PlanetData

Network of Excellence

FP7 – 257641

D33.1 MetaReasons: prototype evaluation (modelling and scalability)

Coordinator: Loris Bozzato, Luciano Serafini (FBK)

1st Quality Reviewer: Max Schmachtenberg (UMA)
2nd Quality Reviewer: Nguyen Quoc Viet Hung (EPFL)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M48
Actual delivery date:	M48
Version:	1.0
Total number of pages:	23
Keywords:	Metadata models, Metadata reasoning prototype, Scalability evaluation

Abstract

Different models have been proposed for the management of meta-data in the context of linked data regarding different aspects (like e.g. provenance, access control, privacy and trust). In the MetaReasons approach we propose a framework that aims at seamlessly combining and reasoning with several metalevel aspects, where each one is encoded as a separate "meta-theory" module in the representation of metadata.
Following the implementation of MetaReasons prototype presented in previous deliverable D32.1b, this document presents the evaluation of the final prototype with respect to scalability of reasoning and to modelling.
We will first present the experimental setup and the tools we used to generate the scalability test sets: using such datasets, we will present the results of the evaluation with respect to different reasoning regimes and different solutions for data storage. We then present the practical application of the MetaReasons architecture to model real use cases in which combination of metadata on different datasets is a central issue.

Executive Summary

This deliverable presents the results of the prototype evaluation task (T33.1) of the *MetaReasons* activity (WP31-WP33) in the PlanetData project.

The *MetaReasons* approach has as objective the definition of a unified framework to represent and reason about provenance, access control, privacy and trust meta information of linked data datasets. Activities in this line of work consist of: the definition of the theoretical architecture of the framework and the metatheories encoding different models for the considered aspects (WP31); the implementation of such framework in a working prototype (WP32); the evaluation of the resulting prototype with respect to modelling and scalability criteria (WP33).

The objective of the present task (T33.1 of WP33) is to provide an assessment of the final prototype with respect to scalability of reasoning and applicability to modelling test cases for the considered metadata aspects.

In this deliverable we document the results of the task by providing an experimental evaluation of the developed prototype with respect to scalability and modelling on real use cases scenarios.

After a brief summary of the framework model and implementation, we present the results of the framework evaluation with respect to scalability: we first present the tools and methods we used to generate the analyzed test sets and we show the results of the prototype evaluation with respect to different reasoning regimes and storage solutions.

We then present how MetaReasons has been applied in the modelling of a real usecase to combine reasoning over different metadata aspects relative to several datasets, thus demonstrating the applicability of the framework in a real scenario.

Document Information

IST Project Number	FP7 – 257641	Acronym	PlanetData
Full Title	PlanetData		
Project URL	http://www.planet-data.eu/		
Document URL	http://planet-data-wiki.sti2.at/web/D33.1		
EU Project Officer	Leonhard Maqua		

Deliverable	Number	D33.1	Title	MetaReasons: prototype evaluation (modelling and scalability)
Work Package	Number	WP33	Title	MetaReasons: Prototype Evaluation

Date of Delivery	Contractual	M48	Actual	M48
Status	version 1.0		final ☒	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	Loris Bozzato, Luciano Serafini (FBK)			
Responsible Author	Name	Luciano Serafini	E-mail	serafini@fbk.eu
	Partner	Fondazione Bruno Kessler (FBK)	Phone	+39 (0461) 314 319

Abstract (for dissemination)	Different models have been proposed for the management of meta-data in the context of linked data regarding different aspects (like e.g. provenance, access control, privacy and trust). In the MetaReasons approach we propose a framework that aims at seamlessly combining and reasoning with several metalevel aspects, where each one is encoded as a separate "meta-theory" module in the representation of metadata. Following the implementation of MetaReasons prototype presented in previous deliverable D32.1b, this document presents the evaluation of the final prototype with respect to scalability of reasoning and to modelling. We will first present the experimental setup and the tools we used to generate the scalability test sets: using such datasets, we will present the results of the evaluation with respect to different reasoning regimes and different solutions for data storage. We then present the practical application of the MetaReasons architecture to model real use cases in which combination of metadata on different datasets is a central issue.
Keywords	Metadata models, Metadata reasoning prototype, Scalability evaluation

Version Log			
Issue Date	Rev. No.	Author	Change
08/09/2014	0.1	Loris Bozzato, Luciano Serafini (FBK)	First version.
25/09/2014	0.2	Loris Bozzato, Luciano Serafini (FBK)	Addressed feedback from reviewers.
29/09/2014	1.0	Loris Bozzato, Luciano Serafini (FBK)	Final version.

Contents

Executive Summary 3

Document Information 4

1 Introduction 8

2 Scalability of reasoning 10
2.1 Generation of synthetic test sets 10
2.2 Experimental Setup 11
2.3 Evaluation results 12

3 MetaReasons in modelling: OpenData Trentino use case 17
3.1 OpenData Trentino portal 17
3.2 A MetaReasons repository for OpenData Trentino 17
3.3 Unified queries examples 19

4 Conclusions 22

List of Figures

1.1 MetaReasons architecture. 8

2.1 Scalability graphs for OWLIM. 14
2.2 Scalability graphs for Sesame. 15
2.3 Comparison graphs for Sesame and OWLIM. 16

3.1 Import of OpenData Trentino datasets to RDF repository. 18
3.2 Metalevel structure for OpenData Trentino repository. 19
3.3 Query interface to OpenData Trentino repository. 20

Abbreviations

CKAN Comprehensive Knowledge Archive Network

CSV Comma Separated Values

GTFS General Transit Feed Specification

JSON JavaScript Object Notation

MR MetaReasons

MT Metatheory

NG Named Graph

PAT Provincia Autonoma di Trento (Autonomous Province of Trento)

REST REpresentational State Transfer

SPRINGLES SParql-based Rule Inference over Named Graphs Layer Extending Sesame

UKB Unified Knowledge Base

1 Introduction

The *MetaReasons (MR)* approach proposes a framework that aims at seamlessly combining and reasoning with several metalevel aspects of metadata for linked data management, such as metadata for provenance, access control, privacy and trust. Each metadata aspect is encoded as a separate "meta-theory" module in the representation of metadata.

The MR framework architecture has been first introduced in our previous deliverable D31.1 [2]. An intuitive representation of MetaReasons architecture is depicted in Figure 1.1. The framework is defined over a two

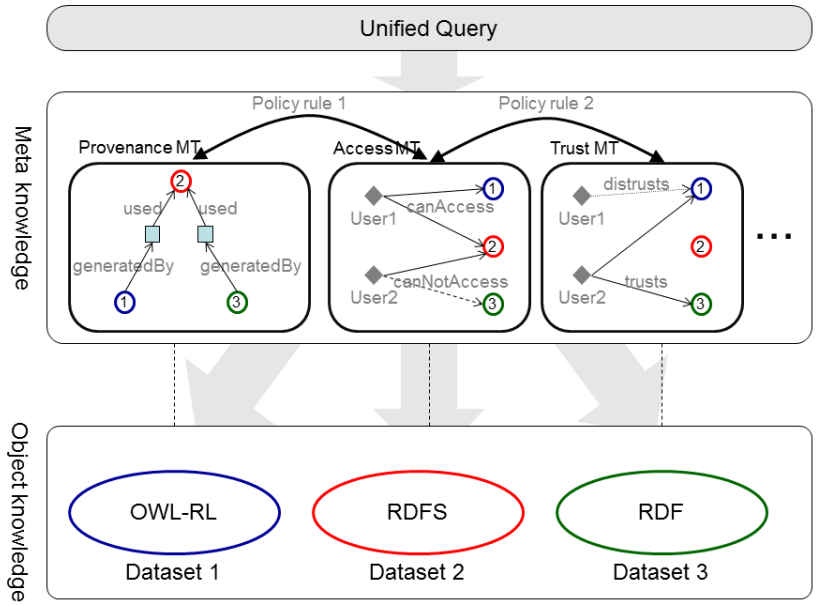

Figure 1.1: MetaReasons architecture.

layered structure: the upper layer, called *metalevel* (or meta knowledge level), represents the metadata information of datasets while contents of the datasets are represented in the lower layer, called *object level* (or object knowledge level). Datasets are seen as atomic individuals in the metalevel: their meta information is linked to these dataset identifiers. The different metadata aspects (e.g. provenance, access level or trust of each dataset) are encoded in distinct *metatheories*, metalevel theories that independently describe a particular aspect of the datasets. *Policies*, connecting different metatheories can be defined in the part of the metaknowledge covering all of the single theories for the different aspects.

On this architecture, *unified queries* over both the meta-knowledge and the object knowledge can be expressed in standard SPARQL (possibly extended with the primitives defined by the architecture). Such queries can span over the knowledge in the integrated datasets by constraining their properties in the metaknowledge: for example, if we want to retrieve "all of the facts about a certain event, from datasets which have a certain level of trust and for which we have access rights" we can express this in a SPARQL query that selects the datasets matching the meta-level requirements and then run the object-query "all of the facts about a certain event" on the selected datasets.

This architecture has been formally defined in deliverable D31.1 [2] and completed with a set of metatheories in deliverable D31.2 [3]. In previous deliverable D32.1b [4], MetaReasons architecture has been implemented over an extension of the Sesame RDF Platform [5] that supports reasoning on named graphs called *SPRINGLES (SParql-based Rule Inference over Named Graphs Layer Extending Sesame)*. The prototype accepts RDF input data expressing OWL-RL axioms and assertions for both the global and local knowledge modules: these different pieces of knowledge are represented as distinct named graphs. The prototype is structured in a client

server architecture: the main component, called *MR core* module and residing in the server-side part, exposes the MR primitives and a SPARQL 1.1 endpoint for query and update operations on the contextualized knowledge. The MR core module offers the ability to compute and materialize the inference closure of the input MR, add and remove knowledge and execute queries over the complete MR structure. The distribution of knowledge in different named graphs asks for a module to compute inference over multiple graphs in a RDF store, since inference mechanisms in current stores usually ignore the graph part: this component is realized by the SPRINGLES software layer. Intuitively, the layer provides methods to demand a closure materialization on the RDF store data: rules are encoded as named graphs aware SPARQL queries and it is possible to customize both the input ruleset and the evaluation strategy. In deliverable D32.1b [4] we detailed how the formal calculus for OWL-RL knowledge bases has been implemented as a SPRINGLES ruleset and evaluation strategy in the prototype implementation.

In this deliverable we describe an evaluation for the realized prototype. In Chapter 2 we describe the test we performed on scalability of reasoning for the developed prototype: we first present the procedure we used to generate the synthetic test sets and the form of such datasets. We then present the results of the evaluation of the prototype over such datasets with respect to different reasoning rulesets and different solutions for RDF data storage. In Chapter 3 we present a real application scenario in which we applied the MR framework. The OpenData Trentino repository demonstrates an example of relatively large unified knowledge base in which the combination of reasoning over different metadata aspects and the integration of reasoning from different data sources is needed.

2 Scalability of reasoning

In this chapter we illustrate the experiments we performed to assess the scalability of the MetaReasons approach and their results. We begin by presenting the synthetic test sets we generated for such evaluation and the method we used to create them.

2.1 Generation of synthetic test sets

We performed an evaluation of the MetaReasons prototype with respect to scalability of reasoning in OWL-RL: the test sets of UKBs over which such evaluation is performed have been synthetically generated.

In order to create such synthetic UKB sets, we developed a simple generator that can output randomly generated UKBs with certain features. In particular, for each generated UKB, the generator takes in input:

– the number n of datasets (i.e. local named graphs) to be generated;

– the dimensions of the signature to be declared (number m of base classes, l of properties and k of individuals);

– the axiom size for the global and local modules (number of global TBox, ABox and RBox axioms and number of TBox, ABox and RBox axioms per dataset).

The generated UKBs are then saved as TRIG files, that can be readily imported in the RDF store for MetaReasons. Intuitively, the generation of a UKB proceeds as follows:

1. The datasets (named `:d0`, . . . , `:dn`) are declared in the metaknowledge named graph.
2. Base classes (named `:A0`, . . . , `:Am`), object properties (named `:R0`, . . . , `:Rl`) and individuals (named `:a0`, . . . , `:ak`) are added to the metaknowledge (and they are then used in the generation of global and local axioms).
3. Then generation of global axioms take place: we chose to generate axioms as follow to create realistic instances of knowledge bases.
   – classes and properties names are taken from the base signature using a random selection criteria in the form of (the positive part of) a gaussian curve centered in 0: intuitively, classes equal or near `:A0` are more probable to appear in axioms than `:An`.
   – individuals are randomly selected using a uniform distribution.
   – TBox, ABox and RBox axioms in $\mathcal{SROIQ}$-RL are added in the requested number to the metaknowledge module following the percentage shown in Table 2.1 (note that the reported axioms are normal form $\mathcal{SROIQ}$-RL axioms, as defined in D31.1 [2]).
4. The same generation criteria are then applied in the case of local graphs representing the contents of datasets.
5. The resulting UKB is then saved to a single TRIG file.

Using this tool, we generated one set of test UKBs (called *MR1*), shown in Table 2.2: The idea of test set MR1 is the following: we generated sets of UKBs with an increasing number of datasets, in which UKBs have an increasing number of axioms. Aim of this set is to simply assess how the system performs in presence of an increase both in number and in dimension of datasets. In order to smooth the influence of anomalies given by particular cases in the randomly generated axioms, we performed our tests over 4 instances of generation of testset MR1.

TBox axiom	%
$A \sqsubseteq B$	50%
$A \sqsubseteq \neg B$	20%
$A \sqsubseteq \exists R.\{a\}$	10%
$A \sqcap B \sqsubseteq C$	5%
$\exists R.A \sqsubseteq B$	5%
$A \sqsubseteq \forall R.B$	5%
$A \sqsubseteqslant 1R.B$	5%

ABox axiom	%
$A(a)$	40%
$R(a,b)$	40%
$\neg R(a,b)$	10%
$a = b$	5%
$a \neq b$	5%

RBox axiom	%
$R \sqsubseteq T$	50%
$\mathrm{Inv}(R,S)$	25%
$R \circ S \sqsubseteq T$	10%
$\mathrm{Dis}(R,S)$	10%
$\mathrm{Irr}(R)$	5%

Table 2.1: Percentages of generated axioms.

				Metalevel KB			Object level KBs			
Datasets	**Classes**	**Roles**	**Individuals**	**TBox**	**RBox**	**ABox**	**TBox**	**RBox**	**ABox**	**Total ax.**
1	10	10	20	10	5	20	10	5	20	70
1	50	50	100	50	25	100	50	25	100	350
1	100	100	200	100	50	200	100	50	200	700
1	500	500	1000	500	250	1000	500	250	1000	3.500
1	1000	1000	2000	1000	500	2000	1000	500	2000	7.000
5	10	10	20	10	5	20	10	5	20	210
5	50	50	100	50	25	100	50	25	100	1.050
5	100	100	200	100	50	200	100	50	200	2.100
5	500	500	1000	500	250	1000	500	250	1000	10.500
5	1000	1000	2000	1000	500	2000	1000	500	2000	21.000
10	10	10	20	10	5	20	10	5	20	385
10	50	50	100	50	25	100	50	25	100	1.925
10	100	100	200	100	50	200	100	50	200	3.850
10	500	500	1000	500	250	1000	500	250	1000	19.250
10	1000	1000	2000	1000	500	2000	1000	500	2000	38.500
50	10	10	20	10	5	20	10	5	20	1.785
50	50	50	100	50	25	100	50	25	100	8.925
50	100	100	200	100	50	200	100	50	200	17.850
50	500	500	1000	500	250	1000	500	250	1000	89.250
50	1000	1000	2000	1000	500	2000	1000	500	2000	178.500
100	10	10	20	10	5	20	10	5	20	3.535
100	50	50	100	50	25	100	50	25	100	17.675
100	100	100	200	100	50	200	100	50	200	35.350
100	500	500	1000	500	250	1000	500	250	1000	176.750
100	1000	1000	2000	1000	500	2000	1000	500	2000	353.500

Table 2.2: Test set MR1.

2.2 Experimental Setup

We carried out experiments on previously presented test set with the task to determine the (average) inference time and repository dimensions with respect to the increase in number of datasets and their contents.

Evaluation experiments were carried out on a 4 core Dual Intel Xeon Processor machine with 32Gb 1866MHZ DDR3 RAM, standard S-ATA (7.200RPM) HDD, running a Linux RedHat 6.5 distribution. We allocated 6Gb of memory to the JVM running the SPRINGLES web-app (i.e. the RDF storage and inference prototype), while 20Gb were allocated to the utility program managing the upload, profiling and cleaning of the test repositories.

In order to abstract from the possible overhead for the repository setup, the tests have been averaged over 5 runs of the closure operation for each UKB. Moreover, the tests results have been averaged over the 4 versions of MR1 in order to reduce the impact of special cases in UKB random generation.

The tests were carried out on different MetaReasons rulesets in order to study their applicability in practical reasoning. The rulesets are limitation to the full set of rules and evaluation strategy presented in previous deliverable D32.1b [4], in particular:

– *mr-rdfs-global:* inference is only applied to the metaknowledge (no local reasoning inside dataset named graphs). Applies only inference rules for RDFS and metatheory rules.

– *mr-rdfs-local:* inference is applied to the metaknowledge and to dataset graphs. Again, applies only inference rules for RDFS and metatheory rules.

			mr-rdfs-global			mr-owl-global			mr-rdfs-local			mr-owl-local		
Ds.	**Cls.**	**Triples**	**Inf.**	**Time O.**	**Time S.**	**Inf.**	**Time O.**	**Time S.**	**Inf.**	**Time O.**	**Time S.**	**Inf.**	**Time O.**	**Time S.**
1	10	208	233	70	208	241	171	426	254	123	267	279	256	509
1	50	1052	1145	128	259	1264	191	456	1239	137	303	1500	352	850
1	100	2118	2327	84	270	2499	224	473	2549	150	328	2918	400	1129
1	500	10565	11950	681	994	13296	3165	8790	13149	1196	1721	15767	6158	29427
1	1000	21148	24008	1032	1573	26475	9049	26215	26197	1984	2871	31419	20365	100403
5	10	633	685	26	187	696	51	217	774	58	211	868	142	354
5	50	3099	3259	102	193	3363	267	396	3773	347	343	4361	836	1917
5	100	6158	6437	188	262	6728	563	939	7599	673	589	9305	2399	8076
5	500	30727	32124	1324	907	33456	6773	17719	38379	4711	3832	46629	21255	153700
5	1000	61881	64275	2179	1430	67443	20631	67432	76644	9235	7260	93133	77879	854971
10	10	1127	1272	53	189	1299	120	232	1426	117	209	1631	445	577
10	50	5600	5868	245	231	5986	495	415	7047	1057	528	8401	2688	6329
10	100	11242	11571	410	264	11940	1756	1947	13872	2275	1112	16745	6878	21461
10	500	56330	57840	3166	1182	59025	15170	26704	70370	13809	7425	85854	56666	679196
10	1000	112465	114996	5418	2130	117781	34961	98846	140579	31444	17000	172329	159698	timedout
50	10	5178	7841	467	294	7863	5939	340	8592	1529	555	9538	372649	4343
50	50	25902	28642	2850	432	28794	7636	1219	33950	25806	2906	40591	71393	130921
50	100	51869	54760	6810	545	55003	11337	3192	65989	43200	6874	79579	117258	476160
50	500	259732	263874	45259	2610	265259	111360	134413	326517	250292	54407	399521	908221	timedout
50	1000	520111	525241	104110	4786	527641	265340	431631	655206	633515	138393	timedout	timedout	timedout
100	10	10361	20677	1637	541	20702	216826	685	22051	6318	1388	23778	timedout	23814
100	50	51336	61736	9807	740	61822	23590	1731	72669	60201	8479	84476	timedout	460497
100	100	103311	113848	23719	932	114067	50337	8937	136742	152547	24159	timedout	timedout	timedout
100	500	514708	526260	172180	4158	527568	327231	252335	650471	1151101	173834	timedout	timedout	timedout
100	1000	1028920	1042109	441667	8235	1045381	1123687	1449134	1297301	timedout	373847	timedout	timedout	timedout

Table 2.3: Scalability results for test set MR1.

– *mr-owl-global:* inference is only applied to the metaknowledge, considering all of the inference rules for $\mathcal{SROIQ}$-RL and metatheory rules.

– *mr-owl-local:* full strategy defined by the calculus [4]. Inference is applied to the metaknowledge and knowledge part, using all of the (meta and local) $\mathcal{SROIQ}$-RL and metatheory rules.

More in details, with respect to the rule evaluation strategy presented in Section 3.3 of D32.1b [3], metatheory rules correspond to the metatheory specific tasks in `:step2_metakb_closure` in the closure of the metalevel. The application of RDFS rules corresponds to the limitation of `:task_compute_rl_closure` (in the metalevel and object level closure steps) only to the inference rules for subsumption on classes (rule *prln-subc*) and roles (rule *prln-subr*). In order to assess the impact of the RDF storage solution to the system perfomances, we performed the experiments using as back-ends both *OWLIM lite*[1] and *Sesame Native*[2] RDF stores.

2.3 Evaluation results

The results of the experiments on the presented test set are reported in Table 2.3. In the table, for each of the generated UKBs (referred by number of datasets and number of base classes in the first two columns), we show the number of total asserted triples in column *triples* (averaged on the 4 versions of MR1). The following columns list the results[3] of the closure for each of the rulesets: for a ruleset, we list the (average) total number of triples (asserted + inferred) and the (average) time in milliseconds for the closure operation for OWLIM (*Time O.*) and Sesame (*Time S.*). The value *timedout* in the measures indicates that the closure operation exceeded 30 minutes (1.800.000 ms.).

In order to analyze the results, the behaviour of the prototype for each of the rulesets has been plotted to graphs: in Figure 2.1 it is shown the case for the OWLIM back-end and in Figure 2.2 for Sesame Native storage. Each of the lines represents a set with a fixed number of datasets (1 to 100) and each point a UKB. The X axis

[1] `http://www.ontotext.com/owlim`

[2] `http://www.openrdf.org/`

[3] Values as the ones grayed out in the tables are due to particular cases in the generated axioms: these must be considered as anomalies and have been left out from the graphs.

represents the number of asserted triples, while the Y axis shows the time in milliseconds; the red horizontal line depicts the 30 minutes limit for timeout.

Some conclusions can be derived from these data and graphs: the first most evident fact is that the reasoning regime strongly impacts the scalability of the system. Thus, in practical cases the choice of a naive application of the full OWL RL ruleset might not be viable, in presence of large local datasets: on the other hand, as we will present in next chapter, we managed to apply the system to a relatively large real usecase (in which expressive reasoning inside datasets is not required) by relying on the RDFS rulesets. On the other hand, the management of the named graphs separation of knowledge (with equal number of triples) does not seem to directly impact on the scalability of the prototype.

Another clear fact is the dependence of performances to the RDF store backend: in the set of graphs shown in Figure 2.3 we directly compare the behaviour of the two RDF storage solutions. The comparison is given with respect to the UKBs set with the biggest number of datasets in which we can compare the two (i.e. 100 contexts UKBs for *mr-rdfs-global*, *mr-owl-global* and *mr-rdfs-local*, and 10 contexts UKBs for *mr-owl-local*). The blue dashed line represents the behaviour of OWLIM, while the red line represent the performances of Sesame. Again, the X axis represent the number of triples on the UKBs and the Y axis the time in milliseconds for the inference closure operation and the red line depicts the 30 minutes limit for timeout.

From the above table and these graphs, we can see that the two back-ends influence the performances of the MR prototype depending on different factors: in particular, it appears that Sesame Native performs better in the case of *mr-rdfs-global* and *mr-rdfs-local* rulesets, suggesting that it is the best solution in the case of repositories with lower expressivity and possibly large datasets. On the other hand, from the experiments OWLIM appears to better manage the complex rulesets, while in general poses more overhead on the smaller knowledge bases. (cfr. graphs for *mr-owl-global* and *mr-owl-local*). We remark that the operation of inference computation is functionally independent from the underlying solution for data storage[4], thus the different performances in the two considered solutions should be attributed to the different implementation of the stores (e.g. their solution in the storage of named graphs).

[4]Cfr. architecture of the prototype presented in deliverable D32.1b [4].

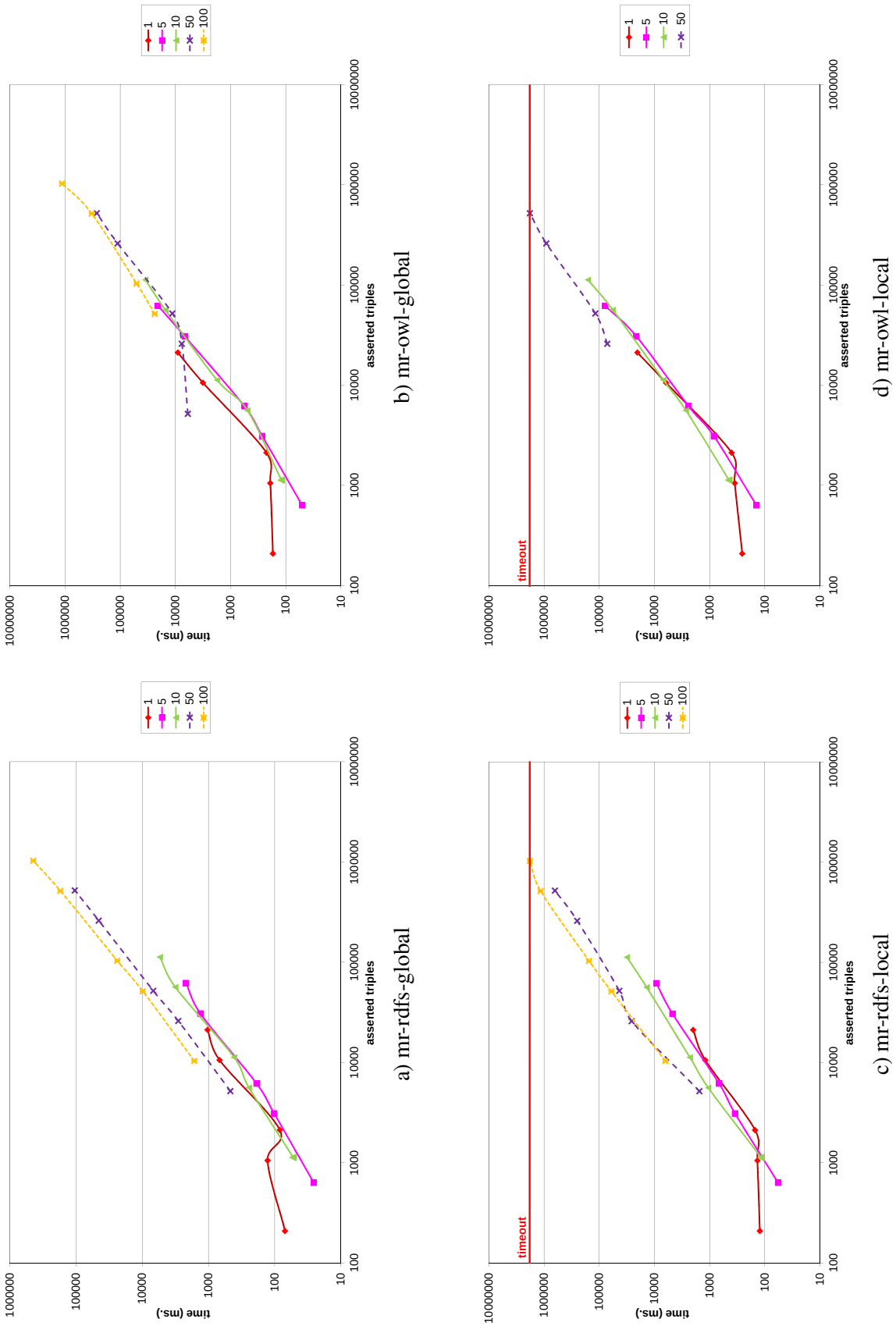

Figure 2.1: Scalability graphs for OWLIM.

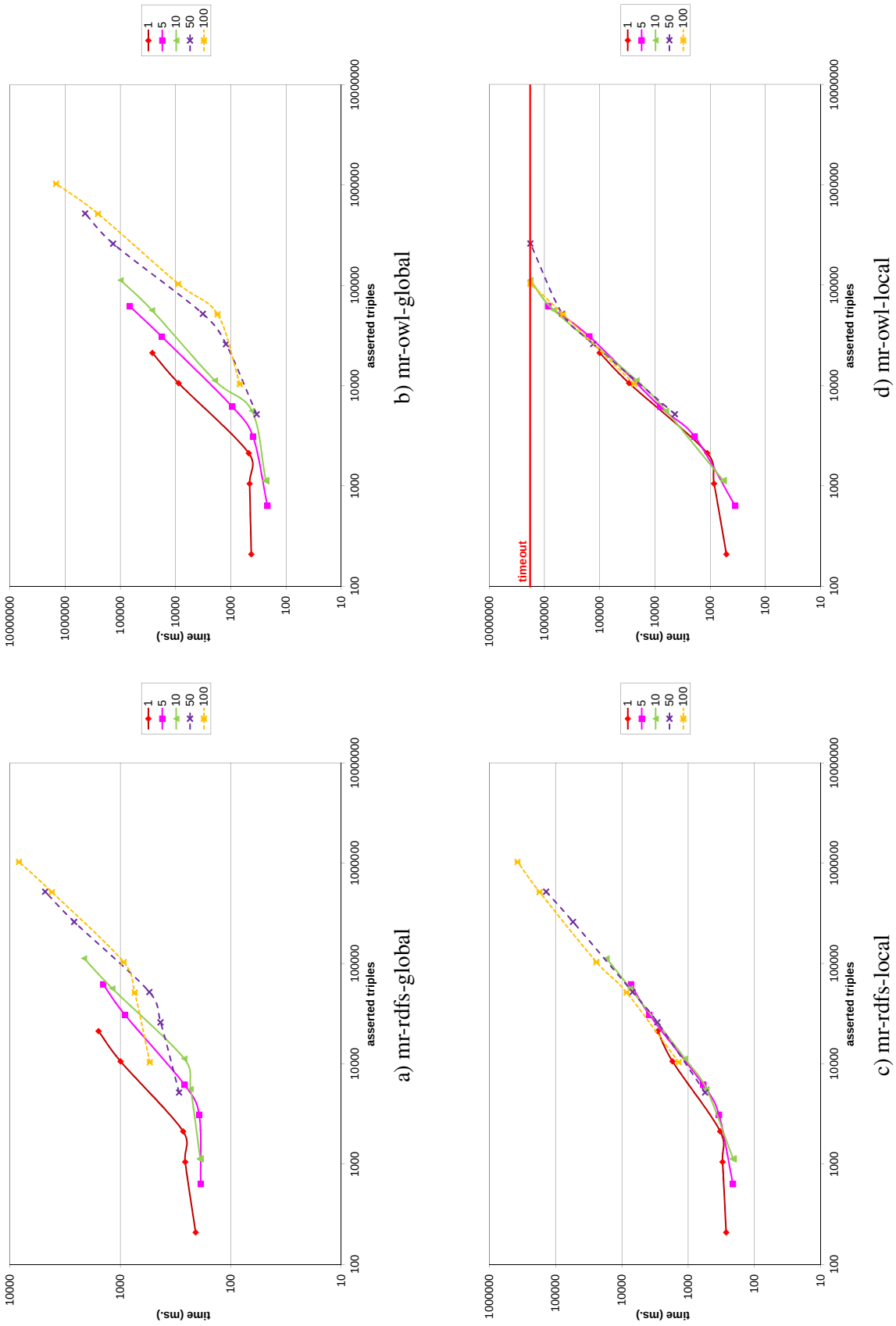

Figure 2.2: Scalability graphs for Sesame.

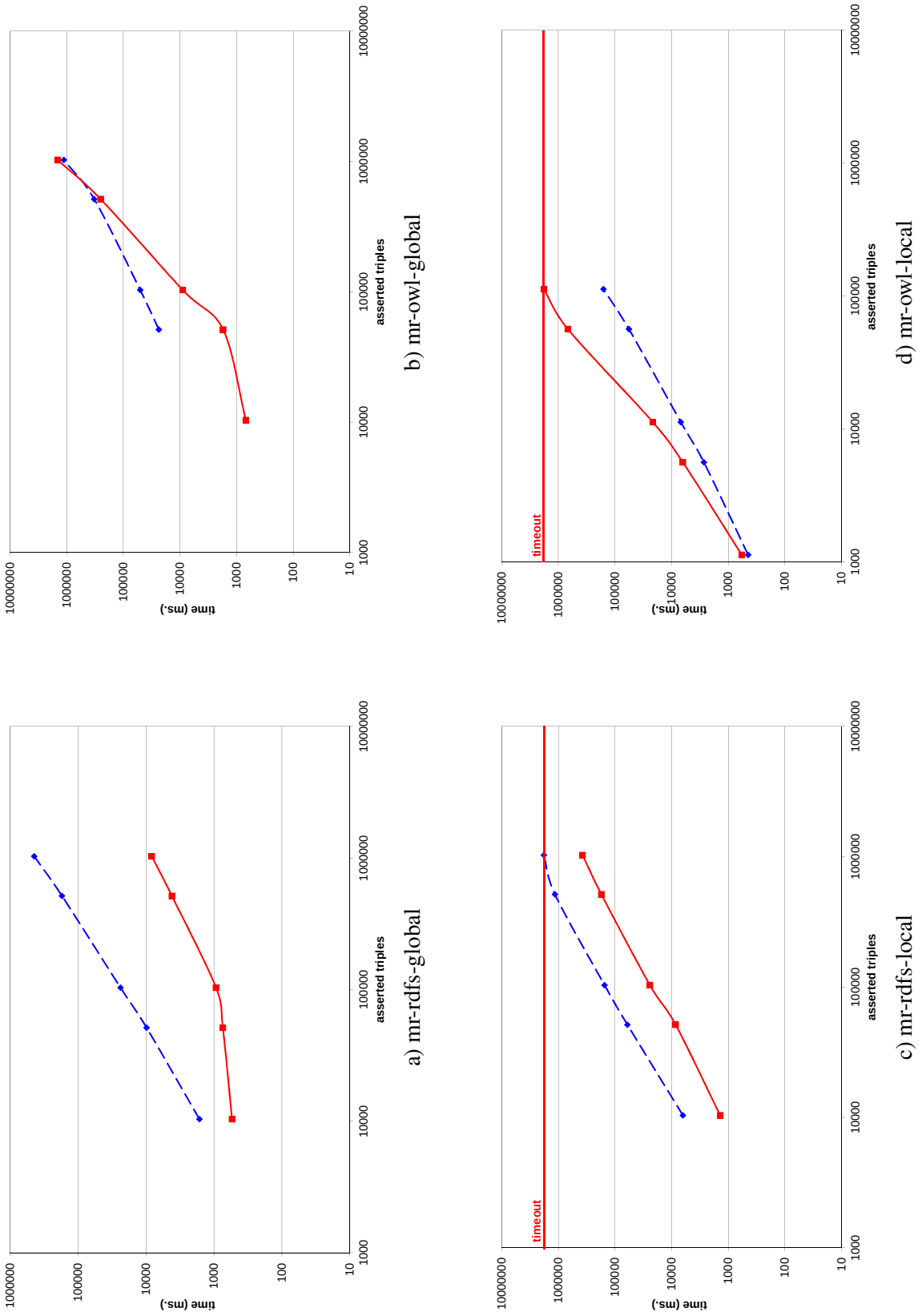

Figure 2.3: Comparison graphs for Sesame and OWLIM.

3 MetaReasons in modelling: OpenData Trentino use case

In this chapter we will show an application of the MetaReasons model and prototype to the modelling of a real and significantly sized repository of linked open data: the OpenData Trentino portal.

3.1 OpenData Trentino portal

OpenData Trentino[1] project (see e.g. [1, 6]) is an initiative born under the recent proposal of the Autonomous Province of Trento (PAT) to "open" to the public the data regarding information from the local government (and associated public entities).

The project is directed by a group of experts (coordinated by the PAT) that centrally monitors and supervises the release and publication of data provided by single local authorities. The project web portal offers as of September 2014 around 860 packages (set of datasets) coming from several departments of the PAT and local authorities. The data catalogue is built over the CKAN[2] data management system: packages are organized in around 20 categories and range from provincial statistics about population to transport information, cadastre, local weather and geographic data. Each dataset package has a dedicated page with links to datasets in different formats and a set of metadata about the package itself (e.g. creation date, tags, responsible organization). Currently the datasets are distributed mostly in CSV, JSON and XML formats: some of the datasets are also provided in RDF representation. All datasets are distributed under an open license offering a free and unlimited use and reuse of data.

3.2 A MetaReasons repository for OpenData Trentino

As motivated by the previous description of the OpenData Trentino initiative, the large set of information provided by the resources of the portal are certainly of interest for the development of (local) data-centered applications. Such large repository of open data, however, suffers from two problems that can hinder its adoption: the lack of a single and coherent format to distribute the data and the absence of a way to uniformly query and retrieve (part of) such data. Moreover, provenance of the datasets (for example, the entities involved in the creation and transformation of such data) has a central interest in the repository organization and, more in general, different combinations of datasets can be considered with respect to their metadata information. These aspects motivated us to model the contents of OpenData Trentino portal in a MetaReasons unified knowledge base and implement it over the current MR prototype.

In order to import under a single RDF repository the datasets of OpenData Trentino portal and manage their conversion to RDF, we implemented an import system, which can be depicted as in Figure 3.1.
The import procedure goes as follows: for each of the packages recognized on the OpenData Trentino portal, a software module downloads all of its datasets in the recognized formats (RDF, JSON, CSV and XML) and stores its metadata information in a local file. These files are stored in a local storage, accessible from the server running the MetaReasons webapp: it is thus possible to retrieve these files from remote applications, using their representation inside the MR model (as we detail below). If the datasets are already in RDF format, their contents are directly imported in their respective named graph in the RDF store. Otherwise, if they are represented in another format, a software module applies a mapping to transform the data in RDF. If a mapping file is not provided, the module is able to generate a basic mapping: since the data in JSON and CSV files provided in the portal are in tabular form, currently the generated mapping simply defines an individual for each of the rows and a datatype property for each of the columns. Both mapping files and the generated RDF files are stored to the local storage and the RDF file contents are then imported to the RDF store.

Currently, the repository counts a total of 2078 datasets, of which 157 imported from RDF files, 799 converted from JSON files and 1122 from CSV files: these corresponds to basically all of the datasets available from

[1] `http://dati.trentino.it/`
[2] `http://ckan.org`

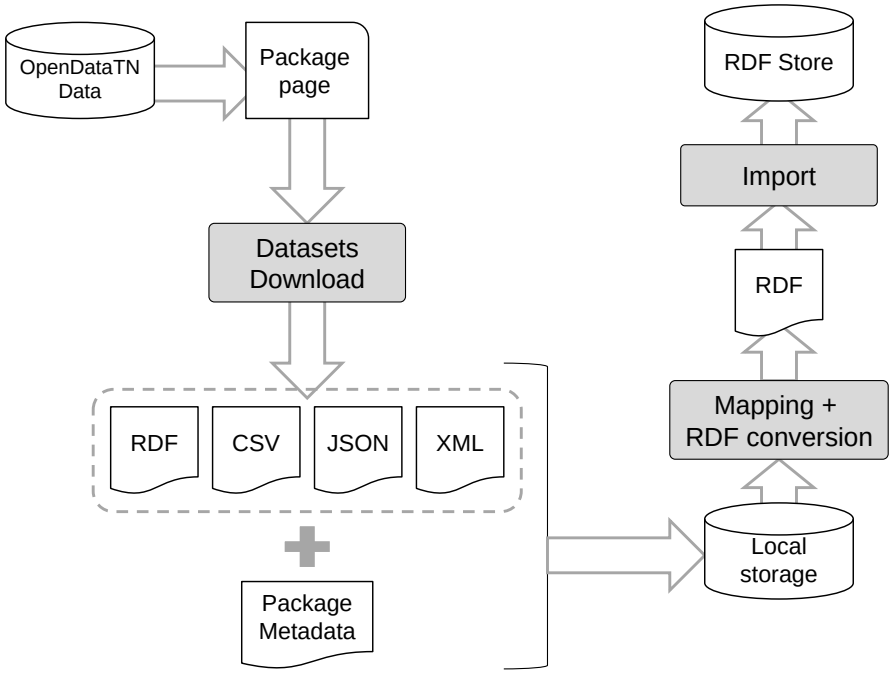

Figure 3.1: Import of OpenData Trentino datasets to RDF repository.

the portal in these formats. The total number of triples of the repository (including all the datasets and the metalevel) surpasses 26.000.000 triples[3]. Under the *mr-rdfs-global* inference ruleset, the inference materialization for the whole repository can be computed in around 147.000 ms (2,5 minutes).

The metalevel structure for the OpenData Trentino UKB is sketched in Figure 3.2.
Intuitively, the structure maps the objects of the portal to elements of the PROV-O ontology for provenance (corresponding to the MT_{hp} metatheory). Each `:Package` (sets of files representing a common set of data) is represented as a `prov:Collection`: different metadata (e.g. about creation date and tags) available from the package information are linked as literals by specific datatype properties we defined in our metalevel schema (e.g. `:hasCreationDate`, `:hasTag`). Every `:Dataset` (that is, single RDF files for a set of data, either imported or converted) is represented as `prov:Entity` and linked to its "parent" package by a `:hasMember` property (declared as a subproperty of `prov:hadMember`): following the MetaReasons architecture, every dataset has an associated named graph containing the RDF contents of the dataset file. In the case in which a dataset has been converted from a different format (currently, JSON or CSV), provenance information about this transformation is represented in the metalevel: the dataset individual is linked to the `prov:Activity` representing such `:MappingActivity` by the property `:wasGeneratedByMapping` (subproperty of the corresponding PROV-O property `prov:wasGeneratedBy`). The mapping activity individual is then linked to the original file, represented as instance of `:SourceFile`, by property `:usedSourceFile` and to the RDF file containing the mapping, represented as instance of `:MappingFile`, by property `:usedMapping`. Both `:SourceFile` and `:MappingFile` are subclasses of `prov:Entity`, while `:usedSourceFile` and `:usedMapping` are subproperties of `prov:used`. The URI we use as identifiers for the source files and mapping files correspond to the effective location of the file in our local storage, thus permitting to retrieve the original files used for the transformation.

In order to access the data inside the repository from external applications, we implemented a REST query interface, depicted in Figure 3.3. Intuitively, a set of predefined SPARQL 1.1 queries have been exposed as REST services (possibly with parameters). This allows external applications to access the contents of the repository by abstracting from the MetaReasons primitives and from the implementation as a SPARQL query (possibly spanning multiple named graphs).

[3]More in detail, we currently have 26.200.672 triples of which 21.901.621 asserted triples and 4.299.051 inferred statements.

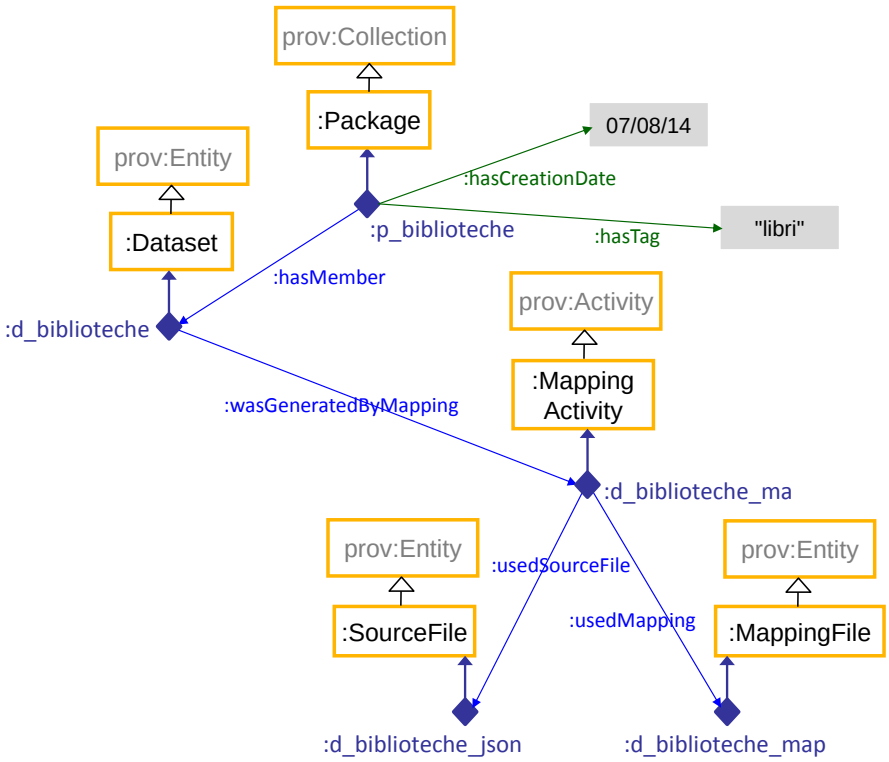

Figure 3.2: Metalevel structure for OpenData Trentino repository.

3.3 Unified queries examples

The set of unified queries we first implemented in the service interfaces include (i) a first set of general queries for the management of the provenance and metalevel information for packages and datasets and (ii) a second set of queries that have been defined to replicate the functionalities of some of the currently existing applications of datasets listed in the portal[4].

A simple example of the first set of queries is implemented in a *getPackageDatasets* service: given a package identifier in the metalevel, the request outputs all of the datasets of such package. The service is implemented with following the SPARQL query (in this case instantiated to a package about accessibility):

```
SELECT ?ds
WHERE {
  GRAPH meta:metakb {
  metakb:p_accessibilit-per-persone-con-disabilit-motorie metakb:hasMember ?ds }
}
```

This shows a query only involving the metalevel (provenance) information of datasets. On the other hand, unified queries can query specific datasets filtered by their metalevel properties: for example, this is the case of service *getPackageIndividualsByYear*, which, given a package URI and an year, returns all of the individuals inside the datasets of the package referring to that year. The service is implemented as the simple query:

```
SELECT ?ds ?uri
WHERE {
  GRAPH meta:metakb { metakb:p_abitazioni metakb:hasMember ?ds }
  GRAPH ?ds { ?uri metakb:anno ?year .
  FILTER ( ?year = 2001) }
}
```

Note that in this case the datasets to be queried are determined by the metalevel query about the package membership.

[4] http://dati.trentino.it/related

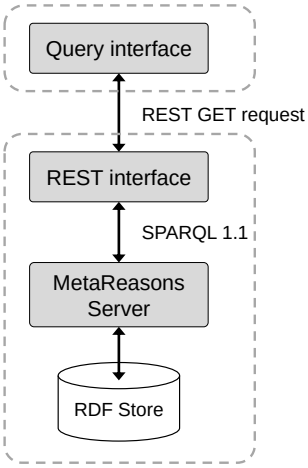

Figure 3.3: Query interface to OpenData Trentino repository.

The queries in the second set mostly connect different specific datasets and packages needed for an application: for example, in service *getComuni1* (related to the application *IET - Interfaccia Economico Territoriale*[5]) we join the codes of municipalities with the statistics about their population and request the cities with more than 5000 inhabitants in year 2009. This is implemented by the SPARQL query:

```
SELECT ?uri ?label ?abitanti
WHERE {
  GRAPH metakb:d_popolazione-residente-a-fine-anno_1_json
      { ?uri metakb:anno ?anno ;
               metakb:valore ?abitanti ;
               metakb:codEnte ?ente .
               FILTER ( (?anno = 2009) && (?abitanti >= 5000) ) }
  GRAPH metakb:d_codici-statistici-enti-locali_1_csv
        { ?uri2 metakb:Nome ?label ;
               metakb:Identificatore ?ente . }
}
```

We can show some complex examples of unified queries combining different datasets by considering datasets about public transport. Data about public transport is distributed in a single package (see `http://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs`) in different datasets files following the *General Transit Feed Specification (GTFS)*[6] format, both for city transport and extra-urban transport.

Using the public transport data, we can formulate queries over single and combined datasets. For example, the REST service *getNomiLineeViaggiSabato* returns all names for (extra-urban) lines and trips that are active on Saturdays. This is implemented by the following SPARQL query, that combines the datasets about lines, trips and service frequencies:

[5] `http://www.urbanistica.provincia.tn.it/sez_siat/Prog_Att_in_Sviluppo`
[6] `https://developers.google.com/transit/gtfs/`

```
SELECT ?rlname ?tripid
WHERE {
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_14_csv {
                    ?trip metakb:trip_id ?tripid ;
                          metakb:service_id ?serviceid ;
                          metakb:route_id ?routeid .}
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_9_csv {
                    ?service metakb:service_id ?serviceid ;
                             metakb:saturday "1"^^xsd:int .}
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_11_csv {
                    ?route metakb:route_id ?routeid ;
                           metakb:route_long_name ?rlname.}
}
```

We can give an example of query that combines most of the datasets of the package: given a specific bus stop (e.g. "Povo Piazza Manci"), find all stop times for a given line (e.g. line 13) after a given time (e.g. 19:00). This is exemplified by service *getOrariLinea13*, implemented as the following SPARQL query:

```
SELECT ?dtime
WHERE {
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_6_csv {
                    ?stop metakb:stop_name ?sname ;
                          metakb:stop_id ?sid .
                    FILTER(contains(?sname,"Povo Piazza Manci")) }
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_5_csv {
                    ?stime metakb:stop_id ?sid ;
                           metakb:trip_id ?tid ;
                           metakb:departure_time ?dtime .
                    FILTER(?dtime >= "19:00:00") }
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_7_csv {
                    ?trip metakb:trip_id ?tid ;
                          metakb:route_id ?rid . }
  GRAPH metakb:d_trasporti-pubblici-del-trentino-formato-gtfs_4_csv {
                    ?route metakb:route_id ?rid ;
                           metakb:route_short_name "13" . }
}
```

4 Conclusions

In this deliverable we presented the results of the evaluation activity for the MetaReasons framework prototype. We first studied the scalability of the current implementation by presenting a set of experiments we carried out over synthetic UKBs: the reasoning procedure has been evaluated with respect to different reasoning regimes (RDFS or OWL RL, with and without object level reasoning) and different solutions for RDF storage (Sesame Native or OWLIM storage). Then we presented an application of MetaReasons model and prototype in a real usecase, the OpenData Trentino repository, that demonstrates the applicability of the model in the combination of metadata on different datasets.

The two aspects presented in this document show both benefits and limits of our model and its current prototype: while the naive application of the complete reasoning to the full UKB can be limiting in terms of scalability, we also shown a use case in which the model can be effectively applied in practice to reason and query over a real and large set of data. Also, scalability experiments have been carried out on realistically shaped UKBs but with general contents (e.g. not tailored to a specific metatheory and without restrictions on the contents of datasets). On the other hand, the real-world use case shown that knowing the form of the specific UKB allows to adapt the reasoning mechanism (e.g. exclude unnecessary rule evaluations) and better control the actual scalability of the framework.

Bibliography

[1] Ivan Bedini, Feroz Farazi, Juan Pane, Ivan Tankoyeu, David Leoni, and Stefano Leucci. Open government data: Fostering innovation. In *Share-PSI 2.0*, 2014.

[2] Loris Bozzato and Luciano Serafini. D31.1 MetaReasons: theoretical architecture description. Deliverable D31.1, PlanetData, January 2014. http://planet-data-wiki.sti2.at/web/D31.1.

[3] Loris Bozzato and Luciano Serafini. D31.2 MetaReasons: metatheories list and description. Deliverable D31.2, PlanetData, March 2014. http://planet-data-wiki.sti2.at/web/D31.2.

[4] Loris Bozzato and Luciano Serafini. D32.1b MetaReasons: system prototype (version 2, final). Deliverable D32.1b, PlanetData, August 2014. http://planet-data-wiki.sti2.at/web/D32.1b.

[5] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

[6] Pavel Shvaiko, Feroz Farazi, Vincenzo Maltese, Alexander Ivanyukovich, Veronica Rizzi, Daniela Ferrari, and Giuliana Ucelli. Trentino government linked open geo-data: A case study. In *ISWC 2012*, pages 196–211, 2012.