

A Mashup Personalization Service based on Semantic Web Rules and Linked Data

Georgia D. Solomou, Aikaterini K. Kalou, Dimitrios A. Koutsomitropoulos and Theodore S. Papatheodorou,
Member, ACM, IEEE

HPCLab, Computer Engineering and Informatics Dpt. University of Patras
Patras, Greece

{solomou, kaloukat, kotsomit,tsp}@hpclab.ceid.upatras.gr

Abstract—In this paper we propose an intelligent personalization service, built upon the idea of combining Linked Data with Semantic Web rules. This service is mashing up information from different bookstores, and suggests users with personalized data according to their preferences. This information as well as the personalization rules are then processed and managed by a scalable knowledge repository. Finally, they are made available as Linked Data, thus enabling third-party recipients to consume knowledge-enhanced information.

Keywords—Linked Data; ontologies; Semantic Web rules; triple stores; mashups

I. INTRODUCTION

The great proliferation of Linked Data paves the way for the deployment of robust applications, able to consume large datasets in a more effective way [2]. However, their limited semantics often lead to applications with poor knowledge discovery capabilities [7, 8]. On the other hand, the use of full-fledged ontologies comes with an additional overhead, especially when inferencing is required.

In an attempt to achieve a trade-off between powerful reasoning and scalability, we propose a web application that combines the basic principles of Semantic Web rules and Web 2.0 mashups [5], aiming to provide an intelligent and scalable service for personalized querying over popular online bookstores. Our application (‘Books@HPCLab’) provides users with the ability to search and find sell-offers for books that fit their preferences. These data are mashed up from different and heterogeneous sources on the Web, like Amazon and Half Ebay. The descriptive metadata of retrieved books are finally triplified, thus producing a linking connection to their offers. All collected information is stored in a scalable rule-based triple store (OWLIM [1]).

Gathered information is mapped to an OWL ontology that has been developed in order to describe users, books and offers (the ‘BookShop’ ontology). The BookShop ontology is kept in the triple store together with a number of rules, which reflect user preferences. This ontology- and rule- oriented approach renders the gathered information reusable and sharable. At the same time, we expose and make it available to the Linked and Open Data (LOD) world, through an appropriate interface.

Books@HPCLab application is a significantly enhanced version of a previous one presented at [10]. This new application actually serves as a proof of concept about the successful and efficient combination of Semantic Web, Linked Data and rules in designing intelligent mashups.

The rest of this paper is organized as follows: Section 2 gives an overview of the overall application architecture and component interaction. Then, Section 3 puts focus on the BookShop ontology and the rules used for personalization. The major architectural components are discussed in Section 4, where the data aggregation and triplication procedures are explained. Section 5 outlines the functionality of the application, which is then evaluated in section 6, by performing several timings in comparison to its earlier version. Conclusions and future work are presented in the last section 7.

II. ARCHITECTURAL OVERVIEW

In this section, we are sketching out the system’s architecture. We present its structural components, designated to carry through the mashing and linking functionality that is finally provided to the end users. We also describe in what sense we take advantage of Semantic Web technologies.

The full functionality of Books@HPCLab application is carried out by several distinct components that interact with a central management mechanism, the ‘Core Unit’. An overview of the system’s architecture is given in Fig. 1. The Core Unit implements the application logic of the service and is responsible for communicating with the ‘Mashup Component’, the ‘Triple Store’, the ‘User Interface (UI)’ and finally with the ‘Linked Data Component’, which carries out the proper exposure of retrieved information as Linked Data.

To implement communication and resource interchange among the central Core Unit and the other components, we exploit web services, programs, scripts, and other Web technologies. The implemented architecture is based on a REST-ful communication strategy, where data transfer is implemented via HTTP requests. The semantic processing of gathered information is accomplished by designated Semantic Web technologies, like ontologies, rules, inference engines and a triple store mechanism.

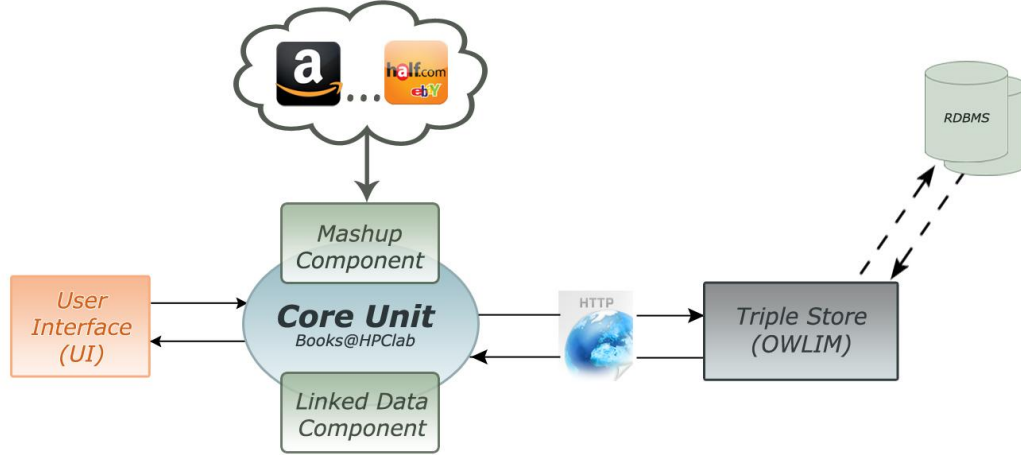


Figure 1. Architecture overview.

The UI component is the application front-end that provides users with all necessary facilities for mashing information that is in conformance with their preferences. By interacting with the UI, a user can create his profile, submit a search request and finally view and access appropriate results – available also as Linked Data through the Linked Data component. The user’s request, consisting of search keywords, is passed on to the Core Unit which in turn assigns it to the corresponding components.

The Mashup Component is responsible for establishing communication and interaction with the Amazon and Half EBay Web APIs. It aggregates data by mashing the user requests. Harvested data are then transformed to meaningful RDF descriptions and get stored at the triple store. The storage mechanism is built upon OWLIM [1], which benefits from the scalable architecture and other features of the Sesame framework [3]. Data loading is accomplished through appropriate HTTP requests.

The triple store keeps the ontology schema and several rules, which reflect user preferences for personalization. The population of data as well as the reasoning process is performed on loading. The stored rules are fired immediately and produce a number of inferred statements. Results (explicit and inferred) become available as RDF data and can be accessed via HTTP, by making SPARQL queries.

III. ONTOLOGIES AND RULES

Here we describe our application’s core ontology, as well as rules that come to augment our service with personalization capabilities. We first give an overview of the ontology’s main characteristics and proceed with a brief explanation about our custom rule set.

A. The BookShop Ontology

To design our book ontology, we took into account the kind of metadata offered by Amazon and Half Ebay responses. Our design process has resulted in the core

ontology *BookShop*. BookShop contains four main classes: *Book*, *Author*, *Offer* and *User*. Overall, ontology expressivity complies with OWL Horst [11], with the exception of disjoint classes that are only partially supported under pD^* -entailment.

The class *Book* is enriched with relations about title, publisher, dimensions, ISBN, publication year, number of pages, format, rating, images in various sizes and a URL – corresponding to the Amazon online bookstore – so as to describe the instances of this class. All these relations are captured in the ontology as datatype properties.

Items for sale on Amazon and Half Ebay can be sold by more than one seller for different prices and in different conditions (‘New’ or ‘Used’). Thus, any item – any book in this case – is associated with an offer. An offer is a combination of price, condition and vendor/seller. Therefore, to find the price for a book, we have to get all the offers made by vendors selling this book on online bookstores. The concept of an offer is represented by the class *Offer*. Datatype properties such as *Condition*, *Price* and *Origin* express the price, the condition of the offered book and the seller URL in these bookstores, respectively.

The class *User* is meant to express user profiles. We capture the preferences of each user in this class, such as preferred condition, preferred rating, preferred publication year and preferred maximum price (preference criteria). All this data about users are represented as datatype properties.

Relationships between instances of the BookShop ontology are represented by object properties that express relations between instances of two classes. In this context, a *Book* must have at least one *Author* (*hasAuthor* and inversely *isAuthorOf*) and there is at least one *Offer* for a *Book* (*isOfferOf* and inversely *hasOffer*).

Object properties that link an instance of the class *Book* to an instance of the class *User* and inversely, are defined to express the user’s preference for a book depending on which preference fields of the user’s profile are covered. Then, a

set of rules is responsible for actually populating these properties. For example, the object property *prefersBookbyCondition* would relate a user with books being in a condition the user prefers. There are also properties for the case when more than one preference criteria are met. For example, *prefersBook_byRate2* would hold in case exactly two criteria are satisfied. It is natural that whenever a *prefersBookbyX* relationship holds (with *X* being condition, rating, etc), then *prefersBook_byRate1* would also be true for this particular user-book pair. Thus we define all *prefersBookbyX* properties as sub-properties of *prefersBook_byRate1*.

B. Personalization Rules

Rules follow the OWLIM's inherently supported rule formalism. Based on what rules dictate, a second matching process is performed upon retrieved results, taking into account user preferences (user profile). These "personalization rules" actually act as a filter to the result set, being able to distinguish among those books that satisfy user's preferences and those that are irrelevant to the user's profile.

TABLE I. EXAMPLE RULES THAT WERE CREATED FOR MATCHING USERS' PREFERENCES.

No #	Rule Body	Rule Head
1	IF <i>u</i> <i>prefersCondition</i> <i>z</i> AND <i>b</i> <i>hasOffer</i> <i>o</i> AND <i>o</i> <i>hasBookCondition</i> <i>z</i>	<i>u</i> <i>prefersBookbyCondition</i> <i>b</i>
2	IF <i>u</i> <i>prefersMaxPrice</i> <i>z</i> AND <i>b</i> <i>hasOffer</i> <i>o</i> AND <i>o</i> <i>offerPrice</i> <i>z</i>	<i>u</i> <i>prefersBookbyPrice</i> <i>b</i>
3	IF <i>u</i> <i>prefersRating</i> <i>z</i> AND <i>b</i> <i>hasRating</i> <i>z</i>	<i>u</i> <i>prefersBookbyRating</i> <i>b</i>
4	IF <i>u</i> <i>prefersPublicationDate</i> <i>z</i> AND <i>b</i> <i>PublicationDate</i> <i>z</i>	<i>u</i> <i>prefersBookbyPublicationDate</i> <i>b</i>
5	IF <i>u</i> <i>p</i> <i>b</i> AND <i>u</i> <i>q</i> <i>b</i> AND <i>p</i> , <i>q</i> <i>subPropertyOf</i> <i>prefersBook_ByRate1</i> AND <i>p</i> , <i>q</i> <i>!=</i> <i>prefersBook_ByRate1</i> AND <i>p</i> <i>!=</i> <i>q</i>	<i>u</i> <i>prefersBook_byRate2</i> <i>b</i>

As a starting point, four rules were written in order to check the satisfiability of each preference criterion separately (the 'rule head' in Table I). Take for example the case of rule 1: It expresses the desired effect when correlating a certain book condition with a user's particular preference about this characteristic. Along these lines, rules 2-4 match books that satisfy the remaining three user preferences.

For the case where two, three or more preference criteria are satisfied together, we wrote more rules so as to check the exact number of satisfied criteria. Rule 5 is such an example, for the case a book matches two criteria. Recall

that *prefersBook_byRate1* is a catch-all for rule heads 1-4. Then this rule can be read as follows: "match all user-book pairs that are related with two rule preferences (*p* and *q*), but make sure that these preferences are unique, i.e. different from each other". Another couple of rules have been devised in a likewise fashion, so as to check all possible combinations.

IV. DATA AGGREGATION AND SEMANTIC STORAGE

A. Mashup and Linked Data Components

The Mashup component is the one that carries out the communication with Amazon and Half Ebay services, by interacting with their respective Web APIs. Whenever the user sends a *searching call*, the searching process starts to query data from Amazon Web Services (AWS), and especially from the *US E-Commerce Service (ECS)*. In order to extract the appropriate data for our application, we choose the *ItemSearch* operation, among the set of available ECS operations.

Once our application completes the search process at Amazon, it starts searching Half Ebay: for each book returned by Amazon, we find additional offers that may be available at Half Ebay. We use the *eBay Shopping Web Services* and particularly, the *FindHalfProducts* operation. The interaction with the eBay Shopping API is based also on the REST-protocol and the exchange of URL requests and XML files-responses.

```
<Item>
  <ASIN>156484272X</ASIN>
  <DetailPageURL>http://www.amazon.com/Web-2-0-How-Gwen-Solomon/dp/
    156484272X%3FSubscriptionId%3
    D05QEM4HDNYGR2EDE91R2%26tag%3Dws%26linkCode%3Dxm2%26camp
    %3D2025%26creative%3D165953%26creativeASIN%3D156484272X
  </DetailPageURL>
  <ItemAttributes>
    <Author>Gwen Solomon</Author>
    <ISBN>9781564842725</ISBN>
    <Title>Web 2.0: How-To for Educators</Title>
    ...
  </ItemAttributes>
  <Offers>
    <Offer>
      <Merchant>
        <GlancePage>http://www.amazon.com/gp/help/seller/
          home.html?seller=ASWK2P83YYFWO
        </GlancePage>
      </Merchant>
      <OfferAttributes>
        <Condition>Used</Condition>
      </OfferAttributes>
      <OfferListing>
        <Price>
          <FormattedPrice>$20.27</FormattedPrice>
        </Price>
      </OfferListing>
    </Offer>
    ...
  </Offers>
</Item>
```

Figure 2. Sample Amazon response.

```

<FindHalfProductsResponse xmlns="urn:ebay:apis:eBLBaseComponents">
  <Products>
    <Product>
      <Title>Web 2.0: How-to for Educators by Lynne Schrum and Gwen Solomon
        (2010, Paperback)
      </Title>
      <DetailsURL>http://syicatalogs.ebay.com/ws/
        eBayISAPL.dll?PageSyiProductDetails&IncludeAttributes=1&Show
        AttributesTable=1&ProductMementoString=111322:2:1055:3768915169
        :349003979:cd998ecc605cf041c490bb9931398786:1:1:1:1411329640
      </DetailsURL>
      <ProductID type="ISBN">156484272X</ProductID>
      <ProductID type="ISBN">9781564842725</ProductID>
      ...
    <ItemArray>
      <Item>
        <ViewItemURLForNaturalSearch>http://product.half.ebay.com/Web-2-0-
          by-Gwen-Solomon-Lynne-Schrom-2010-Paperback-Gwen-Solomon-
          Lynne-Schrom-Paperback-2010_W0QQPrZ 92445819QQt
          gZvidetailsQQitemZ342138964277
        </ViewItemURLForNaturalSearch>
        <CurrentPrice currencyID="USD">24.03</CurrentPrice>
        <HalfItemCondition>BrandNew</HalfItemCondition>
        ...
      </Item>
    </ItemArray>
  </Products>
</FindHalfProductsResponse>

```

Figure 3. Sample Half Ebay Response.

The application uses the RDF API for PHP [4] in order to process aggregated data, to assign resolvable identifiers and ultimately to convert them to Linked Data in RDF format. This conversion happens in both directions, i.e. when the data are collected as well as when a user requests RDF data in the book view page (see next section). In the latter case however, information is already available in the triple store and are fetched directly, with only a minimal intervention to tide them up. Fig. 2 and Fig. 3 present typical sample responses from Amazon and Half Ebay respectively. Fig. 4 shows how these responses are mapped to the BookShop ontology and translated in to RDF. Note finally that we assign book identifiers such that they can be resolved within our application (see next section).

B. Semantic Storage

All information gathered by the Amazon and Half Ebay web service - after a necessary transformation phase that ends up with its RDF representation - is stored in the application's triple store in the form of RDF triples. The repository is built upon OWLIM Lite v. 4.0 (previously *SwiftOWLIM*), which implements the Sesame SAIL interface, so that it can exploit all Sesame components. As far as reasoning is concerned, this is based on R-entailment defined by ter Horst [11], where inference rules are applied directly to RDF triples. OWLIM is configurable as to what set of inference rules may support, thus choosing the level of supported semantics. Each rule is built of premises (body) and conclusions (head). Both premises and conclusions are RDF statements, where the use of variables is allowable at any position.

An important feature of the recent version of Sesame (Sesame v. 2.4.0) is its compliance with the latest version of SPARQL query language, namely SPARQL 1.1 [6]. As a consequence, OWLIM becomes able of supporting SPARQL 1.1 query functionality directly through the Sesame APIs. So, accessing and retrieving the data stored in the triple store is as easy as evaluating SPARQL 1.1 queries. Given that Sesame uses an HTTP-based communication protocol, the aforementioned functionality is also feasible via the standard HTTP methods (GET, POST, PUT, etc). Sesame can also be coupled with a relational database, thus providing for fast indexing and evaluation of queries.

All RDF triples that answer a user's specific search request are stored in the same OWLIM repository. Nevertheless, the underlying Sesame architecture allows for storing triples in the form of 'quads' where an additional 'context' information piece can be attached. Context declares the provenance of data and is useful in order to distinguish among different users' data.

```

<BookShop:Book rdf:about="http://levantes.hpcLab.ceid.upatras.gr:8000/BooksHPCLab/book/156484272X">
  <BookShop:Title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Web 2.0: How-To for Educators</BookShop:Title>
  <BookShop:DetailPageURL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    http://www.amazon.com/Web-2-0-How-Gwen-Solomon/dp/156484272X%3FSubscriptionId%3D05QEM4HDNYGR2EDE91R2%
    26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D156484272X
  </BookShop:DetailPageURL>
  <BookShop:ISBN-10 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">9781564842725 </BookShop:ISBN-10>
  ...
  <BookShop:hasAuthor>
    <rdf:Description rdf:about="#Author_1_156484272X">
      <foaf:name rdf:datatype="http://www.w3.org/2001/XMLSchema#Literal">Gwe</foaf:name>
      <foaf:surname rdf:datatype="http://www.w3.org/2001/XMLSchema#Literal">Solomon</foaf:surname>
    </rdf:Description>
  </BookShop:hasAuthor>
  <BookShop:hasOffer>
    <rdf:Description rdf:about="http://www.amazon.com/gp/help/seller/home.html?seller=ASWK2P83YYFWO">
      <BookShop:BookCondition rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Used</BookShop:BookCondition>
      <BookShop:OfferPrice rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal"> 20.27</BookShop:OfferPrice>
    </rdf:Description>
  </BookShop:hasOffer>
  <BookShop:hasOffer>
    <rdf:Description rdf:about="http://product.half.ebay.com/Web-2-0-by-Gwen-Solomon-Lynne-Schrom-
      2010-Paperback-Gwen-Solomon-Lynne-Schrom-Paperback2010_W0QQPrZ92445819QQtQgZvidetailsQ QitemZ342138964277">
      <BookShop:BookCondition rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> New</BookShop:BookCondition>
      <BookShop:OfferPrice rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal"> 24.03</BookShop:OfferPrice>
    </rdf:Description>
  </BookShop:hasOffer>
</BookShop:Book>

```

Figure 4. Data triplification and mapping to the BookShop ontology.

Welcome User_16

Search Settings Logout

Favourite Books

ID	Book's title	Score
1	Java Programming: From the Beginning	!!!
2	Schaum's Outline of Programming with Java	!!
3	SCJA Sun Certified Java Associate Study Guide (Exam CX-310-019) (Certification Press)	!!
4	SCJP Sun Certified Programmer for Java 6 Exam 310-065	!!
5	Java 7 The Complete Reference, 8th Edition	!!
6	Java 7, A Beginner's Guide, 5th Edition	!!
7	Java 7 Programming	!!
8	Learn to Program with Java	!!
9	Java 2: A Beginner's Guide	!!
10	Java Demystified	!!

Previous Page Page to 10 Next Page

Figure 5. Data triplification and mapping to the BookShop ontology.

V. FUNCTIONALITY

The first page of the application includes two choices for the visitor, “*New User*” and “*Registered User*”. New users are presented with a completion form, which includes fields such as *Book Condition*, *Maximum Book Price*, *Publication Year* and *Maximum Book Rating*, thus forming the user profile. After successful authorization, the user is directed to the main page of the application, which includes a search form. It consists only of a “*Search*” button and a text field, where the user types the keyword or the key-phrase to initiate the book searching process.

During searching, results are imported into the triple store and the rules, that would determine how user preferences are matched, are fired against the data. Search results are ranked based on the rules outcome, as shown in Fig. 5. In

particular, the more criteria a Book satisfies (the more preference rules it triggers), the higher it appears in the results.

Each table's row includes information, such as an auto-incrementing number, the book's title and a number of exclamation marks which express the number of satisfied criteria. Each book's title is a link and clicking it, a page appears with all the available features of the specific book like Title, Author, ISBN (Fig. 6). The title and image of the book are links to its “official” page at Amazon. The Offers for this book are also presented in a table. Following the *Search* link (top right of Fig. 5) the user is taken back to the main page in order to initiate a new query.

Finally, book information is made available as Linked Data in RDF, where books' and offers' links are resolvable within our application or within their original context,







 <p>Java Programming: From the Beginning</p> <p>Book's Details</p> <ul style="list-style-type: none"> - Author : K. N. King - ISBN : 0393974375 - Publisher : W. W. Norton & Company - Publication Year : 2000 - Format : Paperback - Number of Pages : 788 - Transport Weight : 273 Pounds - Book's Dimensions : 94 × 913 × 748 <p>RDF</p>	ID	Condition	Price	Web Origin
	1	New	65.0	
	2	New	145.18	
	3	New	150.57	
	4	New	74.37	
	5	Used	73.82	

Figure 6. Book Information.



Figure 7. Data triplification and mapping to the BookShop ontology.

respectively (Fig. 7). This is also true for the search results list, which is exposed as RDF along with the inferred book rankings, for other applications to be able to consume and reuse.

VI. EVALUATION

In order to identify the major points made by the design of this mashup, we evaluate it against the previous version of Books@HPCLab, originally introduced in [10]. In summary, we notice a great improvement in query response time, which is mostly due to the utilization of a specialized triple store, as well as an overall performance improvement due to additional optimizations introduced.

First, we describe the methodology of the experiments, as well as their configuration. To clarify the comparison, we also summarize the different features between the two implementations. Then we present the results for some typical queries made with each of the two applications respectively.

A. Methodology and Configuration

There have been measurements about the original mashup application (*Books@HPCLab*) and the new implementation (*Books@HPCLab_revised*). All Books@HPCLab timings were made on a machine with a P4 HT processor, running at 3.0 Ghz and with 2.0 GB of RAM. Measurements about the revised implementation involve also a second machine hosting the triple store: Intel Core 2 6300@1.86 Ghz with 2.0 GB of RAM. However, the former computer is responsible for the mashup creation (collecting and triplifying Amazon and eBay data) in both cases.

We measure the performance of both applications in respect to four queries posed by the same user (i.e. same personalization preferences). We distinguish between *load time*, which is how long it takes to fetch, triplify and store mashed-up data and *reasoning time* that is, the time it takes to evaluate the personalization rules against the data set and get back the results. Full-text query evaluation is made independently by the Amazon API and therefore is included in the load time.

Results are comprised of the books that actually match at least one preference criterion. In principle, one would expect these results to be identical in both cases; however there may be slight differences in result count due to the fact that some eBay offers may have been added or withdrawn in the course of time between the two experiments.

Table II sums up the main differences between the two implementations. *Optimized* means that we use a different algorithm to fetch the data from their sources resulting in fewer requests, which are actually responsible for most part of the load time.

TABLE II. FEATURE COMPARISON FOR THE TWO IMPLEMENTATIONS.

	Books@HPCLab	Books@HPCLab_revised
(1) Mashup Creation	Web APIs	Web APIs (optimized)
(2) Dataset storage / management	File System	Triple Store (OWLim)
(3) Rule-based reasoning	SWRL, Pellet	Proprietary, OWLim
(4) Communication protocol	fopen, fwrite, OWLAPI	HTTP REST, SPARQL

B. Results

Below (Table III) are the results about Books@HPCLab. First column includes the query strings used to mash up the data from Amazon and eBay. To give an insight of the reasoning load, we also give the number of total triples found in the data. The number of results includes only preference-matching books. All timings are given in seconds.

TABLE III. BOOKS@HPCLAB PERFORMANCE.

Query	Load Time	Reasoning Time	#triples	#results
Web 2.0	135	76	14808	96
Web 2.0 applications	139	85	15551	100
Programming in java	139	87	18959	100
Java	141	71	19808	97

The high load times observed are due to the increased number of requests made to the Web APIs: Amazon and eBay only allow paged fetch of their results, which enforces us to make multiple successive requests, each taking about one second. The poor reasoning performance is indicative of the inability of current DL-based reasoners to scale over a large number of triples, especially when rules are involved. In Table IV, we examine the measurements about Books@HPCLab_revised.

In this case, load times are found somewhat reduced (Fig. 8): indeed, we now make fewer calls to the eBay API, by asking for multiple book offers in one request, thus resulting in fewer costly requests overall. The huge performance improvement is however in reasoning (Fig. 9). We notice a reduction in reasoning time by almost an order of magnitude, which cannot be justified by the different hardware: Clearly SPARQL queries are rapid when compared to the DL-based classification and the forward chaining algorithm of the triple store pays off.

TABLE IV. BOOKS@HPCLAB_REVISIED PERFORMANCE.

Query	Load Time	Reasoning Time	#triples	#results
Web 2.0	80	12	69576	95
Web 2.0 applications	83	7	69576	99
Programming in java	108	9	69576	90
Java	113	16	69576	88

In fact, the actual reasoning time for Books@HPCLab_revised has been included in the load time: Each time a new query dataset gets added to the store all the rules are fired and new triples are computed, based on total materialization. Besides, the most part of it includes the overhead to transfer the dataset to the store over HTTP. In contrast, Books@HPCLab loads the ontologies directly from the disk. Therefore, ‘reasoning time’ would simply amount to the evaluation of the SPARQL query and results fetch.

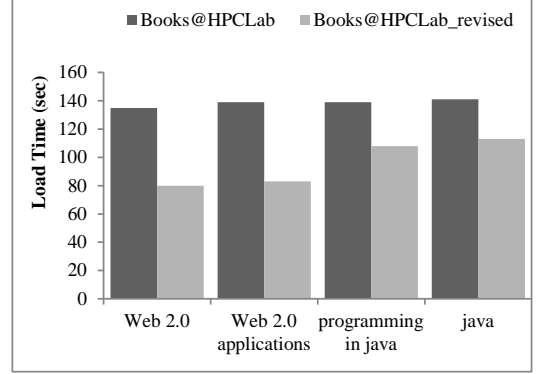


Figure 8. Load time comparison per query.

Notice also that the number of triples for each query has increased. This is because we now reason over the whole dataset, i.e. the sum of triples produced by every query. To this, one should add the 450 triples found in the schema ontology (BookShop) that are loaded in the repository during initialization. Still, even though the triple count has quadrupled, the new implementation hugely outperforms its predecessor in terms of reasoning performance. To the DL-based reasoner’s advocacy we have to add OWLIm’s inability to reason about datatypes. The latter can be easily integrated in SWRL with the use of built-ins [9]. Note also the fact that the BookShop ontology falls into the ‘OWL Horst’ dialect, a proper subset of OWL 2-RL and only a small subset of OWL 2-DL, which the DL reasoner is able to provide sound and complete reasoning for.

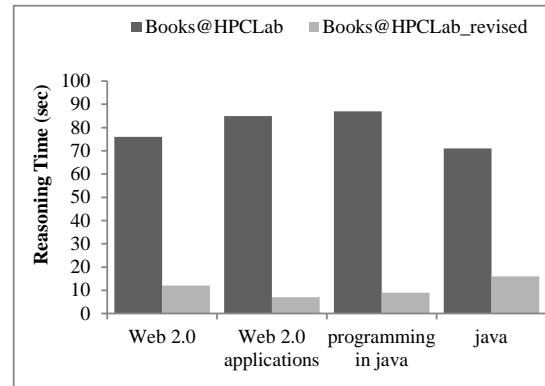


Figure 9. Reasoning time comparison per query.

VII. CONCLUSIONS AND FUTURE WORK

One of the greatest features Semantic Web has to offer is undoubtedly reasoning-based querying. Linked Data is another important dimension which enables opening, linking and sharing information in machine understandable formats. However, it often appears to disregard the strong semantic underpinnings the Semantic Web comes with; and this due to the high complexity and stiff scalability of reasoning, especially for expressive languages. In this paper we have shown that a reasonable compromise can be

achieved by relying on rule-based reasoning. By combining linked data and rules we can produce meaningful added-value, oriented towards user recommendation, with controlled and scalable performance costs.

At the same time, we have set up a prototype that can bootstrap semantic personalization services with Linked Data coming from legacy contexts. Our semantic mashup actually triplifies and links data from different sources that can be independently consumed later by other applications. In addition, it can act as an independent framework for efficient management of rule-based ontologies; the component design is oblivious to the underlying business logic (e.g. Java, servlets) or the front-end implementation (e.g. PHP).

Having eliminated reasoning times, mashup creation requests are mainly responsible for lowering overall performance. An obvious improvement would be the asynchronous processing of these requests and loading the datasets to the triple store little-by-little or caching, in case of repeated queries, since datasets are already persistent. More succinct data formats may also be necessary, since RDF/XML appears quite verbose and can put a considerable communication overhead. To this end, JSON (<http://json.org>, <http://json-ld.org>) can be an alternative for data serialization. Finally, the use of datatype operators inside rules comes in handy, although it can be mimicked by SPARQL queries. To this direction the consideration of triple stores like OWLIM to express rules in SPARQL-like format appears promising.

REFERENCES

- [1] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "OWLIM: A Family of Scalable Semantic Repositories", *Semantic Web Journal*, vol. 2(1), pp. 33—42, 2011
- [2] C. Bizer, T. Heath and T. Berners-Lee, "Linked Data - The Story So Far", *International Journal on Semantic Web and Information Systems*, vol. 5(3), pp 1—22, 2009
- [3] J. Broekstra, A. Kampman and F. van Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema", *Proc. of 1st International Semantic Web Conference*, LNCS, vol. 2342, pp 54—68, 2002
- [4] P. Cowles, "Exposing Web Applications Semantically Using RAP (RDF API for PHP)", *PHP Architect*, vol 3(10), pp 34—39, 2004
- [5] J. Crupi, and C. Warner, "Enterprise Mashups: The New Face of Your SOA", *SOA World Magazine*, <http://soa.sys-con.com/node/719917>, 2009
- [6] S. Harris and A. Seaborne, "SPARQL 1.1 Query Language", *W3C Recommendation*. <http://www.w3.org/TR/sparql11-query/>, 2011
- [7] A. Hogan, "Integrating Linked Data though RDFS and OWL: Some Lessons Learnt", *Proc. of 5th International Conference on Web Reasoning and Rule Systems*, Springer, in press.
- [8] A. Hogan, J. Pan, A. Polleres and R. Yuan, "Scalable OWL 2 Reasoning for Linked Data", *Proc. of Reasoning Web. Semantic Technologies for the Web of Data*. LNCS vol. 6848, Springer, 2011
- [9] P. Hitzler and B. Parsia, "Ontologies and Rules", S. Staab and R. Studer, Eds. *Handbook on Ontologies*, pp. 111--132. Springer Verlag, 2nd Edition, 2009
- [10] A. Kalou, T. Pomonis, D. Koutsomitropoulos and T. Papatheodorou, "Intelligent Book Mashup: Using Semantic Web Ontologies and Rules for User Personalisation", *Proc of 4th IEEE Int. Conference on Semantic Computing*, pp. 536—541, 2010
- [11] H. J. ter Horst, "Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity", *Proc. of 4th International Semantic Web Conference*, LNCS, vol. 3729, pp. 668—684, 2005