ClioPatria: A Logical Programming Infrastructure for the Semantic Web

Jan Wielemaker [a] Wouter Beek [a] Michiel Hildebrand [b] Jacco van Ossenbruggen [b]
[a] *VU University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands,
e-mail: {J.Wielemaker,W.G.J.Beek}@vu.nl*
[b] *CWI, Science Park 123, 1098 XG Amsterdam, The Netherlands,
e-mail: {M.Hildebrand,jacco.van.ossenbruggen}@cwi.nl*

Abstract. ClioPatria is a comprehensive semantic web development framework based on SWI-Prolog. SWI-Prolog provides an efficient C-based main-memory RDF store that is designed to cooperate naturally and efficiently with Prolog, realizing a flexible RDF-based environment for rule based programming. ClioPatria extends this core with a SPARQL and LOD server, an extensible web frontend to manage the server, browse the data, query the data using SPARQL and Prolog and a Git-based plugin manager. The ability to query RDF using Prolog provides query composition and smooth integration with application logic. ClioPatria is primarily positioned as a prototyping platform for exploring novel ways of reasoning with RDF data. It has been used in several research projects in order to perform tasks such as data integration and enrichment and semantic search.

Keywords: Triple Store, Logic Programming, Semantic Web Framework

1. Introduction

Many Semantic Web applications converge to a three-tier architecture[1] —storage, application logic, presentation— that is popular in the relational database world, were SQL is replaced by SPARQL. SPARQL servers provide a varying and sometimes configurable level of entailment reasoning to support applications in data integration (e.g., `owl:sameAs`, `rdfs:subPropertyOf`) and hierarchical reasoning (e.g., `rdfs:subPropertyOf` and `rdfs:subClassOf`). The application logic is generally expressed in an (object-oriented) general purpose programming language.

Although we acknowledge there is a need for such architectures, because of their proven robustness, scalability and familiarity for the software industry, we claim that these architectures are not suitable to explore the full potential of the ideas behind the Semantic Web for two reasons. Firstly, the separation between storage (SPARQL) and application logic (general purpose language) often causes a single computation to be expressed in a mixture of these two languages. This mixture harms readability as well as performance. Secondly, most languages used for the logic tier do not support higher level rule-based reasoning easily and suffer from the object-relational impedance mismatch [13], which makes expressing the application logic cumbersome.

We have developed ClioPatria to explore applications of the Semantic Web that are not well supported by the three-tier architecture. Its first application [22, 37] became the winner of the ISWC-2006 Semantic Web challenge. This application provides search over multiple museum collections while clustering the results based on their relation to the search term. The core of the application is an RDF path finding heuristic and an algorithm for clustering paths based on the class and property hierarchies. Such techniques could not be expressed in SPARQL 1.0. Although SPARQL 1.1 primitives such as path-expressions allow for pushing a little more of the application logic to the query language, SPARQL-based solutions remain a mixture of

[1]`https://msdn.microsoft.com/en-us/library/ee658109.aspx`

SPARQL queries and client code that is hard to maintain and performs poorly due to the large number of SPARQL queries required to perform the search.

ClioPatria is a Prolog-based RDF framework that aims primarily at *prototyping*. It particularly facilitates prototyping due to the following features:

- A dedicated RDF store that is implemented in the C-language and designed to interface to Prolog naturally, providing a stable and efficient core storage and rule engine.
- Libraries that support all standard RDF interchange formats, the SPARQL 1.1 query language and LOD services, allowing to interface to other tools from the Linked Data community,
- The Prolog language provides a stable rule formalism and facilitates prototyping through live recompilation of the program under development, strong reflective and meta-programming support and the ability to 'retry' (go back to a previous checkpoint) goals in the debugger that did not produce the expected result and thus inspect the execution at a more detailed level. [38]
- An integrated web development framework allows for serving an entire RDF-based web application from a single executable.

We would like to position ClioPatria primarily as an RDF *library* that is tightly connected to a programming language (Prolog). This positions it relatively close to Java-based systems such as Sesame, Jena, the Python-based rdflib and the Common Lisp-based AllegroGraph (see Section 3). It is positioned at a much greater distance from (often) disk-based SPARQL engines such as 4store and Virtuoso.

This document is organised as follows: before getting to related work in Section 3 we make the reader familiar with ClioPatria's concepts, features and design in Section 2. Application areas are described in Section 4, where we show ClioPatria's impact. In Section 5 we discuss the current limitations and our plans for the future. Section 6 concludes.

2. Introducing ClioPatria

2.1. Prolog and RDF

The central feature of ClioPatria is to use Prolog as rule and 'glue' language. RDF triples are naturally expressed in Prolog by the relation `rdf`(*Subject, Predicate, Object*), where URIs are represented by Prolog atoms (interned strings) and literals are represented by a Prolog term that expresses the lexical form and the datatype or language tag. First, we give some examples of triples represented in Prolog taken from the RDF 1.1 primer document:[2]

```
rdf('http://example.org/bob#me',
    'http://...rdf-syntax-ns#type',
    'http://xmlns.com/foaf/0.1/Person').
rdf('http://example.org/bob#me',
    'http://xmlns.com/foaf/0.1/knows',
    'http://example.org/alice#me').
rdf('http://example.org/bob#me',
    'http://schema.org/birthDate',
    literal(type('http://...#date',
                 '1990-07-04'))).
```

Like SPARQL and Turtle, the Prolog RDF layer allows for prefix notation. Prefixes are declared using `rdf_register_prefix`(*Prefix, URI*), after which URIs can be written as *prefix*:*localname*. We will use this notation in the remainder of this article, using the conventional prefixes as defined on `http://prefix.cc/`.

Given an RDF database, Prolog allows us to write graph patterns as they appear in e.g., SPARQL as a conjunction of `rdf/3` relations. For example to get the birth dates of people somebody knows we can write:

```
known_with_birthdate(Me,He,Date):-
  rdf(Me, foaf:knows, He),
  rdf(He, schema:birthDate, Date).
```

The above code corresponds to the following SPARQL query:

```
SELECT ?me ?he ?date WHERE
{ ?me foaf:knows ?he .
  ?he schema:birthDate ?date
}
```

We claim, however, that the `known_with_birthdate/3` version in Prolog has many advantages over the SPARQL version, which we will explain below.

First, because the Prolog version has an explicit name, it is easy to reuse it in other queries. This allows developers to incrementally build more sophis-

[2]`http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/#section-n-triples`

ticated functionality on top of existing, proven and tested building blocks. This is an effective way to avoid the large and complex SPARQL queries often found in other Semantic Web applications.

A second advantage is that the three arguments to `known_with_birthdate/3` can be used both to query and to filter. For readers that are not familiar with Prolog: identifiers that start with a capital letter are *variables*, while identifiers that start with a lower case letter or appear within single quotes are *constants*, called *atoms* in Prolog.

The interactive session below illustrates the use of `known_with_birthdate/3` to query the triple store. Note that the predicate generates all solutions one by one (the semicolon (;) is typed by the user to ask for the next solution).

```
?- known_with_birthdate(Me, He, Date).
Me = 'http://example.org/bob#me'
He = 'http://example.org/alice#me'
Date = literal(type(xsd:date,
                    '1990-07-04') ;
Me = 'http://example.org/bob#me'
He = 'http://example.org/john#me'
Date = literal(type(xsd:date,
                    '1992-09-06') ;
...
```

Instead of passing variables, we can also pass concrete values as arguments to the predicate. This is logically equivalent to filtering all results to those where the argument has the specified value. Calling `known_with_birthdate`(*'http://.../bob#me', He, BirthDate*) results in the first call to `rdf/3` using the combined subject-predicate index instead of just the predicate index, producing the reduced solution set efficiently. Calling `known_with_birthdate/3` with only a given *BirthDate* also produces the correct results, but inefficiently. This is discussed below.

Combined with operations on literals and Prolog's aggregation primitives, the above apparatus allows any SPARQL query to be expressed naturally. And, it can do more. Firstly, the above `known_with_birthdate/3` rule can be used as an abstraction of the two `rdf/3` statements and can be embedded in more complex graph patterns. This ability to *compose* queries facilitates exploring RDF data by assembling a library of simple graph patterns that can be named, documented and shared. Secondly, external sources such as relational databases or arbitrary Prolog relations can naturally be mixed with the RDF query. Thirdly, more powerful control structures can deal with recursion, constraint satisfaction, etc.

Prolog's *SLD resolution*, basically depth-first traversal of the search space, solves the stated queries under the provision that the user takes measures to avoid unbounded recursion.[3] The performance of the SLD resolution strategy is very sensitive to proper ordering of the graph patterns. Prolog's *reflective* capabilities that allow Prolog to inspect and rewrite Prolog programs can solve this. The predicate `rdf_optimise`(*PaternIn, PatternOut*) rearranges a graph pattern based on statistics maintained by the core triple store [33].

2.1.1. Expressiveness

Due to various restrictions, standardized query languages for SW data have limited expressivity. In practice, this requires the combination of server-side query execution and client-side post-processing of the results. There are several ways in which ClioPatria provides a more expressive programming environment.

Literal search in SPARQL is limited to the use of regular expressions and (numerical) conditions. ClioPatria couples its full-text literal index (Section 2.2) to Prolog's strong background in Natural Language Processing (NLP) [36]. This allows more complicated operations such as stemming, phonetic matching, string distance calculation, and NLP parsing to be performed on literals. The alternative approach of using a general text indexing engine besides a triple store results in increased memory requirements and access times, and is difficult to synchronize under dynamic RDF triple assertions and retractions.

Even though SPARQL 1.1 is more powerful than version 1.0, there are still realistic queries that cannot be expressed in it [16]. For example, "which property paths connect a search string on literals to resources of type *Painting*". For a fixed length, this can be expressed as follows in SPARQL (searching for *query* at distance 2):

```
SELECT ?l ?p1 ?r1 ?p2 ?r WHERE
{ ?r  rdf:type art:Painting .
  ?r  ?p1 ?r2 .
  ?r2 ?p2 ?l .
  FILTER regex(l, "query")
}
```

[3]Some Prolog systems, such as XSB and YAP, can detect and avoid unbounded recursion automatically using a technique called *tabling* [24].

SPARQL does not provide the primitives to generalize this query for arbitrary path lengths, while this is easily expressed in Prolog:

```
literal_path_to_painting(Q,R,[L-P0|P]):-
  rdf(R0,P0,literal(substring(Q),L)),
  path_to_painting(R0,R,P,[]).

path_to_painting(R,R,[],_):-
  rdf(R, rdf:type, art:'Painting').
path_to_painting(R0,R,[R0-P|Path],V):-
  rdf(R0,P,R1),
  \+ memberchk(R1,V),
  path_to_painting(R1,R,Path,[R1|V]).
```

The above algorithm implements naive depth-first search. In the previously mentioned ISWC challenge application we implemented a best-first algorithm where the distance was based on predicate semantics (e.g., owl:sameAs does not change the distance) and predicate statistics (a predicate that links to more resources increments the distance more). It should be clear that SPARQL is not of much value for such problems.

2.1.2. *Reasoning*

Entailment reasoning can be implemented in several ways. ClioPatria provides a library that provides RDFS entailment primitives such as `rdfs_individual_of`(*?Resource, ?Class*). This is the same as {?resource rdf:type ?class} after computing all triples entailed by the RDFS semantics. In addition, ClioPatria provides *entailment modules*. An entailment module is a Prolog module that extends the relation `rdf/3` which initially expresses only the plain triples. For example, the transitive semantics of `rdfs:subClassOf` can be expressed in an entailment module with the rule below, exploiting the transitive closure built-in `rdf_reachable/3`:

```
rdf(S, rdfs:subClassOf, O):-
  rdf_reachable(S, rdfs:subClassOf, O).
```

Besides entailment models for standardized entailment regimes such as RDFS and OWL, it is just as easy to write an entailment module with custom, domain-specific rules. This provides similar functionality as *function properties* in Jena. According to the following example, persons younger than 18 are children:

```
rdf(Person, rdf:type, dbo:'Child'):-
    rdf(Person, dbo:age, Age),
    xsd_smaller(Age, 18).
```

Both forward and backward reasoning can be naturally expressed in Prolog. In many scenarios backward reasoning is to be prefered, because it reduces the memory footprint, avoids the problem of keeping inferred triples consistent after database modifications, and allows a single application to use different entailment definitions. Some deductions, however, are expensive to implement using backward reasoning. For example, finding all `rdf:type` triples that are entailed by RDFS domain and range restrictions requires a considerable amount of search. In such cases one can opt for forward chaining instead.

2.2. *The triple store*

Although triples could be expressed simply using a Prolog *dynamic predicate*, we have choosen for a low-level C implementation. This allows us to exploit the restricted form of this relation (no rule *body*, URIs as subject and predicate and only restricted ground terms as object) to reduce memory requirements and improve indexing. The underlying store is a memory-based quad-store that provides 9 hash-based indexes (s=subject, p=predicate, o=object, g=graph): s, p, o, sp, po, spo, g, sg and pg. These indexes are created and resized lazily. In addition, a closure of the `rdfs:subPropertyOf` relation is maintained to speed up queries that rely on subproperty entailment. An ordered index of all RDF literals is maintained to facilitate prefix or literal range queries. Optionally, the store can maintain a full-text index of all literals that provides fast search for literals that contain specified tokens, optionally extended with stemming and methaphone (‘sounds-as’).

The triple store has strong support for dynamic changes. Dynamic changes fit well with Prolog’s bias towards backward chained reasoning (Section 2.1.2). Modifications to RDF data follow the Prolog *logical update view*, which implies that the results of queries are not affected by changes to the triple store that are made during query execution [35]. The triple store is designed to cooperate with the multi-threaded Prolog core. Read operations are implemented using lock-free techniques. Write operations use (short held) locks. Multiple updates may be combined into atomic *transactions* that can be nested. Prolog *failure* or a Prolog exception causes the transaction to be discarded. The same technology supports *snapshots*, which allows a Prolog goal to make isolated and temporary changes to the RDF store (Section 4.7).

The triple store optionally provides reliable persistency based on a *journal* file and fast save/load based on an optimized binary representation for RDF data.

Table 1 gives some core metrics of the RDF store. The store has been used with up to 200 million triples. Additional information can be found in [34].

2.3. Semantic web standards

The primary asset of the RDF initiative is a common data model and languages to operate on this model. ClioPatria contains modules for reading/writing the most standard RDF serialization formats (i.e., N-Quads, N-Triples, RDFa, RDF/XML, TriG and Turtle). The parsers and generators have been stress-tested extensively within the LOD Laundromat project (Section 4.3).

ClioPatria supports SPARQL 1.1. SPARQL queries are compiled into Prolog control structures. The resulting structure is optimized (Section 2.1) before execution. All SPARQL functions are implemented in a SPARQL runtime support library. The current implementation applies almost all these functions after performing the (optimised) graph pattern query, e.g. ignoring opportunities to map regular expression prefix searches to its ordered literal index. Interactive exploration is supported by the YASGUI query editor [21].

Support for Linked Data support is available as a configurable option. Based on content negotiation, ClioPatria dereferences resources with machine-readable RDF or a human-readable HTML page. ClioPatria can serve URIs directly but also supports handling of URI redirection, e.g., through using `http://purl.org`. By default, the machine-readable dereference of a resource is the concise bounded graph of that resource. Alternatives can be selected or programmed as Prolog rules. It is possible to install custom HTML rendering of a resource, for example to define the page layout, supress properties, include the rendering of related resources or render images.

2.4. Plugins & community extensions

A CPACK (or ClioPatria package) extends ClioPatria by providing additional Prolog libraries and/or web services. Packs are available from the ClioPatria site[4] and can be installed from the Prolog shell. A CPACK consists of a directory hierarchy that provides Prolog libraries, configuration files and web resources such as CSS and JavaScript files. It is acompanied with RDF meta data that explains the type, purpose and authors of the package. In addition, the RDF description may specify dependencies. The CPACK is made available as a Git repository. If a CPACK is submitted, its Prolog code is analysed for dependencies on and conflicts with known CPACKs. Installing a CPACK installs all dependencies, adds the new directories to ClioPatria's file search paths and optionally loads them into the system based on the default configuration information provided. Due to the strong dependency on code analysis before accepting a package, packages typically coexist flawlessly without human intervention. The drawback is that a package must follow quite strict guidelines with respect to code organization.

2.5. Application development

Initially, ClioPatria applications are either developed as CPACKs (Section 2.4) or as independent Prolog source files that are loaded into ClioPatria. Recent versions provide a web-based alternative based on *Pengines*, or Prolog Engines [14].[5] The Pengine infrastructure allows for embedding a Prolog program in a web page using a `<script type="text/x-prolog">` element and pose queries to this program from within JavaScript. If a query is submitted, both the query and program from the `<script>` elements are sent to the server using an HTTP POST query. The server loads the code into a sandboxed temporary Prolog module and executes the query. The client receives the results as JSON objects. Similar to the Prolog toplevel, the client can ask for additional solutions until it is satisfied.

This architecture allows web applications to be written without loading server extensions into ClioPatria, while the user can still profit from the advantages described in Section 2.1 such as query composition, data integration and the expressivity of the full Prolog language.

RDF data may be explored and suitable query fragments may be assembled using the Pengines-based web application called SWISH. SWISH is an acronym that stands for "**SWI**-Prolog for **SH**aring" or "**SWI**-Prolog **SH**ell". It provides a web-based code editor with code highlighting and template-based completion, a query editor with the same features and a result area. Programs can be saved, tagged and searched

[4] `http://cliopatria.swi-prolog.org`

[5] Pengines are online at `http://pengines.swi-prolog.org`

Table 1

Core metrics of the SWI-Prolog RDF store. The values are based on the 64-bit version running Ubuntu 14.04 on an Intel i7-2600K@3.40Ghz CPU.

Metric	**Value**	**Explanation**
Memory	300 bytes/triple	Rough approximation of the storage needed per triple, including indexing. The value depends on the number of distinct URIs and literals as well as the length of URIs and literals.
Add triple (I)	3.5 μs	Add a triple with a URI object that is computed in Prolog, where the URI is already interned as a Prolog atom.
Add triple (II)	7.5 μs	Add a triple with a literal object that is computed in Prolog. This includes the time needed to intern the literal into the ordered literal store of the RDF database.
First triple	1.4 μs	Get the first triple from an `rdf/3` query. This is almost independent from the number of triples in the store and whether the arguments are known.
Next Triple	0.6 μs	Get the next triple from a `rdf/3` query.

for to facilitate community-based management of code fragments.

3. Related work

Triples stores with reasoning capabilities bear some similarity to deductive databases such as XSB [25]. Even though Datalog systems are not directly applicable to SW data, [1] uses Datalog to extend the SPARQL 1.1 language with more powerful navigational patterns. Despite strong theoretical similarities, only few Logic Programming systems exist that specialize in storing and querying SW data. One such a system is GiaBATA [12] which allows queries to be performed against named graphs and under partial RDFS and OWL entailment. It facilitates the implementation of custom reasoning capabilities through the specification of rules. Its implementation consists of a rewriting of SPARQL queries into Datalog programs. Instead of Prolog, it uses Answer Set Programming (ASP) as the embedded programming language. A similar approach is taken in the Smart-M3 platform [17], which is also based on ASP.

Euler [20] is an inference engine that uses hybrid reasoning and Euler path detection to avoid loops. As such, it is not intended to be used as a large-scale triple store, although it may use a triple store for input. Even though Euler runs on YAP and SWI-Prolog it is intended to be programmed in terms of Notation3 rules. Euler provides advanced reasoning paradigms such as abduction and induction besides deduction.

Several Semantic Web programming frameworks exist. Most frameworks are within the imperative and/or object-oriented paradigm, e.g., Jena [9] and Sesame [4] that are both based on Java and rdflib[6] is based on Python. Querying in these libraries typically consists of creating an *iterator* from either a SPARQL query or a simple triple pattern. For example, rdflib defines `triples()`. This interator can be used to find all people someone knowns at distance two as follows:

```
for s,p,o in \
    g.triples((me,FOAF.knows,None)):
  for x,y,z in \
      g.triples((o,FOAF.knows,None)):
```

In Prolog, non-determinism avoids the need for explicit interators and thus the fragment below suffices. Also note that this Prolog fragment is a non-deterministic goal and can thus be combined with other goals in a conjunction or disjunction. The Python loop above is *not* an iterator that can be reused ‘as is’.

```
rdf(X, foaf:knows, Y),
rdf(Y, foaf:knows, Z),
```

AllegroGraph[7] is a high-performance, persistent graph database. It combines in-memory with disk-based storage, which allows it to be very scalable (billions of quadruples). AllegroGraph supports SPARQL, RDFS++ and Prolog reasoning. In many senses, AllegroGraph has similar aims as ClioPatria. The main difference is that the principal ‘glue’ language is Common Lisp and the Prolog system (thus) uses Lisp syntax rather than Prolog syntax. We consider their inclusion of “Prolog reasoning for client applications” a

[6]`https://github.com/RDFLib/rdflib`
[7]`http://franz.com/agraph/allegrograph/`

strong support for ClioPatria's architecture. A side-by-side comparison of features is complex and outside the scope of this document.

4. Application areas

In this section we briefly describe a number of applications based on ClioPatria or SWI-Prolog's triple store. The applications are collected from our academic network, the mailinglist, literature research and searches on GitHub. We organised the applications in *application areas*, based on the way they (primarily) use ClioPatria's features. The section may be read both as a proof of impact and as a list of examples of how ClioPatria can be deployed.

4.1. Semantic search & faceted navigation

The semantic search application mentioned in Section 1 started the development of ClioPatria. This search engine is reused and extended in several projects, e.g., CHIP, DBtune, Europeana Thought Lab and K-Space NewsML demonstrator [27]. This application is used to explore faceted navigation based on the automatic generation of the search facets in /facet [11]. In [10] the approach used to build /facet is generalized to arbitrary web widgets (e.g., autocompletion) that dynamically adapt to the schema of the loaded data. These web widgets are used by art collection annotators of the Rijksmuseum to browse art thesauri and by end users to browse news articles.

In [28], ClioPatria is used to develop a mobile cultural heritage guide that combines information from the LOD cloud and cultural heritage institutes in order to display information about points of interest on a mobile phone. In this project ClioPatria serves multiple purposes. Firstly, as a triple store with common SW functionality such as graph search, semantic crawling and reasoning. Secondly, domain-specific reasoning capabilities, in this case spatial reasoning, are integrated on top of the generic SW reasoning capabilities. Thirdly, since mobile devices may have limited connectivity and processing power, ClioPatria performs the computationally heavy task of facet classification on the server side. In this way only the relevant information needs to be transmitted to the client application.

In [6,5] ClioPatria is used to allow scholars to search for cultural heritage resources in the Digital Archives of Italian psychology[8] in terms of the CIDOC-CRM[9] ontology.

4.2. Web application generator & platform

ClioPatria provides an integrated comprehensive web application platform, capable of dealing with all three tiers: storage, application logic and presentation. Most of the search applications from Section 4.1 are completely coded in ClioPatria.

As another example, Amalgame [32] provides a web-based interface for interactive vocabulary alignment. It is distributed as a ClioPatria CPACK (Section 2.4) and uses the triple store backend to store the RDF representations of the vocabularies that are aligned. It also stores an RDF representation of the alignment strategy that the user incrementally builds while interacting with the tool as well as the RDF representations of the alignment results. By using the triple store's persistency features, Amalgame exploits the advantages of a fast in-memory store with the possibility to continue tasks after server reboots or timed-out sessions. Amalgame has a flexible plugin-infrastructure that allows new or modified alignment algorithms to be inserted without restarting the web server or triple store. This is important in a research prototyping environment, where such changes happen frequently, and reloading large vocabularies after each change is too time consuming. Finally, Amalgame relies on the NLP extensions of the triple store to perform fast string matching between RDF literals (Section 2.1.1), for example using the built-in Porter stemmer and the *isub* [23] string distance matcher. The prefix index on literals is used to provide efficient autocompletion services. Such support is crucial as it allows the user to lookup terms interactively in order to correct alignment mistakes that were made by the system.

4.3. Web crawling & robustness

LOD Laundromat[10] [3] is a system that performs a large, LOD cloud-scale data cleaning operation. It searches for files and endpoints that might contain RDF triples starting from seed points. First, possible compression and archiving (e.g., ZIP) is analysed

[8] Archivi Storici della Psicologia Italiana (ASPI), `http://www.aspi.unimib.it`

[9] `http://www.cidoc-crm.org/`

[10] `http://lodlaundromat.org/`

based on the SWI-Prolog binding to libarchive[11]. Next, for each individual file, heuristics are used to detect whether the data contains RDF and – if so – what serialization format. Finally, the data is parsed by the applicable RDF parser from ClioPatria while keeping statistics about errors and warnings. LOD Laundromat has processed hundreds of thousands of data documents, parsing tens of billions of triples. All data is disseminated into a canonical and standards-compliant format. This project shows that ClioPatria has sufficient robustness to support large-scale processing.

4.4. Data integration & enrichment

ClioPatria, due to its underlying triple store that stresses dynamic data, has proven to be a suitable platform for combining RDF data with external data, as well as for generating and restructuring RDF data. In such data integration scenarios, Prolog is used as the rule language and 'glue' towards other data sources such as relational databases.

SWI-Prolog's RDF infrastructure has been used to convert WordNet to RDF [29]. The XMLRDF toolkit[12], written on top of ClioPatria, converts XML datadumps into RDF using a generic mapping, after which the RDF can be transformed into the desired shape by using rules that are translated into Prolog. XMLRDF has been used in the Europeana Digital Library project as well as for the conversion and enrichment of many Dutch cultural heritage datasets and thesauri [8,7].

The Blipkit [18] system is a comprehensive system for querying and transforming data in bioinformatics. Blipkit's RDF support in terms of performance and expressiveness was evaluated by Lampa [15]. Lampa stresses the enhanced maintainability of Prolog queries due to reuse of query fragments represented as predicates (Figure 18 in [15]).

4.5. Wiki systems

PlWiki [19] is a semantic wiki implemented on top of ClioPatria. The authors claim that a crucial reason for using a Prolog based solution is the transparent integration of various rule formalisms such as SWRL and XTT.

[11]`http://www.libarchive.org/`
[12]`http://semanticweb.cs.vu.nl/xmlrdf/`

4.6. Reasoning

We collected some projects related to reasoning on top of the Prolog RDF store. Many projects about reasoning do not require the higher level ClioPatria features.

4.6.1. Robotics

The openEASE[13] uses the SWI-Prolog RDF infrastructure for the KnowRob [26] knowledge base, which represents ontological knowledge in OWL. Prolog is used to integrate this rather abstract information with the grounded data on the robot such as coordinates, poses, images, time points etc, and make use of procedural attachments to OWL classes and properties for integrating external reasoners, e.g., to compute based on a robot's kinematics if it is able to reach towards a point in space.

4.7. Counterfactual reasoning

Actual-Causation[14] is a research project in which causal models can be created and simulated. According to a popular definition in analytic philosophy, causation is defined in terms of counterfactuals, i.e. conditions that would apply if the course of events had been different. Implementation-wise, causal models are collections of RDF statements that reside in a ClioPatria triple store. The snapshot feature of ClioPatria is used to make isolated and short-living copies of the database, one for each counterfactual. Within each snapshot specific facts are added/retracted/altered in order to express the respective counterfactuals. Backward chaining is used to validate whether certain causal effects still follow under those different circumstances. In a triple store that does not support snapshots, this would have been implemented less elegantly, e.g., by inefficiently copying data between an extensive number of named graphs.

4.8. Spatial reasoning

In the Poseidon project [30] ClioPatria is used to reason about ship trajectories. The library that allows for combining Linked Data with spatial reasoning has been made available as a SWI-Prolog package [31] and is an example of integrating domain-specific knowledge with SW reasoning facilities into a single clean paradigm.

[13]`http://www.open-ease.org/getting-started/`
[14]`http://wouterbeek.github.io/Actual-Causation`

4.9. *Qualitative reasoning*

WebQR [2] is another SW-based domain-specific knowledge system that allows physical systems to be qualitatively simulated. In WebQR a user creates a model whose components are SW resources. A ClioPatria-based caching infrastructure is used to retrieve knowledge statements about those components from the web. Traditional RDF(S) entailment, in combination with domain-specific reasoning rules, is used in order to automate modeling steps that would otherwise be performed by human modelers. For example, WebQR can decide whether a modeled resource can vary over time (=quantity) or not (=entity). Whether a derivation is possible depends on whether specific information has yet been cached, using ClioPatria's dynamic triple storage capabilities.

5. Future work

ClioPatria is being actively developed and its features are continuously expanded to satisfy project requirements. Below we give a list of recognised limitations.

- The current system has limited scalability of its RDF store because it is purely memory based. The strong dynamic RDF support of the platform allows the local store to be used as a cache which relies on large external stores for background knowledge.
- Reasoning that requires recursion often puts the onus on the user to explicitly avoid cycles. This can be avoided by using *tabling*, a technique that is available in the Prolog systems XSB and YAP, but not yet in SWI-Prolog.
- Standardized SW reasoning in ClioPatria is currently limited to RDFS++. For example, full-scale OWL-DL reasoning is not yet supported, although this should be relatively easy given the underlying architecture.
- ClioPatria implements SPARQL 1.1 query and update requests, but not yet SPARQL federation.
- While Lampa [15] claims that queries written in Prolog often outperform competing SPARQL endpoints, poor integration of SPARQL literal operations into the execution plan does not yet expose this performance to the SPARQL endpoint of ClioPatria.

6. Conclusion

In this paper we have presented the ClioPatria Semantic Web toolkit. ClioPatria is based on SWI-Prolog and is tightly connected to an efficient main-memory RDF quad store. We have shown that the use of ClioPatria and its Logic Programming (LP) paradigm has several benefits as a prototyping environment for SW programming. Accessing RDF data using LP does not suffer from the object-relational impedance mismatch and LP allows expressions to be built from small reusable components. By providing a general purpose language closely connected to the triple store we can formulate more expressive queries in graphs as well as on literals. An emphasis on backward chaining makes LP suitable for reasoning with dynamic RDF. This feature is reflected in the database by supporting transactions and isolated modifications in *snapshots*.

The toolkit is completed by standard SW functionality such as a SPARQL 1.1 endpoint, parsers and serializers for most RDF formats, a plugin infrastructure and a web frontend. The most important application areas for ClioPatria are currently data integration and enrichment and semantic search.

The recent introduction of Pengines (Prolog Engines) and SWISH (SWI-Prolog SHaring/SHell) allows the use of flexible Prolog rule-based reasoning without changing the software on the server. Pengines allow for remote execution of Prolog programs and can be accessed both from JavaScript to realise web applications and from Prolog to realise distributed computing.

Acknowledgements
The development of ClioPatria has been initiated under the Dutch MultimediaN project. Further development has taken place in the context of many different projects. This publication was supported by the Dutch national program COMMIT/

References

[1] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the Semantic Web. In *Proceedings of PODS*, 2014.
[2] Wouter Beek, Sander Latour, and Stefan Schlobach. WebQR: Building a knowledge representation application on the semantic web. In *Proceedings of the ISWC Developers Workshop 2014*, pages 102–107, 2014.
[3] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD laundromat: A uniform way of publishing other people's dirty data. In *The*

Semantic Web–ISWC 2014, pages 213–228. Springer International Publishing, 2014.

[4] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *The Semantic Web–ISWC 2002*, pages 54–68. Springer, 2002.

[5] Claudio Cortese. Semantic search and browsing nell'ambito dei beni culturali. *Bollettino del CILEA*, (118):26–30, 2012.

[6] Claudio Cortese and Glauco Mantegari. Extending the digital archives of italian psychology with semantic data. In *Proceedings of the 1st International Workshop on Semantic Digital Archives (SDA 2011)*, pages 60–71, 2011.

[7] Victor de Boer, Matthias van Rossum, Jurjen Leinenga, and Rik Hoekstra. Dutch ships and sailors linked data. In *The Semantic Web–ISWC 2014*, pages 229–244. Springer, 2014.

[8] Victor De Boer, Jan Wielemaker, Judith Van Gent, Michiel Hildebrand, Antoine Isaac, Jacco van Ossenbruggen, and Guus Schreiber. Supporting Linked Data production for cultural heritage institutes: the Amsterdam Museum case study. In *The Semantic Web: Research and Applications*, pages 733–747. Springer, 2012.

[9] Michael Grobe. RDF, Jena, SPARQL and the 'Semantic Web'. In *Proceedings of the 37th annual ACM SIGUCCS fall conference*, pages 131–138. ACM, 2009.

[10] Michiel Hildebrand and Jacco van Ossenbruggen. Configuring semantic web interfaces by data mapping. *Visual Interfaces to the Social and the Semantic Web (VISSW 2009)*, 443:96, 2009.

[11] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. /facet: A browser for heterogeneous semantic web repositories. In *The Semantic Web-ISWC 2006*, pages 272–285. Springer, 2006.

[12] Giovambattista Ianni, Thomas Krennwallner, Alessandra Martello, and Axel Polleres. A rule system for querying persistent RDFS data. In *Proceedings of the 6th European Semantic Web Conference*, 2009.

[13] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A classification of object-relational impedance mismatch. In *Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*, DBKDA '09, pages 36–43, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Torbjörn Lager and Jan Wielemaker. Pengines: Web logic programming made easy. *arXiv preprint arXiv:1405.3953*, 2014.

[15] Samuel Lampa. SWI-Prolog as a Semantic Web tool for semantic querying in Bioclipse: Integration and performance benchmarking. Master's thesis, 2010.

[16] Leonid Libkin, Juan Reutter, and Domagoj Vrgoč. Trial for RDF: adapting graph query languages for RDF data. In *Proceedings of the 32nd symposium on Principles of database systems*, pages 201–212. ACM, 2013.

[17] Vesa Luukkala and Ilkka Niemelä. Enhancing a smart space with answer set programming. In *Semantic Web Rules*, pages 89–103. Springer, 2010.

[18] Chris Mungall. Experiences using logic programming in bioinformatics. In Patricia M. Hill and David Scott Warren, editors, *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2009.

[19] Grzegorz J Nalepa. Plwiki–a generic semantic wiki architecture. In *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, pages 345–356. Springer, 2009.

[20] Pieterjan De Potter, Hans Cools, Kristof Depraetere, Giovanni Mels, Pedro Debevere, Jos De Roo, Csaba Huszka, Dirk Colaert, Erik Mannens, and Rik Van de Walle. Semantic patient information aggregation and medicinal decision support. *Computer Methods and Programs in Biomedicine*, 108(2):724–735, 2012.

[21] Laurens Rietveld and Rinke Hoekstra. YASGUI: Not just another SPARQL client. In *The Semantic Web: ESWC 2013 Satellite Events*, pages 78–86. Springer, 2013.

[22] Guus Schreiber, Alia Amin, Mark van Assem, Victor de Boer, Lynda Hardman, Michiel Hildebrand, Laura Hollink, Zhisheng Huang, Janneke van Kersen, Marco de Niet, et al. MultimediaN E-Culture demonstrator. In *The Semantic Web-ISWC 2006*, pages 951–958. Springer, 2006.

[23] Giorgos Stoilos, Giorgos B. Stamou, and Stefanos D. Kollias. A string metric for ontology alignment. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, volume 3729 of *Lecture Notes in Computer Science*, pages 624–637. Springer, 2005.

[24] Terrance Swift and David S Warren. XSB: Extending Prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12(1-2):157–187, 2012.

[25] Terrance Swift and David S. Warren. XSB: Extending Prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12:157–187, 1 2012.

[26] Moritz Tenorth and Michael Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *International Journal of Robotics Research (IJRR)*, 32(5):566 – 590, April 2013.

[27] Raphaël Troncy. *Bringing the IPTC news architecture into the Semantic Web*. Springer, 2008.

[28] Chris Van Aart, Bob Wielinga, and Willem Robert Van Hage. Mobile cultural heritage guide: location-aware semantic search. In *Knowledge engineering and management by the masses*, pages 257–271. Springer, 2010.

[29] Mark Van Assem, Aldo Gangemi, and Guus Schreiber. Conversion of WordNet to a standard RDF/OWL representation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, pages 237–242, 2006.

[30] Willem Robert van Hage, Véronique Malaisé, Gerben de Vries, Guus Schreiber, and Maarten van Someren. Abstracting and reasoning over ship trajectories and web data with the Simple Event Model (SEM). *Multimedia Tools Appl.*, 57(1):175–197, 2012.

[31] Willem Robert Van Hage, Jan Wielemaker, and Guus Schreiber. The space package: Tight integration between space and semantics. *Transactions in GIS*, 14(2):131–146, 2010.

[32] Jacco van Ossenbruggen, Michiel Hildebrand, and Viktor de Boer. Interactive vocabulary alignment. In Stefan Gradmann, Francesca Borri, Carlo Meghini, and Heiko Schuldt, editors, *Research and Advanced Technology for Digital Libraries - International Conference on Theory and Practice of Digital Libraries, TPDL 2011, Berlin, Germany, September 26-28, 2011. Proceedings*, volume 6966 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 2011.

[33] Jan Wielemaker. An optimised Semantic Web query language implementation in Prolog. In *Logic Programming*, pages 128–

142. Springer, 2005.

[34] Jan Wielemaker. *Logic programming for knowledge-intensive interactive applications*. PhD thesis, University of Amsterdam, 2009. http://dare.uva.nl/en/record/300739.

[35] Jan Wielemaker. Extending the logical update view with transaction support. *CoRR*, abs/1301.7669, 2013.

[36] Jan Wielemaker, Michiel Hildebrand, and Jacco van Ossenbruggen. Using Prolog as the fundament for applications on the Semantic Web. *Proceedings of ALPSWS2007*, pages 84–98, 2007.

[37] Jan Wielemaker, Michiel Hildebrand, Jacco van Ossenbruggen, and Guus Schreiber. Thesaurus-based search in large heterogeneous collections. 2008.

[38] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.