishop et al., 2011). GraphDB provides functions like inserting and transforming any type of data into RDF format, data reconciliation, ontology visualization and high-performance cluster, as well as supports integration to MongoDB. As a result, GraphDB shows competitive advantages for large-scale metadata management, semantic similarity search.

[®] <https://aws.amazon.com/cn/neptune/>
[®] <https://allegrograph.com/>
[®] <https://db-engines.com/en/article/RDF+Stores>
[®] <https://www.marklogic.com/>
[®] <https://ontotext.com/products/graphdb/>

Virtuoso® is an open source hybrid-RDBMS (Relational Database Management Systems) that can be used to manage data represented as relational tables and/or RDF sentence collections. As a sophisticated engine (blending the ACID and scalability) is developed, Virtuoso provides data wranglers with the semantically-informed navigation capacities of SPARQL. The data wranglers may execute custom reasoning and inferring. It is quite popular in many typical application scenarios such as safe & secure fusion of AI and data access, integration, and management due to its features of high-performance, security, and open standards-based.

Blazegraph® is a high-performance and scalable graph database supporting SW (RDF/SPARQL) and Graph Database APIs with high availability. The Java-written platform's principal trait is that it offers access to SPARQL 1.1 family of specifications like query, update and service description.

It's important to note that RDF stores and associated query languages such as SPARQL Query (Harris & Seaborne, 2013) and SPARQL Update (Gearon, Passant, & Polleres, 2013) offer the means to query and update large amounts of RDF data. SPARQL is the standard query language for RDF data.

With RDF data continuing to evolve in technology and scale, maintaining RDF data process is getting more complicated. Instead of requiring data wranglers to select different tools and configure them with a custom script, an integrated solution that provides standard maintenance and easy-to-use interfaces for multi-scenario applications is needed to ensure data processing tasks are executed regularly and efficiently.

3 Design and implementation

3.1 RDFAdaptor framework

The RDFAdaptor is conceived as a set of Kettle's ETL plugins focused on RDF data processing, which allows users to transform, merge, transfer, update, share linked data, and also aims to take maximum advantage of the Kettle's ETL ecosystem. Figure 1 illustrates the overall framework of the RDFAdaptor plugin.

The RDFAdaptor data access connector is built on top of Kettle and RDF stores, with Kettle's built-in interface to multiple non-RDF data sources and SPARQL 1.1 Query Language supported by most available RDF databases serving as a common communication protocol for indexing of RDF data. The RDFAdaptor set reuses an

© <https://db-engines.com/en/system/Virtuoso>

© <https://www.blazegraph.com/>

Research Paper

open-source Java framework RDF4J® (former Sesame API) as middleware to realize access to data repositories with SPARQL 1.1 support (Broekstra, Kampman, & Harmelen, 2002), thereby indicates the dataset level and infrastructure aspect. RDF4J offers an easy-to-use and vendor-neutral access API to all leading RDF database solutions (a.k.a. triple stores), and allows to connect with remote SPARQL endpoints and create applications that leverage the power of linked data and Semantic Web. On basis of the Kettle ETL ecosystem, the plugin set is implemented by Java programming backend.

There are four types of specific plugins, which are determined by the combination with RDF4J and SPARQL 1.1 protocol, as well as their intended purposes:

- *RDFZier*: A plugin that transforms input data to RDF data. Input data to this plugin is obtained from variety of external non-RDF data sources (several physical repositories) such as structured data from relational database by querying the chosen field information with SQL language, or csv/excel file of local system.
- *RDFTranslatorAndLoader*: A plugin works as a translator for bidirectional converting between various RDF data formats available on the Web with extended function of splitting/merging files, and load converted results to RDF stores. It supports input RDF data from file system, remote URL (SPARQL endpoints) or string stream. For instance, *RDFTranslatorAndLoader* can change RDF/XML format to the mainstream Turtle or JSON-LD with setting file split/merge size, maybe one splits into several and vice versa.
- *SparqlIn*: A plugin that extracts RDF data from SPARQL query endpoints. It has two types of configuration templates: one is SPARQL Select Query, output data can be RDF format (xml or json) and non-RDF (csv or tsv), the other is SPARQL Construct Query which keeps the output in RDF.
- *SparqlUpdate*: A plugin that is capable of executing SPARQL Update on a graph store, including two categories of Graph Update and Graph Management. It inherits all the facilities of SPARQL 1.1 Update protocol and offers a variety of update services within the graph store, e.g. inserting, deleting, and delete-inserting of triples from some graphs, creating, copying and deletion an RDF graph, etc.

Taking into account user-friendliness and efficiency, RDFAdaptor was developed with various of one-stop configuration templates created in graphical user interface for all plugins, thus kept clean and straightforward, only SparqlUpdate needs users

to do a little programming in JavaScript. More details about the usage follow in Section 4.

Given the time consuming of data collection and complexity of managing, to address these challenges we implemented a hybrid data storage mode which supports both local/remote file systems and RDF stores in a pattern of selection operation. The middleware RDF4J enables the access directly to Virtuoso, GraphDB and MarkLogic, while Blazegraph is accessed by RESTful API.

Figure 1. RDFAdaptor framework.

3.2 RDFAdaptor implementation

The RDFAdaptor inherits plenty of functions of Pentaho Data Integration, having most features required to process RDF data in a high intuitive way. From a user standpoint, RDFAdaptor is free of installation and easy to use. It has been successfully deployed and used in WINDOWS environment with Java Development Kit (JDK) installed, and provides a redeveloped user interactive interface (Spoon) to add and start tasks. Figure 3 illustrates an excerpt of the frontend interface screen where a data process task is designed. A task is composed of three main parts:

- *Task name*: It briefly defines the operation purpose when preparing and creating a data process task, and will further help for task management in practice. e.g. sparql-input-test, as illustrated in Figure 2(a).
- *Steps*: The main components (a set of functional objects) deployed to make a data process task, contain inputs, desired plugins or outputs. To make the operation easier and more friendly, configuration templates are defined as common field forms by using direct mapping, which have been customized for plugin setting requirements, and can be edited directly by users even if non-specialists (see Figure 2(b)).

Research Paper

- *Hops*: They are responsible for linking the chosen objects (nodes) together and indicating which direction the data flows go. Once the hops are defined, tasks are validated and ready to run.

It further can be seen that the user interface additionally includes modules such as *Database connection* (only for database connection), *Partition schemas*, *slave server*, *Kettle cluster schemas*, *Data Services*, *Hadoop cluster*, and *Execution Results*.

(a) A data processing task

(b) A configuration template

Figure 2. Front-end interface screenshot.

4 RDFAdaptor application

The RDFAdaptor enables a range of use scenarios, including: (1) transformation from non-RDF data to linked data (RDF), (2) RDF data format conversion, (3) linked data migration between triple stores, (4) RDF data update and management.

4.1 RDF data generation

The RDFAdaptor contains a wide variety of data access, and makes possible complex and reliable data transformations into linked data, i.e. the shift from internal or open data in relational databases or files containing structured data such as Excel, CSV, XML, and JSON files to RDF data.

For RDF data generation, RDFAdaptor provides the following features:

- Support multiple mainstream non-RDF format inputs, inheriting from Kettle's strong capability
- Visualized, dynamic and advanced RDF schema mapping
- One-stop execution that saves the effort for programming or script implementing
- Repeatable data transformation, one configuration and run everywhere
- Efficient paralleling process that can provide multithreaded operations

As depicted in subsection 3.2, a task of RDF data generation can be deployed by adding corresponding steps (configuration templates). To express how the task works, an application scenario instance of RDB2RDF is given. Using links to external relational database (table) as input with filed information by SQL query, we use the plugin *RDFZier* together to set up a demo task. The main principle of the plugin-based RDF generation is to extract entities and relations based on a giving data model or ontology. Figure 3 shows the overview of the workflow of RDF data generation with *RDFZier*, where the process takes two key sets of parameter settings supported by *RDFZier* algorithm. For each newly created task, parameter setting will be implemented in the configuration templates (Figure 2(b)). Table 1 shows the details of various parameters.

Figure 3. Workflow of RDF data generation with *RDFZier*.

Table 1. Parameters defined in *RDFizer*.

	Parameter	Description
Namespace	Prefix	collections of names identified by URI references
	Namespace	different prefixes depending on the required namespaces
Mapping Setting	Subject URI	HTTPURI template for the Subject/Resource, a placeholder {sid} would be used and replaced by UniqueKey
	Class Types	the classes to which the resource belongs, supporting multi-class types(split by semicolon), such as skos:Concepts; foaf:Person
	UniqueKey	the unique and stable primary key of resource, part of the Subject URI
	Fields Mapping Parameters	a list of field map from selected data source to target RDF schema, including the input Stream Field, Predicates, Object URIs, Multi-Values Separator, Data Type, Lang Tag
Dataset Metadata	Meta Subject URI	URI pattern of generated dataset
	Meta Class Types	the classes to which the resource belongs
	Parameters	a list of descriptions of generated dataset, including PropertyType, Predicates, Object Values, DataType, Lang Tag
Output Setting	File system setting	option for file system storage, including Filename and RDF format
	RDF store setting	option for RDF store, including triple store name, server URL, Repository ID, Username (if any), Password, Graph URI

The template contains four main components:

- *Namespaces*: the general part of RDF vocabulary which consists of syntax name, class name, property name, and resource name. For instance, the RDF namespace URI reference (or namespace name) is <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
- *Mapping Setting*: the most important and useful setting for the plugin, mapping rules from input data to target RDF data in which users should define the resource description URI pattern by following Cool URIs for the Semantic Web, such as <http://linked.aginfra.cn/{sid}>. If the column of Object URIs have values, the coming stream field would be deal with as ObjectProperty triple, and URI supporting the placeholder {oid}, otherwise, it would be deal with DataProperty and with advanced datatypes and lang-tag settings. If the stream field such one article's multi-keywords separated by semicolon, the plugin could mapping to multi-dc:subject by setting the Multi-Values Separator as “;”.
- *Dataset Metadata*: the configuration is used to describe the feature of target RDF data set, some of the elements of Vocabulary of Interlinked Datasets (VoID)[®] were recommended to use.
- *Output Setting*: options for the multi-hybrid data storage mode, including RDF local and remote files, several kinds of triple store database, and streams output for the Kettle's next step, which could be realized concurrently at the same time. Input rows could be split for batch processing as well.

4.2 RDF translation and loading

The constant expansion of RDF data with wealthy syntaxes brings about the problems in limitations on interoperability between tools/systems and disposal with various syntactical variants. Due to the different features of RDF data formats, data wranglers or Semantic Web developers chose them for personal taste, practical scenario, or intended purpose. To give an example, the most prominent syntaxes for RDF nowadays are RDF/XML, Turtles, and JSON. While RDF/XML preferably is used in information exchanges or large scale datasets, Turtle represents abbreviated syntax more suitable for human readability, JSON (or JSON-LD) is designed for embedding RDF in HTML markup and cutting down on the RDF consumption by Web applications.

Therefore, RDF data translation or conversion between different serialization formats is quite valuable. The task can be developed with the plugin *RDFTranslator-AndLoader*. Figure 4 illustrates the screenshot interface of the service in which can be seen that the configuration template consists of Input setting, Advance setting and Output setting. The data formats currently supported by *RDFTranslatorAndLoader* are: RDF/XML, Turtle, N-Triples, NQuads, JSON-LD, RDF/JSON, TriG, TriX, RDF Binary.

Data source allows local file system, remote URL (SPARQL endpoints) or string stream generated by the previous step in the task. The input format is determined by means of artificial selection, under the premise that correct data source is supplied, or the automatic detection of the document format will fail with an error log. Table 2 depicts the parameter details defined in *RDFTranslatorAndLoader*. It can be seen in the Advance setting that kinds of verification selectors (error detections) for URI syntax, language and datatype are inherited and reused from RDF4J, learning from the user interface of GraphDB's import functions, thereby improving the data conversion accuracy.

Another important feature of the plugin is supporting load the coming triples at the same time, to RDF files, triple stores, and also the streams to Kettle's next step. All the RDF formats, commercial (MarkLogic, GraphDB) and open source (Virtuoso, Blazegraph) triple stores are supported. It also supports loading the same triples to different file locations and different kinds of triple stores at one time.

4.3 RDF data migration

The maintenance of linked data among different RDF repositories is of increasingly importance in the Semantic Web, especially for RDF data migration which can realize the feasibility of managing large-scale and distributed RDF datasets. Given that plenty of knowledge bases are open on the Web, the linked data migration using

(a) Input Setting

(b) Output Setting

Figure 4. Configuration template of *RDFTranslatorAndLoader*.

Table 2. Parameters defined in *RDFTranslatorAndLoader*.

	Parameter	Description
Input	Source	RDF triples to be converted or loaded
	Source Type	data source, such as local file system, Remote URL or string stream
	Source RDF Format	format of the input RDF data, fully supporting the common RDF formats
Advance	Large Input Triples	a selector for input data scale large or not, if the input is large, then the output step can not count, merge or split the triples
	BaseIRI	resolve against a Base IRI if RDF data contains relative IRIs
	BNode	a selector for preserving BNode IDs
	Verify URI syntax	a selector for URI syntax/relative URIs/language tags/datatypes check
	Verify relative URIs	which returns fail log when corresponding errors occur
	Verify language tags	
	Verify datatypes	
	Language tags	a selector for language tags / datatype, including fail parsing if languages / datatypes are not recognised and normalizing recognised language tags / datatypes values
Output	Datatype	
	Target RDF Format	RDF format of the converted output
	Commit or Split Size	number of RDF triples for the output to each RDF files or submit to stores every batch, the default value is 0, which means all the input data would be processed at one time
	Local File Setting	options of file system storage, including three selectors for “Save to File System”, “Keep Source FileName” and “Merge to Single File (take precedence over “Commit or Split Size”)”, File name and location
	TripleStore Setting	options of RDF store, including a selector for “Save to Store”, Triple Store, Server URL, Database/RepositoryID/NameSpace (identifier of database for different triple store), UserName, Password, and Graph URI.
	Stream setting	option of String Stream for further data transferring, including a selector for “Save to Stream”, and Result Field

SPARQL queries is one of the important means for massive RDF data organization and management. The plugin *SparqlIn* takes advantage of the functionalities of SPARQL 1.1 Query Language and can be applied to RDF data migration against remote URL (SPARQL endpoint of the RDF stores) in a read-only mode based on graph cache, namely transferring remote RDF datasets locally or triple stores. Table 3 gives detail descriptions of the configuration parameters in a task of remote RDF data migration, and Figure 5(a) shows the demo screenshot. The task allows both SPARQL SELECT and CONSTRUCT query forms, the former returns variables and their bindings directly which can be accessed by a local API or serialized into either JSON, XML, CSV, or TSV, while the latter returns a RDF graph specified by a graph template only.

It should be noted that the user needs to add a text file-like output step, in which more output parameters (file name, target storage path, parent fold, compression or not, et al.) can be set.

Table 3. Parameters defined in *SparqlIn*.

	Parameter	Description
SPARQL Setting	Accept URL from field	checkbox, if checked means the Url of the SPARQL Endpoint would be coming from Kettle’s previous steps and the value could get from the “URL field name”
	URL field name	only used by giving a list of drop-down options of input fields when the option “Accept URL from field” is selected
	SPARQL Endpoint URL	endpoint Url queried when “Query Endpoint Url From Field” is disabled
	Query Type	query type which provides two options: Graph query or Tuple query
	SPARQL Query	SPARQL query forms: SELECT or CONSTRUCT
	Limit	limitation on data size to be processed if necessary
Output Setting	Offset	the starting position of data processing
	Result Field Name	field specified for file saving
	RDF Format	target local data format, either JSON, XML, CSV or TSV for SELECT query, RDF format only for CONSTRUCT query
	Max Rows	definition of the maximum size of the output file, empty of 0 means get all the triples
Http Auth	HTTP UserID	user ID of SPARQL endpoint if any
	HTTP Password	password of SPARQL endpoint if UserID exists

It should be noted that the parameter “Max Rows” enables users to obtain all the data of triple store in batches when it is the default value of 0.

4.4 RDF graph update

New data usually arrives regularly. A executable and repeatable data transformation is needed for a scalable and lower-cost data process. As mentioned previously in Subsection 2.2, RDF store is also a kind of graph store which works as a mutable container of RDF graphs. Similar to RDF datasets operated on by the SPARQL 1.1 Query Language, RDF graphs can be updated by using SPARQL 1.1 Update as a graph update language. Based on the plugin *SparqlUpdate*, we show how to implement two representative operations of graph update and graph management using update queries against the SPARQL endpoint of the graph store.

Table 4 describes the parameters of configuration setting in an RDF data update task in detail, and Figure 5(b) shows the demo screenshot. Unlike the other three plugins where the user only needs to fill out the configuration templates, *SparqlUpdate* is designed with configuration file and functional modules separated and the latter requires some programming in JavaScript. Currently, the RDFAdaptor has included most functions of Graph Update and Graph Management on SPARQL 1.1 Update Language, which are redeveloped to seven main functional modules (steps) in *SparqlUpdate*.

Step name: SPARQL Input-Construct

General

SPARQL Settings

SPARQL Endpoint URL:

Accept URL from field?: ☐

URL field name:

Query Type:

SPARQL Query:

```
CONSTRUCT {?s ?p ?o}
WHERE {
  GRAPH <http://scigraph.springernature.com>
  {?s ?p ?o. }}
```

Limit:

Offset:

Output Settings

Result Field Name:

RDF Format:

Max Rows:

Http Auth

HTTP UserID:

HTTP Password:

(a) *SPARQLIn* Setting

Step name: SPARQL Update

General

SPARQL Settings

Query Endpoint Url From Field?: ☐

Query Endpoint Url Field:

Query Endpoint Url:

Update Endpoint Url From Field?: ☐

Update Endpoint Url Field:

Update Endpoint Url:

Query From Field?: ☐

Query Field Name:

Base URI:

SPARQL Update Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA { GRAPH <http://rdf.nstl.gov.cn/article>
{
  <http://rdf.nstl.gov.cn/article/00000329-579A-48B7-B9D5-33372812EFFF>
  dc:title "Dehydration of paddy rice in a chitosan membrane drier"}}}
```

Output Settings

Result Field Name:

Http Auth

HTTP UserID:

HTTP Password:

(b) *SPARQLUpdate* SettingFigure 5. Configuration template of *SPARQLIn* and *SPARQLUpdate*.

Research Paper

Table 4. Parameters defined in *SparqlUpdate*.

	Parameter	Description
SPARQL Setting	Query Endpoint Url From Field?	checkbox, if checked means the Url of the SPARQL Query Endpoint would be coming from Kettle’s previous steps and the value could get from the “Query Endpoint Url Field”
	Query Endpoint Url Field	only used by giving a list of drop-down options of input fields when the option “Query Endpoint Url From Field” is selected
	Query Endpoint Url	The value of the Query Endpoint Url would be used when “Query Endpoint Url From Field” is unchecked
	Update Endpoint Url From Field?	checkbox, if checked means the Url of the SPARQL Update Endpoint would be coming from Kettle’s previous steps and the value could get from the “Update Endpoint Url Field”
	Update Endpoint Url Field	only used by giving a list of drop-down options of input fields when the option “Update Endpoint Url From Field” is selected
	Update Endpoint Url	The value of the Update Endpoint Url would be used when “Update Endpoint Url From Field” is unchecked
	Query From Field?	checkbox, if checked means the SPARQL Update Query would be coming from Kettle’s previous steps and the value could get the “Query Field Name”
	Query Field Name	only used when the option “Query From Field” is selected
Output Setting	Base URI	resolve against a Base IRI if RDF data contains relative IRIs
	SPARQL Update Query	JavaScript programming for graph update which is only used when the option “Query From Field” is disable
Http Auth	Result Field Name	field specified for file saving
	HTTP UserID	user ID of SPARQL endpoint if any
	HTTP Password	password of SPARQL endpoint if UserID exists

Graph Update:

- *Insert Data*: add some triples to a destination RDF graph
- *Delete Data*: remove some triples from an RDF graph containing triples
- *Delete/Insert*: delete groups of triples and insert some new simultaneously

Graph Management:

- *Drop Graph*: drop an RDF graph from a graph store
- *Copy Graph*: modify a graph to contain a copy of another one
- *Move Graph*: move the contents of one RDF graph into another
- *Add Graph*: reproduce all of the data from one graph into another

Here shows an example snippet to update the graph *http://rdf.nstl.gov.cn/article* to insert triples with fields of gid (Unique Key), title, and publication date.

```
var graphURI="{GRAPH<http://rdf.nstl.gov.cn/article>}";
var prefix="PREFIX dc:<http://purl.org/dc/elements/1.1/>";
var triples="{<http://rdf.nstl.gov.cn/article/"+gid+">dc:title \"\" + title + \"\" ; dc:publicationDate \"\"+ year + \"\"}";
var update=preFix+"INSERT DATA"+graphURI+triples+"}";
```

Also, the modules in those two types of update operations can be used in a cascading combination to achieve more complex applications.

5 Cases and evaluation

Based on the extensive application scenario design, the RDFAdaptor plugins have been deployed and tested successfully in some practical use cases involving linked data processing. In this section, we describe some experiments or projects in open linked data and data organization in library and information field.

The National Science and Technology Library (NSTL) in China is an information service agency of science and technology literature which provides public services of full-text access or knowledge discovery, et al. Before NSTL releases its core datasets—data about journals, journal articles, books, conference papers, dissertations, scientific reports, patent standards, metrology regulations, and specific resource, lots of preparatory work about data processing are involved. With the Semantic Web opening up new opportunities for data mining and increasing content discoverability, NSTL moved to adopt linked data—the rich representation of data model, and worked to build large knowledge repositories about real world entities and concepts to support advanced knowledge service forms such as semantic retrieval. Traditionally NSTL mainly uses relational databases to store structured data collected and processed from multiple source, so the goal of NSTL is to get through all the links from data collection to relational data release in a low-cost and efficient way.

Currently, NSTL has successfully integrated the RDFAdaptor into the whole data process including:

- ETL of multi-source heterogeneous data
- Semantization of the collected data
- Integration of various types of data
- SPARQL query for internal data manager over graph repositories

With the graph repositories, NSTL is taking further steps in other usage scenarios: e.g. building RDF data into knowledge graphs to serve information needs precisely, or using the links to external repositories to contribute the web of linked science data.

To evaluate the effectiveness of the plugins, a set of experiments on RDF data generation/translation and loading (to triple store) are performed, Table 5 gives the final results.

Research Paper

Table 5. RDF data generation/translation and loading.

Data Source	Data Format	Number of Records	Number of mapped fields	Number of RDF generated	Total Time-consuming
MongDB	json	1,948,268	17	37,038,563	32min18s
SqlServer	RDB	336,831	5	1,159,687	38.6s
		798,389	9	7,521,876	5min4s

The above table shows that the plugins can support the generation/translation and loading of RDF data at a good performance, about 10,000 triples per second in our case, mainly depends on the reading speed of data sources, such that SqlServer is significantly more efficient than MongDB for our testing cases. Moreover, the complexity of data is inversely correlated with efficiency, the more fields there are, the slower the RDF is generated/translated and loaded.

As for another common case, data wranglers may want to directly get RDF triples from some remote SPARQL query points, so we try to dump all the triples from the AGROVOC’s SPARQL query endpoint to local turtle files via a transformation of Kettle using the *SparqlIn* plugin, setting the query type as GraphQuery and getting 500,000 triples for each request. The total time taken for 6,752,536 RDF triples is 2min30s, as shown in Figure 6(a). To evaluate the effectiveness of the plugin, we perform the same test on the typical linked data process framework—LDIF—in SPARQL import mode, which takes 6h47min41s (Figure 6(b)), partly because its scale of data acquisition is limited to 5000 each time. The experiment shows that our plugin outperforms IDLF in both the intuitiveness of task and data processing efficiency.

The experiments and experiences of using the plugins in these use cases confirmed that RDFAdaptor can be used to complement other services or tools with missing functionalities in an efficient way. Using RDFAdaptor simplified the tasks of RDF data processing and at the same time provided us with valuable hints about the importance to adapt to the needs of existing data infrastructure. However, it is apparent that support for some of these functionality dimensions still has limitations and requires further improvement, e.g. manual configuration and editing—a costly process, in plugin *SparqlUpdate*, the scale of a high-volume data process, et al.

6 Conclusions and outlook

In this paper we have presented the RDFAdaptor which comes with four developer-friendly and out-of the box plugins for RDF data processing built on top of the prominent ETL tool—Pentaho Data Integration and RDF java framework RDF4J. The RDFAdaptor is implemented by Java programming backend in a semantic-based way and able to manage data of different types and formats from heterogeneous

(a) Dump AGROVOC RDF Triples via a Transformation with Plugin *SparqlIn*

(b) Dump AGROVOC RDF Triples with LDIF

Figure 6. Dump All AGROVOC RDF Triples from SPARQL Endpoint to Local Files.

sources or multiple repositories even SPARQL endpoint (remote URL). It is intended as a comprehensive solution to address the problem of efficient multiple-source data processing which covers the whole life-cycle from extraction, transformation, integration, and storage, and embodies the following features:

- Support for multiple types of data sources, including non-RDF data (e.g. Excel, CSV, XML and JSON files or relational databases) and RDF data.
- A hybrid data storage mode which supports both local file systems and prominent RDF stores in a pattern of selection operation.
- A friendly and straightforward user interface, which sketches key components or modules such as steps, hops.
- Offer a simple and efficient way to create and start complex data process tasks by providing various configuration templates which can save artificial and cost.

Research Paper

- Multi-scenario applications, including RDF data generation, RDF data translation or conversion between different serialization formats, remote RDF data migration, or RDF graph update.
- Flexibility to complement other services or tools.

In addition to the features presented so far, the exemplary use cases / experiments introduced in Section 5 also reveals certain limitations of RDFAdaptor. As future work we consider to further improve the plugin functions in various aspects. This includes: existing error detection/check (e.g. URI syntax, language tags, datatypes), encapsulation of the plugin *SparqlUpdate*, parameter instructions. Furthermore, with the aim to support interaction with other graph repositories and utilize the value of existing data better, we will try to develop some new plugin functions such as ontology-based data access and resolution, RDF data clustering, which would improve interoperability with the Semantic Web and at the same time they would be extendable to many other areas and allow more practical usage scenarios.

Acknowledgements

This work is supported by “National Social Science Foundation in China” Project (19BTQ061) and “Integration and Development on A Next Generation of Open Knowledge Services System and Key Technologies” project (2020XM05).

Author contributions

Jiao Li (lijiao@caas.cn): wrote the manuscript and design the research framework. Guojian Xian (xianguojian@caas.cn): developed the plugin set. Ruixue Zhao (zhaoruixue@caas.cn): revised the manuscript. Yongwen Huang (huangyongwen@caas.cn): investigated and tested ETL tools. Yuantao Kou (kouyuantao@caas.cn): designed a verification experiment, Tingting Luo (luotingting@caas.cn): prepared data. Tan Sun (suntan@caas.cn): proposed the research problems.

References

- Arenas, M., Bertails, A., Prud'hommeaux, E., & Sequeda, J. (2012). A direct mapping of relational data to RDF. Retrieved from <https://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/>
- Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Ruslan, V.R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1), 33–42. doi: 10.3233/SW-2011-0026
- Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data—the story so far. *International Journal on Semantic Web and Information Systems*, 5(3), 1–22. doi: 10.4018/jswis.2009081901
- Broekstra, J., Kampman, A., & Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Proceedings of the Semantic Web-ISWC 2002*, 54–68. Berlin, Heidelberg: Springer. doi: 10.1007/3-540-48005-6_7

- Cygniak, R., Wood, D., & Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation. Retrieved from <https://www.w3.org/TR/rdf11-concepts/>
- Das, S., Sundara, S., & Cygniak, R. (2012). R2RML: RDB to RDF Mapping Language. W3C Recommendation. Retrieved from <http://www.w3.org/TR/r2rml/>
- Elmasri, R., & Navathe, S.B. (2003). *Fundamentals of Database Systems*. Addison Wesley.
- Erling, O., & Mikhailov, I. (2009). RDF Support in the Virtuoso DBMS. In: Pellegrini T., Auer S., Tochtermann K., Schaffert S. (eds) *Networked Knowledge—Networked Media*. Studies in Computational Intelligence, vol 221. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02184-8_2
- Gearon, P., Passant, A., & Polleres, A. (2013). SPARQL 1.1 update. W3C Recommendation. Retrieved from <https://www.w3.org/TR/sparql11-update/>
- Harris, S., & Seaborne, A. (2013). SPARQL 1.1 query language. W3C Recommendation. Retrieved from <http://www.w3.org/TR/sparql11-query/>
- Heese, R., & Znamirski, M. (2011). Resource centered RDF data management. In *Proceedings of International Workshop on Scalable Semantic Web Knowledge Base Systems*, pp. 138–153.
- Inmon, B. (1997). The Data Warehouse Budget. *DM Review Magazine*.
- Klímek, J., Škoda, P., & Nečaský, M. (2016). LinkedPipes ETL: Evolved Linked Data Preparation. In *Proceedings of International Semantic Web Conference*, 95–100. Springer International Publishing.
- Knap, T., Škoda, P., Klímek, J., & Nečaský, M. (2015). UnifiedViews: Towards ETL Tool for Simple yet powerfull RDF Data Management. In *Proceedings of the Dateso 2015 Annual International Workshop on Databases*, 111–120.
- Knap, T., Hanečák, P., Klímek, J., Mader, C., Nečaský, M., Nuffelen, V.B., & Škoda, P. (2018). UnifiedViews: An ETL tool for RDF data management. *Semantic Web*, 9(3), 1–16. doi: 10.3233/SW-180291
- Kyzirakos, K., Savva, D., Vlachopoulos, I., Vasileiou, A., Karalis, N., Koubrarakis, M., Manegold, S. (2018). Geotriples: transforming geospatial data into rdf graphs using r2rml and rml mappings. *Journal of Web Semantics*, 16–32. doi: /10.1016/j.websem.2018.08.003
- Shvaiko, P., & Euzenat, J. (2013). Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1), 158–176. doi: 10.1109/TKDE.2011.253
- Roshdy, H.M., Fadel, M.K., & ElYamany, F.H. (2013). Developing a RDB-RDF Management Framework for Interoperable Web Environments. In *Proceeding of the 4th IEEE Eurocon Conference*, 307–313. Zagreb: Croatia.
- Schultz, A., Matteini, A., Isele, R., Bizer, C., & Becker, C. (2011). LDIF—Linked data integration framework. In *Proceedings of the Second Inter-national Workshop on Consuming Linked Data (COLD2011)*, 1–6.
- Stefanova, S., & Risch, T. (2013). Scalable reconstruction of RDF-archived relational databases. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, 1–4.
- Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). Silk—A Link Discovery Framework for the Web of Data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web*.

This is an open access article licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Journal of Data and
Information Science

<http://www.jdis.org>

<https://www.degruyter.com/view/j/jdis>