An integrated framework for enhancing the semantic transformation, editing and querying of relational databases

Konstantinos N. Vavliakis a,b,c, Andreas L. Symeonidis a,b,*, Georgios T. Karagiannis c, Pericles A. Mitkas a,b

a Aristotle University of Thessaloniki, GR54124 Thessaloniki, Greece
b Informatics and Telematics Institute – CERTH, GR57001 Thessaloniki, Greece
c Ormylia Foundation, Art Diagnosis Center, Sacred Convent of the Annunciation, GR63071 Chalkidi, Greece

Keywords:
Relational database to ontology transformation
SPARQL query builder
Ontology editor
OWL-DL restriction creation

1. Introduction

It is common knowledge that data, either these published over the Web or those carefully preserved in private corporate enterprise structures, are most of the times stored in relational database schemes, while object-oriented and object-relational database schemes meet low resonance. The flourishing of the Semantic Web (SW), though, dictates the storing of data into ontologies and semantically-aware structures. One may, thus, identify the emergent need for efficient transformation schemes that would allow for the sound generation and population of ontologies from data stored in legacy relational data structures, giving room for advanced data manipulation, inferencing, and querying.

This transformation scheme from relational databases to sound ontologies comprises numerous interdependent facets, from the semantic traversal of relational data and the generation of the proper entities, relationships, and constraints, to the population of the ontology, the building of semantic queries, and the viewing and editing of the inferred knowledge. And, though there is a wide range of mature systems addressing one (or a few) of these operations independently, no integrated system offers all the above functionality in a user-friendly manner. One would have to install a wide variety of tools and probably learn some complex RQL (RDF-based query) language, in order to be able to exploit the potential of the semantic transformation, impose complex semantic queries to data through advanced querying interfaces, and add inferencing rules that would make implicit knowledge explicit, an issue of critical importance in a wide range of knowledge domains, like medicine and network security.

Towards this direction, we have developed Iconomy, an integrated suite for all the semantic web technologies facets. The pivotal characteristics of Iconomy are the introduction of a simple, yet powerful mechanism for the transformation of relational data to semantic entities, and the provision of a user-friendly interface for the user to easily view, manage, edit and query the ontology schema and instances. Iconomy supports knowledge inference, while it also realizes an interface for generating ontology restrictions and for allowing consistency checks on the ontology and the relationships among data. Practically, it provides an integrated graphical environment that allows both novice and expert users to easily and quickly instantiate an ontology schema (simple or sophisticated) and perform tests on it.

The rest of the paper is structured as follows: Section 2 reviews the background theory and related work, while Section 3 discusses
the framework functionality and provides a walk-through scenario of Iconomy on the CIDOC Conceptual Reference Model, an ISO standard for the Cultural Heritage domain. Section 4 thoroughly analyzes Iconomy performance with respect to a number of factors (different storage types, underlying semantic framework, datasets, dataset sizes and complexity), while Section 5 summarizes work presented and concludes the paper.

2. Background information-relevant work

The World Wide Web Consortium (W3C) is currently the main standardization body on Web Technologies, striving towards transforming the traditional Web into a Semantic Web. W3C focuses on issuing specifications and guidelines, as well as software and tools for assisting researchers develop the semantic infrastructure of the new era. Among all W3C recommendations, one may stress the design and development of the Ontology Web Language (van Harmelen et al., 2004) (OWL) and the SPARQL Query Language (Prud’hommeaux et al., 2008), which have boosted extensive work on SW applications.

Going through related bibliography, numerous tools and techniques tackling the different facets of the semantic web can be found. In the field of ontology-related programming, the Jena (Carroll et al., 2003) and Sesame (Broekstra, Kampman, & van Harmelen, 2002) frameworks are widely popular. These open-source APIs provide programmers with the ability to easily store and load RDF (Beckett et al., 2004), RDF-S (Brickley, Guha, & McBride, 2004) and OWL (van Harmelen et al., 2004) documents, infer knowledge and perform semantic queries on them.

As far as the transformation of data stored in relational databases to RDF documents is concerned, two approaches are the most dominant: (a) either relational data are transformed in RDF triples and a system is developed that allows for direct RDF queries, or (b) data remain stored in a classic relational schema and a mapping mechanism is developed, in order to transform the SQL queries to RDF queries. Tools that follow the first approach are namely D2RQ (Bizer et al., 2004), Virtuoso (Erling et al., 2007), DartGrid (Chen et al., 2005) and SquirrelRDF (Steer et al., 2006), while METAmorphosis (Svihla et al., 2004) follows the second approach.

D2RQ applies a mapping mechanism to rewrite Jena/Sesame API calls to SQL queries and passes the query results as RDF triples up to the higher layers of the Jena/Sesame frameworks. Through the D2RQ mapping, the relational database is accessed as a virtual RDF graph, thus SPARQL queries or find(s p o) functions can be formulated and results in RDF format can be drawn. Similar functionality is provided by Virtuoso, which comes both as an open-source tool, as well as a commercial product. The commercial version also incorporates a hybrid database engine with multiple operations, including RDF triple storage. Both D2RQ and Virtuoso offer a powerful declarative language for mapping relational databases to ontologies, nevertheless, neither of the two offers a graphical user interface. Thus, learning to use the tools and exploiting their full potential may require advanced skills and effort.

The DartGrid project came to fill in this gap, by providing a graphical user interface upon the D2RQ engine. DartGrid aspired to create a semantic data integration framework, based on rule-based mapping and query rewriting. Unfortunately, literature on DartGrid indicates limited scope and applicability, since it has been tested only on one pilot case scenario. Finally, SquirrelRDF is a tool that allows non-RDF data stores to be queried using SPARQL, nevertheless with certain limitations (e.g. you cannot query for properties).

On the other hand, the METAmorphosis system (Svihla et al., 2004) employs a two-layered architecture for mapping relational database content to RDF, where the mapping document is created on the first layer and the RDF instances are generated on the second.

On the creation and inspection of ontologies, one may easily identify Protégé (Noy et al., 2001) and TopBraid Composer (TopQuadrant et al., 2007) being the most widely accepted. In fact, these tools have emerged to much more than mere ontology editors; Protégé has become an open-source community framework, where research teams

Fig. 1. Iconomy main interface.
are invited to develop plug-ins on any aspect of ontology manipulation, while TopBraid has become a commercial Eclipse-based tool that offers a stable environment for ontology manipulation, supporting a wide variety of file or persistent storage formats. Through Protégé and TopBraid, semantic description of data has become easy, thus leading to the development of numerous semantically-aware languages such as SPARQL (Prud’hommeaux et al., 2008), Sesame (Haarslev, Möller, & Wessel, 2004), Jena (Broekstra et al., 2003), RDQL (Seaborne et al., 2004) and RQL (Karvounarakis et al., 2003) for advanced querying. Though powerful and expressive, these languages are difficult for the simple user to learn, leaving room for the development of various query building environments that automate the process. Dbpedia.org (Bizer, Auer, Kobilarov, Lehmann, & Cyganiak, 2007) offers such querying mechanisms employing free-text searching or subject-predicate-object pattern creation, where the user has to type the resources he/she is searching for in an interactive query builder.

Apart from the semantically-aware languages, powerful ones based on Description Logics (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003) (DL) have been developed (Baader, Horrocks, Sattler, & Kl -Künstliche Intelligenz). These languages focus on the representation of knowledge in a structured (formal) manner, employing concept names (atomic concepts), role names (atomic roles), and recursive definitions for defining concept terms from concept names and role names using constructors. Various tools exist employing DL primitives, aiming to extract knowledge implicitly stated within the ontology and to test ontology consistency. The most widely accepted ones, also known as Reasoners, are Pellet (Parsia, Sattler, & Toman, 2006), Fact++ (Tsarkov & Horrocks, 2006) and RacerPro (Haarslev, Möller, & Wessel, 2004). Alternative knowledge extraction approaches also exist (Bulskov, Knappe, & Andreasen, 2002), while even data mining techniques have been applied to tackle the problem of extracting relationships from concepts (Bernstein, Provost, & Hill, 2005). These approaches are not that popular, though.

From all the above, it is obvious that work on the individual facets of the Semantic web, i.e. ontology creation, editing, querying and reasoning, as well as semantic transformation and representation of relational data is already mature enough. Nevertheless, none of the already developed tools provides an integrated simple and easy-to-use, yet powerful and highly expression tool for supporting all the above mentioned operations. This is the reason we have developed Iconomy; it supports the on-demand incorporation of any of the powerful state-of-the-art semantic frameworks for data manipulation (either Jena or Sesame), it employs a state-of-the-art reasoner (Pellet- the Open Source OWL-DL Reasoner), which can be enabled upon user request, in order to infer knowledge that has not been explicitly stated, while it adopts the Protégé API for constraint specification. Iconomy has put all the pieces together, in an effort to simplify the process of semantic annotation.

3. Iconomy overview

In this section an overview of the functionality of Iconomy is provided, while a walk-through scenario based on the cultural heritage domain (details discussed later on in this Section) is used as the proof of Iconomy added-value. We analyze the mapping mechanism employed, as well as the interfaces for formulating semantic queries and imposing constraints on the ontology. In the scenario subsection we provide specific examples to assist the reader better grasp the Iconomy methodology for semantically manipulating and querying relational data.

3.1. Iconomy functionality

Iconomy provides a multi-functional user interface to facilitate users in their semantic queries and experiments. Through Iconomy one is able, among others, to:

- Load any ontology, either in .owl or persistent storage format.
- Encrypt and secure ontologies generated through the transformation mechanism.
- Decrypt scrambled .owl files for viewing and editing.
- Browse, walk through, and edit the class and object/datatype property trees of the ontology, as well as focus on specific instances.
- Browse, add, and edit restrictions on the ontology classes through a user-friendly interface.
- Check the consistency of the ontology and report the inconsistent terms.
- Create simple and/or complex SPARQL queries through a menu list/drop-down interface.
- View the explanation of the SPARQL queries in natural language and edit their content in SPARQL code.
- Enable/disable the built-in Reasoner (Pellet).
- Manage user accounts and their rights on the ontologies.

Fig. 1 illustrates the main pane of Iconomy, where one may identify the ‘Class/object/datatype property tree’ pane (Fig. 1.1), the ‘Semantic query creation’ pane (Fig. 1.2), the ‘Semantic query viewer and editor’ pane (Fig. 1.3), and the ‘Semantic query explanation’ pane (Fig. 1.4).

---

1 In general a concept denotes a unary, and a role denotes a binary predicate.
Algorithm 1 Partial pseudo-code for creating resources.

if (T, Co, PK ∈ S and C ∈ O){
    Q = S(T, Co, PK)
    while Q<> do {
        O = O ∪ i(T, Co, PK)
    }
}

T: table, Co: column, PK: primary key(s), S: relational schema, C: class,
O: ontology schema, Q: relational query, and
I: individual

3.2. Relational database to ontology transformation

Iconomy supports the direct transformation of any relational database to its respective ontology through a simple, yet powerful graphical tool. This way users can easily map a database schema to an ontology and instantiate it, along with its 'write' and 'update' permissions. Fig. 2 depicts the transformation mechanism.

The core of the system is the semantic transformation engine, which provides all the necessary methods to the graphical interface and coordinates the cooperating modules. For ontology manipulation, the system may either employ the Jena (v.2.5.7) or Sesame (v.2.2.4) framework APIs (upon user request). In order for the process to initiate, only information with respect to the database and ontology schema are required (currently Oracle™ and MySQL RDBMSs are supported). The outcome of the operation is the ontology schema with all the database properties, which can be exported either in a state-of-the-art file format (RDF/XML, RDF/XML-ABBREV, N3, N-Triples), or in a persistent storage format (Oracle™ and MySQL databases are directly supported in Jena, while Sesame implements its own native format). Iconomy provides password protection and encryption functionalities, so that the generated output (.owl file) can be read/edited only with Iconomy and only after providing correct user credentials.

Regarding the transformation process itself, it comprises two steps:

1. Define the new instances and the classes they belong to.
2. Link the newly created instances to object or datatype properties.

During the first step, the user has to map the appropriate column name of a table in the database to an ontology class, as well as define the primary and unique key(s) of the selected table. Subsequently, database data can be transformed to ontology instances according to the algorithm presented in Block Algorithm 1.

Algorithm 2 Partial pseudo-code for attaching properties to resources

if (T1, Co1, PK1, T2, Co2, PK2, W ∈ S and C1, P, C2 ∈ O){
    Q = S(T1, Co1, PK1, W, T2, Co2, PK2)
    while Q<> do {
        I1 = {i(T1, Co1, PK1, C1): i ∈ O}
        if P ∈ OP{
            I2 = {i(T2, Co2, PK2, C2): i ∈ O}
            O = O ∪ {I1, P, I2}
        } else if P ∈ DP{
            L = Q(j)
            O = O ∪ {I1, P, L}
        }
    }
}

OP: object properties, DP: datatype properties, L: literals

Fig. 3. Creating instances.
In the second step, the newly created instances have to be linked to object or datatype properties, thus weaving the lattice of entities. Properties are assigned to instances, while instances can be related to other instances, or literals. Through the appropriate graphical pane, the user is prompted to select the class, attributes and primary key(s) for the instances couple he/she wants to link, and consequently select the object property that associates these resources. In case of a datatype property, the class of the second resource has to remain empty, so as to create a literal for it. Finally, in order to create the appropriate batch SQL query to connect the resources, the user has to formulate the conditional constraints (‘WHERE’ clauses) referring to the corresponding columns. The algorithm for attaching properties (in pseudo-code) is provided in Block Algorithm 2.

As already mentioned, the transformation process is fully supported by the appropriate graphical interfaces and assisting mechanisms, allowing the user to quickly and easily define his/her preferred mappings. Fig. 3 depicts the three main panels of the ‘Create Instances’ tab (1: Classes, object and datatype properties tree, 2: Database tables and table columns, 3: Database to ontology mappings already defined), while Fig. 4 illustrates the ‘Connect Instances with Properties’ tab.

Drag and drop functionality is provided for all operations, while a number of data checks and software constraints have been implemented in order to reduce errors and secure the transformation process. In case instance definitions (class, table, column names and/or primary keys) are missing in the properties panel, the system automatically induces them in the instance panel and generates the appropriate instances. Data preview and editing is available for the user to validate the correctness of the process. The user may only select the table column he/she wants and the system automatically fills in the appropriate class, according to the definitions in the instances panel. He/she may even select whether an rdfs:label or rdfs:comment property will be added to the created resource, or if statements about inverse properties will be added to the ontology. Finally, a mechanism for discovering the relationships among tables and for creating the appropriate conditional constraints is provided. At any point during the transformation process the user gets controlled reports on the resources successfully created, as well as the incomplete or erroneous ones, since resources appear in different colors with respect to their status (Fig. 4).
3.3. Building semantic queries

In the ‘Semantic query creation’ pane (Fig. 12), the user is able to easily create semantic queries of the form subject–predicate–object, with the help of drop-down lists. Each list is created dynamically, with respect to the selection made by the user, and takes into account the domain and range of the selected predicate. The ‘Semantic query creation’ pane also supports the building of complex queries through ‘AND’–‘OR’–‘NOT’ operands. The subject and object clauses of a query can either be a class or an instance of the ontology under investigation. In case an instance is selected, the equality (‘=’) and inequality (‘≠’) operands are dynamically provided for the user to choose from, while in the case a datatype property is selected as the predicate, the available operands are the ‘=’, ‘≠’, ‘>’, ‘≥’, ‘<’, ‘≤’ operands, since the object is a literal.

The formulated query is automatically translated into the respective SPARQL query, and is projected in the ‘Semantic query viewer and editor’ pane (Fig. 13), where the user is able to view and edit directly the SPARQL code (expert user). On the other hand, an explanation of the query in natural language is also generated and projected in the ‘Semantic query explanation’ pane (non-expert user – Fig. 14).

Having reviewed the query, the user submits it to the Iconomy engine, and the results are projected back in an interactive manner; the user may select (double-click) any of the returned instances and go through its details, while also edit/remove capabilities are provided (‘Instance editor’ – Fig. 5).

It should be pointed out that Iconomy supersedes similar query building tools (one may refer to DBpedia – http://www.dbpedia.org for more information) in the context of interactivity and ease-of-use, providing the non-expert with an efficient way for building semantic queries and manipulating ontologies.

3.4. Adding constraints with Iconomy

Exploiting the potential of DLs encompasses the ability to validate ontology consistency and to infer relationships that are implicitly stated in the ontology. In terms of the Iconomy transformation...
model, theoretic semantics can easily lead to a formal definition of consistency:

An ABox $A$ is consistent with respect to a TBox $T$, if there is an interpretation that is a model of both $A$ and $T$.

OWL defines necessary (superclass) and necessary & sufficient (equivalent class) conditions for the corresponding file, and the reasoning tasks are performed through restrictions that the user defines. Iconomy provides a user-friendly environment for viewing and creating such conditions (Fig. 6), following the Protégé and TopBraid Composer paradigms. It should be stated, though, that specifying the appropriate constraints may require advanced user skills.

3.5. A walk-through scenario of the Iconomy framework

As already discussed, semantic analysis can be applied in practically all domains, nevertheless specific domains that evolve in a graph lattice manner are more appropriate to apply. Special interest groups are usually established (most of the times under the W3C), working towards releasing standards and specifications on a specific domain. Such a remarkable effort exists on the cultural heritage domain, where the CIDOC Conceptual Reference Model (Doerr, 2003) (CIDOC-CRM) has been developed. CIDOC-CRM is an official ISO standard that provides a formal structure for describing the implicit and explicit concepts and relationships used in cultural heritage documentation. It aspires to promote a common understanding of cultural heritage information by providing a unified and extensible integration layer for cultural heritage information coming from heterogeneous sources, such as museums, libraries and, historical archives. In other words, CIDOC-CRM provides a common language for domain experts and implementers to formulate requirements for information systems and to serve as a best practice guide for conceptual modeling.

At the Ormylia Foundation-Art Diagnosis Center, Byzantine artwork is documented and analyzed using multidisciplinary techniques, such as multispectral imaging, spectroscopy and acoustic microscopy. The information derived from this kind of analysis, along with descriptive, interpretative, aesthetic and technical information are currently stored in a well-designed Oracle Database Management System. Nevertheless, later reference to this data is extremely difficult, due to the complex relationships between the data elements stored, thus prohibiting advanced querying on the nature, technique, etc of the artworks stored in the RDBMS (Relational Database Management System).

Through Iconomy we managed to transform these data into a consistent ontology, upon which querying and reasoning is now easy to perform.

3.5.1. Extending CIDOC-CRM

In order to ensure the unambiguous expression of all the terms documented in Ormylia Foundation’s database, we have extended the CIDOC-CRM with terms capable of semantically annotating the database. Using the ontology editing functionality of Iconomy, we resulted in the enrichment of the CIDOC-CRM model with new classes and properties.2 CIDOC-CRM now comprises 257 classes and 298 properties, not taking inverse properties into account. Due to the large number of classes and the complexity of CIDOC-CRM, as well as the complicated onomatology, we refrain from presenting the entire ontology schema. Indicative changes

---

2 The initial model comprised 81 classes and 132 unique properties.
to the ontology include the modification of the E55.Type class, to which more than 15 subclasses have been added (such as the E211.Style.Type, E91.Technique and E90.Artwork.Type subclasses), and the creation of E123.Descriptive.Information, E246.Scientific.Information as subclasses of E73.Information.Object class. One may refer to (Karagiannis et al., 2009) for more information.

3.5.2. Transforming the relational database schema

Algorithm 3 provides part of the transformation scheme applied. It transforms records from the CREATOR.NAME and ITEM.TITLE table columns into instances of the E211.Person and E22.Man.Made.Object classes, respectively. Additionally, through the Select clause, the algorithm assigns the P168.is.the.creator.of object property to those of the newly created instances (Persons) that have created more than 10 artworks.

Algorithm 3 Configuration example for connecting instances with properties

E21_Person,CREATOR.NAME,ID,P168_is_the_creator_of,
E22_Man_Made_Object, ITEM.TITLE,ID,
ITEM.CREATOR_ID=CREATOR.ID and
ITEM.CREATOR_ID in
(select CREATOR_ID from ITEM group by CREATOR_ID
having count(CREATOR_ID)>10)

3.5.3. Performing queries with Iconomy

In order to better demonstrate the querying mechanism of Iconomy through an example, let us consider a query where the user wishes to find all portable artworks that are not of Cretan technique and which consist of three (3) or more layers. Through the 'Semantic query creation' pane (Fig. 12), such a query is easily formulated; the user selects 'Artwork' as the Subject and dynamically all properties related to the specific Subject are loaded in the Predicate drop-down list. In a similar manner, selection of a Predicate value specifies the available values for the Object Type, in case an object property is selected. Since 'Type' and 'Technique' are classes, the interface provides the user the ability to define equality/inequality on a specific instance value. Upon pressing the 'More' button, Iconomy provides a second row of drop-down boxes for the user to create the 'AND/OR/NOT' clause of the query. In case a datatype property is selected the user can fill in the appropriate values. At this point one should mention that, though the 'NOT' operant is not supported in SPARQL syntax, it is supported by the ARQ module of the Jena API and, thus, incorporated in Iconomy. Fig. 7 depicts the formulation of the query and its natural language explanation as provided by the system, while Block Algorithm 4 illustrates the constructed SPARQL query.

Algorithm 4 Query in SPARQL as created from the system

SELECT DISTINCT ?x WHERE {
?f a<http://cidoc.ics.forth.gr/OWL/crm3.4.9#Technique>. FILTER (?f !=<http://cidoc.ics.forth.gr/crm3.4.9#Cretan>)
UNION
FILTER (?g>=3)

Table 1

Summary of the performed sets of experiments.
3.5.4. Creating restrictions on byzantine art domain

As already stated, Iconomy employs the Pellet reasoning mechanism, in an effort to associate knowledge nuggets located in the ABox, TBox and RBox, infer on them, and imply knowledge. We demonstrate the reasoning capabilities of Iconomy through an example.

Let us assume that every Artwork of ‘Pentecost’ theme that consists of ‘Orpiment’ and ‘Caput Mortuum’ is an artwork of the creator ‘Ritzos Andreas’, since he is the only known painter that used this combination of pigments in this type of paintings. For the user to define such a rule in the ontology he/she should need to create a new class named, say ‘Artworks of Ritzos Andreas’, which must be a subclass of the existing ‘Artwork’ class. Then, a necessary and sufficient condition (restriction) for the newly created class should be created, to successfully infer on all instances that are instances of the class Artwork of type ‘Pentecost’ and consist of ‘Orpiment’ and ‘Caput Mortuum’, as being instances of the new class. The restriction should be of the form:

\[ \text{Restriction} (\text{Artwork AND hasTypevalue Pentecost AND consistsOf value gesso AND consistsOf value lead white}). \]

The restriction will be created as an anonymous class, which will be an equivalent class to ‘Artworks of Ritzos Andreas’. Fig. 8 illustrates the creation of such a restriction. Iconomy can now infer that all the relative instances belong to ‘Ritzos Andreas’, without any further actions required by the user. It should be denoted that, in order to perform such assumptions the Reasoner should be enabled; otherwise the class ‘Artworks of Ritzos Andreas’ would be assigned no individuals. Of course, there are much more complicated reasoning procedures, Iconomy provides the necessary interfaces to accomplish such reasoning tasks.

Users should be extremely careful when creating restrictions, as these operations are prone to reasoning errors; in our example a closed world assumption state is implied, as there may be unknown painters that also used this combination of pigments.

4. Iconomy performance experiments

Four sets of experiments were performed, in order to prove Iconomy’s added-value in terms of performance, integrity and scalability. Different experiment configurations aimed to investigate the performance of the two APIs employed (Jena and Sesame) with respect to different types of storage, different datasets, dataset sizes and dataset complexity. As far as storage is concerned, tests were executed using both Memory as well as Persistent Storage. In the case of Memory Storage, Iconomy was used to build a memory-based ontology model (saved as an OWL file using RDF/XML coding), employing both the Jena and Sesame APIs, while in the case
of Persistence Storage, we selected a Jena-MySQL and Sesame Native scheme, respectively. As far as data are concerned, we generated two SQL dumps of data, one containing DBLP data and one containing data from the ECML-KDD Discovery Challenge 2009 (DC09). These two datasets were used in order to instantiate the corresponding ontology schemas, i.e. the DBLP and the Semantic Web Research Community (SWRC) ontologies. For each test, two types of instantiations were performed: one resulting in an ontology containing only instances (in other words a \textit{sparse graph} consisting only of vertices) and one resulting in an ontology with instances connected by a large number of object properties (a relatively \textit{dense graph}). In all cases we build a number of ontologies with sizes varying from a few thousand up to 20 million triples.

All tests were executed on an Intel Core 2 Quad at 2.4MHz PC, running Windows XP SP3, with 4GB of memory and a 500GB, 7200RPM hard disk. We used JDK v.1.6, Jena v.2.6.0, Sesame v.2.2.4 and set the Maximum Java Heap Size to 512MB. No optimizations were made to the persistent storages schemes (addition of customized indexes in the databases). Table 1 summarizes the configuration of each set of experiments, while Figs. 9–12 depict the comparative results.

4.1. Comparing the APIs with respect to memory storage

Naturally, memory storage (Fig. 9) greatly outperforms persistent storage (Figs. 10–12). Nevertheless, one may easily identify that there is a threshold in the number of triples a semantic API can handle, which depends on its internal mechanism with respect to the size of the maximum available Java Heap Size. On our experimental setting, the threshold for the Jena API (Fig. 9a and c) was around 1.3 million triples for dense graphs, while for the Sesame API (Fig. 9b and d) the threshold was considerably higher (3 million triples). One may also notice that Sesame is a bit faster than Jena in memory storage. When applied on sparse graphs, though, capacity in triple storage is comparable, with Jena managing to store slightly more triples.

May one want to overcome the triple threshold imposed by computational limitation, he/she should either increase the maximum Java Heap Size (not always effective, though) or use persistent storage. In the latter case, he/she would obviously have to pay the respective cost in performance.

4.2. Comparing the APIs with respect to persistent storage

Fig. 10 provides a similar comparison of the Jena and Sesame APIs, nevertheless with respect to Persistent storage. One may easily identify that in the case of sparse graphs (Fig. 10a and b), Sesame Native outperforms Jena-MySQL for graphs with less than 8–9 million triples, while for larger graphs Jena-MySQL seems to have a better performance for both datasets. This does not apply
for dense graphs, though (Fig. 10c and d), where Sesame Native is faster than Jena-MySQL in all cases, for both datasets.

4.3. Comparing the APIs with respect to the datasets

Fig. 11 depicts the APIs ability in generating sparse and dense graphs for the two datasets. Primary keys at the DC09 dataset (SWRC ontology) tables comprise three attributes (columns), while DBLP dataset tables have single-column Primary keys. Consequently, more complex queries have to be formulated in order to access information in the DC09 dataset, while the average URI length that an SQL tuple is converted to is greater, also. Nevertheless, the two datasets are converted into ontologies in comparable times for similar setups with both APIs, following the trendline of the graph. Only in the case of Sesame Native on dense ontologies (Fig. 11d) a variation from the trendline is observed for the SWRC Dense dataset for more than 6 million triples. This issue is discussed in the next subsection.

4.4. Comparing the APIs with respect to sparse vs dense graph generation

Fig. 12 illustrates sparse and dense graph generation with different Jena and Sesame configurations. One may notice (Fig. 12a and b) that Jena-MySQL ‘prefers’ the generation of sparse graphs than dense graphs with the same number of triples. In the case of Sesame Native though, no safe conclusions may be drawn. For SWRC, the generation time for small (<3 million triples) sparse and dense graphs is equivalent, while large dense graphs (>3 million triples) are generated faster with respect to sparse ones (Fig. 12c). For DBLP, the performance of Sesame Native is analogous, but only up to the bottleneck point of 6 million triples (Fig. 12d). On exceeding this threshold, the generation time of dense graphs becomes significantly greater. The only logical assumption we can make on this is that some internal cache manager or hashmap of the Sesame Native engine reaches an upper threshold and this is the reason for the great difference between the generation time of ontologies containing 6 and 8 million triples (also observed in the previous subsection).

It should be mentioned that we refrained from presenting results on semantic querying on RDF data, since performance is solely dependent on the Jena/Sesame APIs and the type of semantic repository we use, rather than economy. One may find a rich bibliography in benchmarking RDF storage and retrieval in Liu and Hu (2005), Rohloff, Dean, Emmons, Ryder, and Sumner (2007), Svihla and Jelišnek (2007), Wilkinson, Sayers, Kuno, and Reynolds (2003).
5. Conclusions and further work

*Iconomy* aspires to provide an integrated framework for semantically manipulating and querying data residing both in ontologies and relational databases. It provides advanced options on the creation and synchronization of an ontology to and from a relational database, the automatic creation of queries, and data viewing/editing. And, all this functionality is supported by a simple and friendly user interface.

Numerous tests were performed in order to validate the performance of *Iconomy*, with respect to ease-of-use, scalability and extensibility. The graphical user interface was tested and successfully handled ontologies with up to 4 million triples. It should be stated though, that this constraint is posed by the Jena API and the maximum Java Heap size available, rather than the user interface itself. The 4-million-triples constraint can be eliminated when loading an ontology from persistence storage, but in this case time greatly increases and the reasoning services practically become unavailable.

In order to prove the predictable behavior of the semantic engine of *Iconomy*, a number of sets of experiments were run on different setups and interesting conclusions were drawn: (a) the transformation mechanism performs the same with different data-sets, but is greatly depends on the density of the generated graph and the semantic API engine it employs, (b) memory storage outperforms Persistent Storage, but cannot scale up to million of triples, (c) the Sesame API, in general, better handles dense graphs than the Jena API, while (d) the Jena API performs better on the generation of sparse graphs with more than 8–9 million triples.

Future research efforts include further improvement of the transformation mechanism with respect to the ontology mapping process, as well as towards improving extensibility of *Iconomy* by providing support for more persistent storage schemes. The incorporation of other semantic languages, such as SeRQL, is considered, while the ontology editing functionality may be upgraded.

Considering that the Semantic Web is the new era, the need for efficient tools and technologies is evident. *Iconomy* aspires to provide such a vehicle for allowing the seamless transition from the traditional to the Semantic Web, through its manipulation of large data sets, reasoning and incorporation of legacy data to new, semantically-annotated entities.

References