Semantic Web 0 (0) 1
IOS Press

Hierarchical Visual Exploration and Analysis on the Web of Data

Nikos Bikakis [a,b], George Papastefanatos [b], Melina Skourla [a] and Timos Sellis [c]
[a] *National Technical University of Athens, Greece*
[b] *IMIS, ATHENA Research Center, Greece*
[c] *RMIT University, Australia*

Abstract. The purpose of data visualization is to offer intuitive ways for information perception and manipulation, especially for non-expert users. The Web of Data has realized the availability of a huge amount and variety of datasets; most of them offer SPARQL endpoints for online access and analysis. However, most traditional visualization tools and methods operate on an offline way without offering the ability for ad hoc visualization and analysis of large dynamic sets of data. In this work, we present a model for building, visualizing, and interacting with hierarchically organized Linked Data (LD). Our model is build on top of a lightweight tree-based structure which can be easily constructed on-the-fly for a given set of data. This tree structure organizes input data objects into a hierarchical model based on the values of their properties that they exhibit. Additionally, we define two versions of this structure, which adopts different data organization approaches, well-suited to visual exploration and analysis context. Furthermore, statistical computations can be efficiently performed on-the-fly in the proposed structure. The presented model is realized in a web-based prototype tool, called rdf:SynopsViz that offers hierarchical visual exploration and analysis over LD datasets. Finally, we provide an evaluation of our approach employing LD datasets.

Keywords: Linked Data, Semantic Web, Visual analytics, Hierarchical exploration, RDF Data.

1. Introduction

The purpose of data visualization is to offer intuitive ways for information perception and manipulation that essentially amplify, especially for non-expert users, the overall cognitive performance of information processing. This is of great importance in the Web of Data, where the volume and heterogeneity of available information make it difficult for humans to manually explore and analyse large datasets. Moreover, information visualization enables users to infer correlations and causalities and supports sense-making activities [22] over data that are not always possible with traditional data mining techniques.

Following the abundance of Linked Data (LD) on the web, several recent efforts have offered tools and techniques for information visualization and visual exploration of LD in many different domains, such as statistical, biological, spatiotemporal [23,53]. They offer different ways to consumers of LD to interact with the data and perform visual operations, such as navigation and browsing over resources, visual representation and analysis of resources and the relationships between them, as well as visual querying and filtering. These are further specialized to the requirements of the specific domain; e.g., graph-based visualizations are usually used for link-based analysis, maps are employed for spatial-enriched data, charts and timelines focus on numerical and temporal data analysis, etc.

One of the major challenges in the LD visual exploration is related to the size that characterizes many popular LD datasets. Considering the visual information seeking mantra: "*overview first, zoom and filter, then details on demand*" [64], gaining overview is a crucial task in the visual exploration scenario. However, offering an overview of a large dataset, is an extremely challenged task. Additionally, information overloading is a common issue in large datasets visualizations; a basic requirement for the proposed ap-

proaches is to offer mechanisms for information abstraction and summarization.

The above challenges can be overcome by adopting hierarchical approaches [29]. Hierarchical approaches allow the visual exploration of very large datasets, offering a dataset overview, as well as an intuitive and user-friendly way for finding specific parts within a dataset. Particularly, in hierarchical approaches, the user first obtains an overview of the dataset (both structure and a summary of its content) before proceeding to data exploration operations, such as roll-up and drill-down, filtering out a specific part of it and finally retrieving details about the data. Therefore, hierarchical approaches directly support the visual information seeking mantra. Also, hierarchical approaches can effectively address the problem of information overloading as it provides information abstraction and summarization.

A second challenge is related to the availability of SPARQL endpoits for online data access and retrieval, possessing the challenge of visualizing data that is dynamically retrieved by a remote site. In this respect, visualization techniques must offer scalability and efficient processing for on-the-fly analysis and visualization of dynamic datasets. Finally, the requirement for online visualization must be coupled with the diversity of preferences and requirements posed by different users and tasks. Therefore, the proposed approaches should provide the user with the ability to customize the exploration experience, allowing users to organize data into different ways according to the type of information or the level of details she wishes to visualize.

Addressing the above challenges, in this work, we present a generic model that combines user-customized hierarchical exploration with online analysis of LD. At the core lies a lightweight hierarchical model, constructed on-the-fly for a given set of data. Particularly, the proposed model is a tree-based structure that aggregates LD resources (i.e., RDF triples) into multiple levels of hierarchically related groups based on the numeric and temporal values of one or more properties. It also enriches groups with statistical information regarding their contents, offering richer overviews and insights on the detailed data. An additional feature is that it allows users to organize data exploration in different ways, by parametrizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, etc. Finally, the proposed model is realized in a web-based tool, called rdf:SynopsViz that offers a variety of visualization techniques, such as charts, timelines, and treemaps for hierarchical visual exploration and analysis over LD datasets.

Contributions. The main contributions of this paper are summarized as follows.

- We introduce a model for building, visualizing, and interacting with hierarchically organized numeric and temporal LD.
- We implement our model as a lightweight, main memory tree-based structure, which can be easily constructed on-the-fly.
- We propose two tree structure versions, which adopt different approaches for the data organization.
- We describe a simple method to estimate the tree construction parameters, when no user preferences are available.
- We develop a prototype system which implements the presented model, offering hierarchical visual exploration and analysis over LD.
- We conduct an experimental evaluation using LD datasets.

Outline. The remaining of this paper is organized as follows. Section 2 presents the hierarchical model developed in rdf:SynopsViz. Then, Section 3 presents the rdf:SynopsViz system and demonstrate the basic functionality. Section 4 reviews related work. The evaluation of our system is presented in Section 16, while Section 6 concludes this paper.

2. The HETree Model

In this section we present *HETree* (Hierarchical Exploration Tree), our model for building, visualizing, and interacting with hierarchically organized LD. *HETree* is defined in the context of hierarchical visual exploration and analysis over one-dimensional numeric and temporal (i.e., datetime values) LD. The proposed model can be adopted by various existing visualization techniques (e.g., charts, scatterplots, timeline, etc.), offering scalable and multilevel visual representations.

In what follows, we present some basic aspects of our working scenario (i.e., visual exploration and analysis scenario) and highlight the main assumptions and requirements employed in the construction of our model. First, the input data in our scenario can be retrieved directly from a triple store, but also produced dynamically; i.e., either from a SPARQL query or from

data filtering (e.g., faceted browsing). Thus, we consider that data visualization is performed online; i.e., we do not assume an offline preprocessing phase in the construction of the visualization model. Second, users can specify different requirements or preferences with respect to the data organization. For example, a user prefers to organize the data as a deep hierarchy for a specific task, while for another task a flat hierarchical organization is more appropriate. Therefore, even if the data is not dynamically produced, the data organization is dynamically adapted to the user preferences. The same also holds for any additional information (e.g., statistical information) that is computed for each group of objects. This information must be recomputed when the groups of objects (i.e., data organization) are modified.

From the above, a basic requirement is that the model must be constructed on-the-fly for any given data and users preferences. Therefore, we implement our model as a lightweight, main memory tree structure, which can be easily constructed on-the-fly. We define two versions of this tree structure, following data organization approaches well-suited to visual exploration and analysis context: the first version considers fixed-range groups of data objects, whereas the second considers fixed-size groups. Finally, our structure allows efficient on-the-fly statistical computations, which are extremely valuable for the exploration and analysis scenario.

The basic idea of our model is to hierarchically group data objects based on values of one of their properties. Input data objects (i.e., RDF triples) are stored at the leaves, while internal nodes aggregate their children nodes. The root of the tree represents (i.e., aggregates) the whole dataset. The basic concepts of our model can be considered similar to a simplified version of a static 1D R-Tree [34]. R-Tree is a disk-based multi-dimensional indexing structure, which has been proposed to efficiently handle spatial queries. In order to provide efficient query processing in disk-based environment, R-Tree considers a large number of issues (e.g., space coverage, nodes overlaps, fill guarantees, etc.). On the other hand, we introduce a lightweight, main memory structure that hierarchically organizes 1D data and is easily constructed on-the-fly. Also, the proposed structure aims at organizing the data in a practical manner for a (visual) exploration scenario, rather than for indexing and querying efficiency.

Regarding the visual representation of the model and data exploration, we consider that both data objects sets (leaf nodes) and entities representing groups of objects (internal nodes) are visually represented enabling the user to explore the data in a hierarchical manner. Starting either from the tree root or a leaf node, the user visually interacts with organized data, by traversing and rendering nodes of the tree. For example, in a top-bottom traversal path, the user starts from the root and, for each node, only its children are rendered and are available for further exploration. In the case that the current node is a leaf node, the actual data objects (i.e., triples) are rendered. Note that our tree structure organizes data in a hierarchical model without setting any constraint at the way the user interacts with these hierarchies. As such, it is possible that different strategies can be adopted, regarding the traversal policy, as well as the nodes of the tree that are rendered in each visualization stage.

In the rest of this section, preliminaries are presented in Section 2.1. In Section 2.2, we introduce the proposed tree structure. Sections 2.3 and 2.4 present the two versions of the structure. Finally, Section 2.5 discusses the specification of the parameters required for the tree construction, and Section 2.6 presents how statistics computations can be performed over the tree.

$p0$ *age* 35	$p5$ *age* 35
$p1$ *age* 100	$p6$ *age* 45
$p2$ *age* 55	$p7$ *age* 80
$p3$ *age* 40	$p8$ *age* 20
$p4$ *age* 30	$p9$ *age* 50

Fig. 1. Running example input data (RDF triples)

2.1. Preliminaries

Given an *RDF dataset* R consisted of a set of *RDF triples*. As *input data*, we assume a set of RDF triples D, where $D \subseteq R$ and triples in D have as objects either numeric (e.g., xsd:int, xsd:decimal) or temporal values (e.g., xsd:dateTime, xsd:date). In other words, we consider triples that contain as predicates datatype properties with either numeric or temporal ranges. Let tr be an RDF triple, $tr.s$, $tr.p$ and $tr.o$ represent, respectively, the *subject*, *predicate* and *object* of the RDF triple tr.

Given input data D, S is an *ordered set* of RDF triples, produced from D, where triples are sorted based on objects' values, in ascending order. Assume that $S[i]$ denote to the i-th triple, with $S[1]$ be the first triple. Then, for each $i < j$, we have that $S[i].o \leq S[j].o$. Also, $D = S$, i.e., for each $tr \in D$ iff $tr \in S$.

Figure 1 presents a set of 10 RDF triples, representing persons and their ages. For simplicity, the RDF triples are presented in a simplified "abstract" form (omitting namespaces and IRIs). Hence, in Figure 1, we assume that the subjects $p0$-$p9$ are instances of a class *Person* and the predicate *age* is a datatype property with *xsd:int* range.

Example 1. From Figure 1 we assume the RDF triple $tr = p0$ *age* 35, then we have that $tr.s = p0$, $tr.p = age$ and $tr.o = 35$. Additionally, considering as input data D, the set of all RDF triples in Figure 1, the ordered set S is resulted by ordering the triples D based on object values, in ascending order, i.e., $S = \{p8$ *age* $20,\ p4$ *age* $30,\ p0$ *age* $35,\ p5$ *age* $35,\ p3$ *age* $40, p6$ *age* $45, p9$ *age* $50, p2$ *age* $55, p7$ *age* $80,\ p1$ *age* $100\}$. In S we have that, $S[1] = p8$ *age* 20 and $S[10] = p1$ *age* 100. ■

Assume an *interval* $I = [a, b]$, where $a, b \in \mathbb{R}$; then, $I = \{k \in \mathbb{R} \mid a \leq k \leq b\}$. Similarly, for $I = [a, b)$, we have that $I = \{k \in \mathbb{R} \mid a \leq k < b\}$. Let I^- and I^+ denote the lower or upper bound of the interval I, respectively. That is, given $I = [a, b]$, then $I^- = a$ and $I^+ = b$. The *length* of an interval I is defined as $|I^+ - I^-|$.

In this work we assume rooted trees. The number of the children of a node is its *degree*. Nodes with degree 0 (i.e., no children) are called *leaf nodes*. The *level* of a node is defined by letting the root be at level zero. If a node is at level l, then its children are at level $l + 1$. The *height* of a tree is the maximum level of any node in the tree. Any non-leaf node is called *internal node*. The *degree of a tree* is the maximum degree of a node in the tree. An *ordered tree* is a tree where the children of each node are ordered. *Balanced* is a tree where the level of each leaf node differs at most one from the level of other leaves nodes. A tree is called an *m-ary tree* if every internal node has no more than m children. A *full m-ary tree* is a tree where every internal node has exactly m children.

2.2. The HETree Structure

In this section, we present in more details the *HETree* structure. HETree hierarchically organizes numeric and temporal[1] data into groups; intervals are used to represents these groups. HETree is defined by the tree degree and the number of leaf nodes[2]. Essentially, the number of leaf nodes corresponds to the number of groups where input data objects are organized. The tree degree corresponds to the (maximum) number of groups where a group is split in the lower level.

Given a set of RDF triples D, a positive integer l denoting the number of leaf nodes; and a positive integer d denoting the tree degree; a *HETree* (D, l, d) is an *ordered balanced d-ary tree*, with the following basic properties.

- The tree has exactly l number of leaf nodes.
- Each leaf node contains a set of RDF triples, sorted in ascending order based on their objects' values.
- Each internal node, has at most *d* children nodes. Let n be an internal node, $n.c_i$ denote the i-th child for the node n, with $n.c_1$ be the leftmost child.
- Each node corresponds to an interval. Given a node n, $n.I$ denote the interval for the node n.
- At each level, all nodes are sorted based on the lower bounds of their intervals. That is, let n be an internal node, for any $i < j$, we have that $n.c_i.I^- \leq n.c_j.I^-$.
- For a leaf node, its interval is bound by the objects' values of the triples included in this leaf node. Let n be the leftmost leaf node; assume that n contains x triples from D. Then, we have that $n.I^- = S[1].o$ and $n.I^+ = S[x].o$, where S is the ordered RDF set resulted from D.
- For an internal node, its interval is bound by the union of the intervals of its children. That is, let n be an internal node, having k children nodes; then, we have $n.I^- = n.c_1.I^-$ and $n.I^+ = n.c_k.I^+$.

Example 2. Given the set of RDF triples D from Figure 1. Figure 2 presents a HETree with five leaf nodes (i.e., $l = 5$) and degree equal to three (i.e., $d = 3$). Considering the leftmost leaf node d, we can see that it contains two triples in the following order $p8$ *age* 20, $p4$ *age* 30. As a result, the lower bound for the its interval, is equal to the value of the first triple object, i.e., $d.I^- = 20$; the upper bound

[1] Note that, our structure uniformly handles numeric and temporal data. Also, other types of one-dimensional data may be supported, with the requirement that a total order can be defined over the data.

[2] Note that, following a similar approach, the HETree can also be defined by specifying the tree height instead of degree or number of leaves.

is equal to the value of the last triple object, i.e., $d.I^+ = 30$. For the internal node b, its interval is bound by the intervals of its children nodes (d, e, f); i.e., considering the lower bound of its leftmost child (i.e., 20) and the upper bound of its rightmost child (i.e., 45). ■

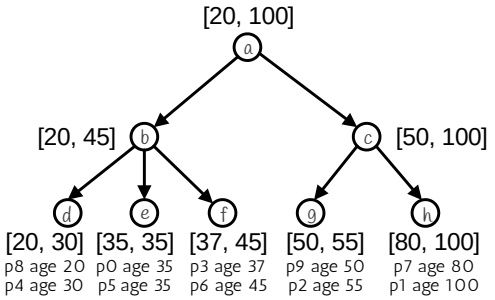

Fig. 2. A Content-based HETree (HETree-C)

We, furthermore, present two different approaches for organizing the data in the HETree. Assume the scenario in which, a user wishes to (visually) explore and analyse the historic events from DBpedia, per decade. In this case, user orders historic events by their dates and organizes them into groups of equal ranges (i.e., decade). In a second scenario, assume that a user wishes to analyse in the Eurostat dataset the gross domestic product (GDP) organized into fixed groups of countries. In this case, the user is interested in finding information like: the range and the variance of the GDP values over the top-10 countries with the highest GDP factor. In this scenario, the user orders countries by their GDP and organizes them into groups of equal sizes (i.e., 10 countries per group).

In the first approach, we organize RDF triples into groups, where the objects values of each group covers equal range of values. In the second approach, we organize RDF triples into groups, where each group contains the same number of triples. In the following sections, we present in details the two approaches for organizing the data in the HETree.

2.3. A Content-based HETree (HETree-C)

In this section we introduce a version of the HETree, named *HETree-C* (Content-based HETree). This HETree version organizes data into equally sized groups. The basic property of the HETree-C is that each leaf node contains approximately the same number of RDF triples and the content (i.e., triples) of a leaf node specifies its interval. For the tree construction, the triples are first assigned to the leaves and then the intervals are defined.

A *HETree-C* (D, l, d) is a HETree, with the following extra property. Each leaf node contains λ or $\lambda - 1$ triples, where $\lambda = \left\lceil \frac{|D|}{l} \right\rceil$[3]. Particularly, the $l - (\lambda \cdot l - |D|)$ leftmost leaves contain λ triples, while the rest leaves contain $\lambda - 1$. We can equivalently define the HETree-C by providing the number of triples per leaf λ, instead of the number of leaves l.

Example 3. Figure 2 presents an HETree-C constructed by considering the set of RDF triples D from Figure 1, $l = 5$ and $d = 3$. As we can observe, all the leaf nodes contain equal number of triples. Particularly, we have that $\lambda = \lceil \frac{10}{5} \rceil = 2$. Also, we have that the $5 - (2 \cdot 5 - 10) = 5$ leftmost leaves (i.e., all leaves in this case), will contain λ triples. As we can verify from Figure 2, all leaves (d, e, f, g, h) contain 2 triples. Note also that all nodes in HETree-C define closed intervals. ■

Algorithm 1. Create HETree-C (D, l, d)

Input: D: set of triples; l: number of leaf nodes; d: tree degree
Output: r: root node of the HETree-C tree

1 $S \leftarrow$ sort D based on objects values
2 $L \leftarrow$ ConstructLeaves-C(S, l)
3 $r \leftarrow$ ConstructIntervalNodes(L, d)
4 **return** r

2.3.1. The HETree-C Construction

We construct the HETree-C in a bottom-up way. Algorithm 1 describes the HETree-C construction. The algorithm takes as input: (1) a set of RDF triples D; (2) the number of leaf nodes l; and (3) the tree degree d. Initially, the algorithm sort the RDF triples set D in ascending order, based on triples objects values (*line 1*). Then, the algorithm uses two main procedures to construct the tree. The first procedure, named ConstructLeaves-C, creates the leaf nodes of the tree (*line 2*). The second procedure, named ConstructInternalNodes, creates the internal nodes of the tree (*line 3*). Finally, the root node of the constructed tree is returned (*line 4*).

[3]We assume here that, the number of triples is greater than the number of leaves; i.e., $|D| > l$.

Procedure 1: ConstructLeaves-C(S, l)

Input: S: ordered set of triples; l: number of leaf nodes
Output: L: ordered set of leaf nodes

```
1  λ ← ⌈|S|/l⌉
2  k ← l − (λ ∗ l − |S|)
3  firstTriple ← 1
4  for i ← 1 to l do
5      create an empty leaf node n
6      if i ≤ k then
7          numOfTriples ← λ
8      else
9          numOfTriples ← λ − 1
10     lastTriple ← firstTriple + numOfTriples
11     for t ← firstTriple to lastTriple do
12         insert triple S[t] into leaf node n
13     n.I⁻ ← S[firstTriple].o
14     n.I⁺ ← S[lastTriple].o
15     L[i] ← n
16     firstTriple ← lastTriple + 1
17 return L
```

Procedure 1 presents the pseudocode for the ConstructLeaves-C procedure. First, the procedure uses an ordered set of RDF triples S and creates l leaf nodes containing the S triples. Then, the procedure computes the number of triples per leaf λ (*line 1*), as well as the number of leaves that contain λ triples (*line 2*). Then, l leaf nodes are constructed (*lines 4–16*). For the first k leaves, λ triples are inserted, while for the rest leaves, $\lambda - 1$ triples are inserted (*lines 6–9*). The interval of each leaf is specified by the objects values of the first and last triple inserted in this leaf (*lines 13–14*). Finally, the set of created leaf nodes is returned (*line 17*).

Procedure 2 describes the ConstructInternalNodes procedure. This procedure builds the internal nodes in a recursive manner. Particularly, the procedure takes as input a set of nodes H, as well as the tree degree d. The basic idea of this procedure is the following. For the nodes H, their parents nodes P are created; then, the procedure calls itself using as input the parent nodes P. The recursion terminates when the number of created parent nodes is equal to one; i.e., the root of the tree is created.

In more details, the procedure computes the number of parents to be created (*line 1*), as well as the degree for the last parent (*line 2*). Then, the procedure creates *numOfParents* nodes (*lines 4–16*), for each parent node the appropriate children nodes are assigned (*lines 11–12*). The interval of each parent

Procedure 2: ConstructIntervalNodes(H, d)

Input: H: ordered set of nodes; d: tree degree
Output: r: root node for H
Variables: P: ordered set of H's parent nodes

```
1  numOfParents ← ⌈|H|/d⌉
2  lastParentDegree ← d − (numOfParents ∗ d − |H|)
3  firstChild ← 1
4  for p ← 1 to numOfParents do
5      create an empty internal node n
6      if p = numOfParents then
7          numOfChildren ← lastParentDegree
8      else
9          numOfChildren ← d
10     lastChild ← firstChild + numOfChildren
11     for j ← firstChild to lastChild do
12         n.c_j ← H[j]
13     n.I⁻ ← H[firstChild].I⁻
14     n.I⁺ ← H[lastChild].I⁺
15     P[p] ← n
16     firstChild ← lastChild + 1
17 if numOfParents=1 then
18     r ← P
19     return r
20 else
21     return ConstructIntervalNodes(P, d)
```

node is specified by the intervals of its children nodes (*lines 13–14*). After the creation of the parent nodes and based on their number, the procedure either returns the created nodes (in case of one parent node) (*line 19*), or call itself using as input the created parent nodes P (*line 21*).

Computational Analysis. The computational cost for the HETree-C construction algorithm (Algorithm 1) is the sum of three parts. Assume that $|D| = \kappa$. The first is sorting the input data, which can be done in the worst case in $O(\kappa log\kappa)$, employing a linearithmic sorting algorithm; e.g., merge-sort. The second part is the ConstructLeaves-C procedure, which requires $O(l \cdot \lambda) = O(l \cdot \lceil \frac{\kappa}{l} \rceil) = O(\kappa)$. The third part is the ConstructInternalNodes procedure, which requires $O(\lceil \frac{l}{d} \rceil \cdot d + \lceil \frac{l}{d^2} \rceil \cdot d + \lceil \frac{l}{d^3} \rceil \cdot d + \ldots) = O(\lceil \frac{l}{d} \rceil \cdot d)$. Since we have that $\kappa > l$, then in worst case $O(\lceil \frac{l}{d} \rceil \cdot d) = O(\lceil \frac{\kappa}{d} \rceil \cdot d) = O(\kappa)$. Therefore, the overall computational cost for the HETree-C construction in the worst case is $O(\kappa log\kappa + \kappa + \kappa) = O(\kappa log\kappa) = O(|D|log|D|)$.

2.4. A Range-based HETree (HETree-R)

The second version of the HETree is called *HETree-R* (Range-based HETree). HETree-R organizes data into equally ranged groups. The basic property of the HETree-R is that each leaf node covers an equal range of values. Therefore, in HETree-R, the data space defined by the objects values is equally divided over the leaves. Opposite to HETree-C, in HETree-R the interval of a leaf specifies its content. Therefore, for the HETree-R construction, the intervals of all leaves are first defined and then triples are inserted.

A *HETree-R* (D, l, d) is a HETree, with the following extra property. The interval of each leaf node has the same length; i.e., covers equal range of values. Formally, let S be the sorted RDF set resulted from D, for each leaf node its interval has length ρ, where $\rho = \frac{|S[1].o - S[|S|].o|}{l}$[4]. Therefore, for a leaf node n, we have that $|n.I^- - n.I^+| = \rho$. For example, for the leftmost leaf, its interval is $[S[1].o, S[1].o + \rho)$. The HETree-R is equivalently defined by providing the interval length ρ, instead of the number of leaves l.

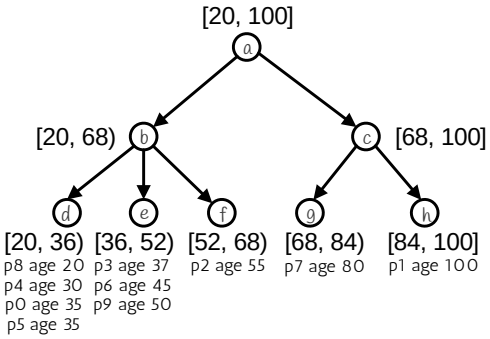

Fig. 3. A Range-based HETree (HETree-R)

Example 4. Figure 3 presents a HETree-R tree constructed by considering the set of triples D (Figrue 1), $l = 5$ and $d = 3$. As we can observe from Figure 3, each leaf node covers equal range of values. Particularly, we have that the interval of each leaf must have length $\rho = \frac{|20-100|}{5} = 16$. Hence, the leftmost leaf d has the interval $[20, 20 + 16)$. Based on the specified intervals, the d leaf node contains four triples, the e leaf three triples, while the rest leaves contain one triple. Note that, all nodes in HETree-R define closed-open intervals, except the rightmost node in each level, which defines a closed interval. ■

[4]We assume here that, there is at least one triple in D with different object value than the rest triples.

Algorithm 2. Create HETree-R (D, l, d)

Input: D: set of triples; l: number of leaf nodes; d: tree degree
Output: r: root node of the HETree-R tree

```
1 S ← sort D based on objects values
2 L ← ConstructLeaves-R(S, l)
3 r ← ConstructIntervalNodes(L, d)
4 return r
```

2.4.1. The HETree-R Construction

This section studies the construction of the HETree-R structure. The HETree-R is also constructed in a button-up fashion.

Algorithm 2 presents the algorithm for the HETree-R construction. The algorithm takes as input: (1) a set of RDF triples D; (2) the number of leaf nodes l; and (3) the tree degree d. Similarly to Algorithm 1, after sorting the triples (*line 1*), the algorithm uses two main procedures to build the tree. The ConstructLeaves-R procedure creates the leaf nodes of the HETree-R (*line 2*); the ConstructInternalNodes procedure (*line 3*) is the same with that in the Algorithm 1 and creates the internal nodes. Finally, the HETree-R root node is returned (*line 4*).

Procedure 3 presents the ConstructLeaves-R procedure. The procedure takes as input an ordered set of RDF triples S, as well as the number of leaves nodes l. First, the procedure computes the range ρ of the leaves (*line 1*). Then, it constructs l leaf nodes (*lines 2–9*) and assigns same intervals to all of them (*lines 4–8*). Afterwards, it traverse all RDF triples in S (*lines 10–12*) and place them to the appropriate leaf node (*line 12*). Finally, it removes empty leaf nodes (*lines 13–15*) and the set of created leaves is returned (*line 16*).

Computational Analysis. The computational cost for the HETree-R construction algorithm (Algorithm 2) is again the sum of three parts. The first and the third part are the same as in Algorithm 1. Assume again that $|D| = \kappa$. The second part is the ConstructLeaves-R procedure, which requires $O(l + \kappa + l)$. Since we have that $\kappa > l$, then $O(l + \kappa + l) = O(\kappa)$. Using the computational costs for the first and the third part from Section 2.3.1, we have that in worst case, the overall

Procedure 3: ConstructLeaves-R(S, l)

Input: S: ordered set of triples; l: number of leaf nodes
Output: L: ordered set of leaf nodes

$\rho \leftarrow \frac{|S[1].o - S[|S|].o|}{l}$
for $i \leftarrow 1$ **to** l **do**
  create an empty leaf node n
  if $i = 1$ **then**
    $n.I^- \leftarrow S[1].o$
  else
    $n.I^- \leftarrow L[i-1].I^+$
  $n.I^+ \leftarrow n.I^- + \rho$
  $L[i] \leftarrow n$
for $t \leftarrow 1$ **to** $|S|$ **do**
  $j \leftarrow \left\lfloor \frac{S[t].o - S[1].o}{\rho} \right\rfloor + 1$
  insert triple $S[t]$ into leaf node $L[j]$
for $i \leftarrow 1$ **to** l **do**
  if $L[i]$ *does not contain triples* **then**
    remove $L[i]$;
return L

computational cost for the HETree-R construction is $O(\kappa log\kappa + \kappa + \kappa) = O(\kappa log\kappa) = O(|D|log|D|)$.

2.5. Setting the HETree Parameters

In our working scenario, the user specifies the parameters required for the HETree construction (e.g., number of leaves l). In this section, we describe our approach for automatically calculating the HETree parameters based on the input data, when no user preferences are provided. Our goal is to derive the parameters by the input data, such that the resulting HETree can address some basic guidelines set by the visualization environment. In what follows, we discuss in details the proposed approach.

An important parameter in hierarchical visualizations is the minimum and maximum number of objects that can be effectively rendered in the most detailed level[5]. In our case, the above numbers correspond to the number of triples contained in the leaf nodes. The proper calculation of these numbers is crucial such that the resulted tree avoids overloaded and scattered visualizations.

Therefore, in HETree construction, our approach considers the minimum and the maximum number of triples per leaf node, denoted as λ_{min} and λ_{max}, respectively. Besides the number of objects rendered in the lowest level, our approach consider full m-ary trees, such that a more "uniform" structure (i.e., all the groups are divided into same number of groups) is resulted. The following example illustrates our approach to calculate the HETree parameters.

[5]Similar bounds can also be defined for other tree levels.

Table 1
Number of leaf nodes for full balanced m-ary trees

	Degree				
Height	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6
2	4	9	16	25	36
3	8	27	64	625	216
4	16	81	256	3125	1296
5	32	243	1024	15625	7776
6	64	729	4048	78125	46656

Example 5. Assume that, based on the adopted visualization technique, the ideal number of data objects to be rendered in the lowest level in the screen, is between 25 and 50. Hence, we have that $\lambda_{min} = 25$ and $\lambda_{max} = 50$.

Now, let's assume that we want to visualize the RDF triples set D_1, using a HETree-C, where $|D_1| = 500$. Based on the number of triples and the λ bounds, we can estimate the bounds for the number of leaves. Let l_{min} and l_{max} denote the minimum and the upper bound for the number of leaves. Therefore, we have that $\frac{|D_1|}{\lambda_{max}} \leq l \leq \frac{|D_1|}{\lambda_{min}} \Leftrightarrow \frac{500}{50} \leq l \leq \frac{500}{25} \Leftrightarrow 10 \leq l \leq 20$.

Hence, our HETree-C should have between $l_{min} = 10$ and $l_{max} = 20$ leaf nodes. Since, we are adopting full m-ary trees, from Table 1 we can indicate the tree characteristics that conform with the number of leaves guideline. The candidate settings (i.e., leaf number and degree) are indicated in Table 1, using dark-grey colour. The setting with $d = 2$, is rejected since is considered an "extreme" choice for a visualization scenario. Note that, in our work, in any case we assume settings with $d \geq 3$ and $height \geq 2$. Therefore, a HETree-C with $l = 16$ and $d = 4$ is a suitable structure for our case.

Now, let's assume that we want to visualize the RDF triples set D_2, where $|D_2| = 1000$. Following a similar approach, we have that $20 \leq l \leq 40$. The candidate settings are indicated in Table 1 using light-grey colour. Also, here the setting with $d = 2$, is rejected. Hence, we have the following settings

that satisfy the considered guideline: (1) $l = 27$, $d = 3$; (2) $l = 25$, $d = 5$; and (3) $l = 36$, $d = 6$.

In case, where more than one settings satisfy the considered guideline, we select the preferable one according to following set of rules. From the candidate settings, we prefer the setting which results in the highest tree (*1st Criterion*). In case that the highest tree is constructed by more than one settings, we consider the distance c, between l and the centre of l_{min} and l_{max} (*2nd Criterion*); i.e., $c = |l - \frac{l_{min}+l_{max}}{2}|$. The setting with the lowest c value is selected. Note that, based on the visualization context, different criteria and preferences may be followed.

In our example, from the candidate settings, the setting (1) is selected, since it will construct the highest tree (height=3, Table 1). On the other hand, the settings (2) and (3) will construct trees with lower heights (height=2).

Now, assume a scenario where only the settings (2) and (3) are candidates. In this case, since both settings result to trees with equal heights, the *2nd Criterion* is considered. Hence, for the (2) setting we have $c_2 = |25 - \frac{20+40}{2}| = 5$. Similarly, for the (3) setting $c_3 = |36 - \frac{20+40}{2}| = 6$. Therefore, between the settings (2) and (3), the setting (2) is preferable, since $c_2 < c_3$.

In case of HETree-R, a similar approach is followed, assuming normal distribution over the values of the triples objects. ■

2.6. Statistics Computations over HETree

Data statistics is a crucial aspect in the context of hierarchical visual exploration and analysis. Statistical informations over groups of objects offer rich insights on the underlying data. In this way, useful information regarding different set of objects with common characteristics is provided. Additionally, this information may also guide the users through their navigation over the hierarchy.

In this section, we present how statistics computation is performed over the nodes of the HETree. Statistics computations exploit two main aspects of the HETree structure: (1) the internal nodes cover (aggregate) their children nodes; and (2) the tree is constructed in bottom-up fashion. Statistics computation is performed during the tree construction; for the leaf nodes, we gather statistics from the triples they contain, whereas for the internal nodes we aggregate the statistics of their children.

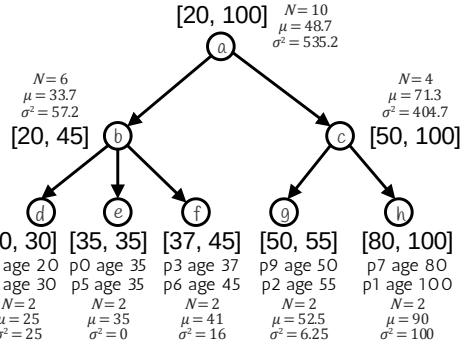

Fig. 4. Statistics computation over HETree

For simplicity, here, we assume that each node contains the following extra fields, used for simple statistics computations, although more complex or RDF-related (e.g., most common subject, subject with the minimum value, etc.) statistics can be computed. Assume a node n, as $n.N$ we denote the *number* of RDF triples covered by n; as $n.\mu$ and $n.\sigma^2$ we denote the mean and the variance of the triples objects values covered by n, respectively. Additionally, we assume the minimum and the maximum values, denoted as $n.min$ and $n.max$, respectively.

Statistics computations can be easily performed in the construction algorithms (Algorithms 1 & 2) without any modifications. The follow example illustrates these computations.

Example 6. In this example we assume the HETree-C presented in Figure 2. Figure 4 shows the HETree-C with the computed statistics in each node. When all the leaf nodes have been constructed, the statistics for each leaf is computed. For instance, we can see from Figure 4, that for the rightmost leaf h we have: $h.N = 2$, $h.\mu = \frac{80+100}{2} = 90$ and $h.\sigma^2 = \frac{1}{2} \cdot ((80-90)^2 + (100-90)^2) = 100$. Also, we have $h.min = 80$ and $h.max = 100$. Note that, in the HETree-C case, for each node, the min and max correspond to its interval bounds. Following the above process, we compute the statistics for all leaf nodes.

Then, for each parent node we construct, we compute its statistics using the computed statistics of its children nodes. Considering the c internal node, with the children nodes g and h, we have that $c.min = 50$ and $c.max = 100$. Also, we have that $c.N = g.N + h.N = 2 + 2 = 4$. Now

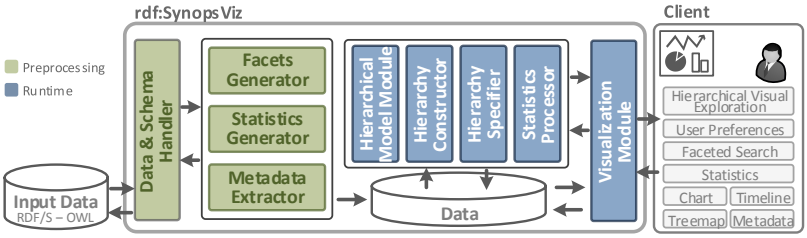

Fig. 5. System architecture

we will compute the mean value by combining the children mean values: $c.\mu = \frac{g.N \cdot g.\mu + h.N \cdot h.\mu}{g.N + h.N} = \frac{2 \cdot 52.5 + 2 \cdot 90}{2+2} = 71.3$. Similarly, for variance we have $c.\sigma^2 = \frac{g.N \cdot g.\sigma^2 + h.N \cdot h.\sigma^2 + g.N \cdot (g.\mu - c.\mu)^2 + h.N \cdot (h.\mu - c.\mu)^2}{g.N + h.N} = \frac{2 \cdot 6.25 + 2 \cdot 100 + 2 \cdot (52.5 - 71.3)^2 + 2 \cdot (90 - 71.3)^2}{2+2} = 404.7$.

The similar approach is also followed for the case of HETree-R. ■

3. The rdf:SynopsViz Tool

Based on the proposed hierarchical model, we have developed a web-based prototype called *rdf:SynopsViz*[6]. The key features of rdf:SynopsViz are summarized as follows: (1) It supports the aforementioned *hierarchical model* for RDF data visualization, browsing and analysis. (2) It offers *automatic* on-the-fly hierarchy construction, as well as *user-defined* hierarchy construction based on user's preferences. (3) Provides *faceted* browsing and filtering over classes and properties. (4) Integrates *statistics with visualization*; visualizations have been enriched with useful statistics and data information. (5) Offers several visualizations techniques (e.g., timeline, chart, treemap). (6) Provides a large number of dataset's *statistics* regarding the: *data-level* (e.g., number of sameAs triples), *schema-level* (e.g., most common classes/properties), and *structure level* (e.g., entities with the larger in-degree). (7) Provides numerous *metadata* related to the dataset: licensing, provenance, linking, availability, undesirability, etc. The latter can be considered useful for assessing data quality [75].

In the rest of this section, Section 3.1 describes the system architecture, Section 3.2 demonstrates the basic functionality of the rdf:SynopsViz. Finally, Section 3.3 provides technical information about the implementation.

[6]synopsviz.imis.athena-innovation.gr

3.1. System Architecture

The architecture of rdf:SynopsViz is presented in Figure 5. Our scenario involves three main parts: the Client GUI, the rdf:SynopsViz, and the Input data. The *Client* part, corresponds to the system's front-end offering several functionalities to the end-users. For example, hierarchical visual exploration, facet search, etc. (see Section 3.2 for more details). rdf:SynopsViz consumes RDF data as *Input data*; optionally, OWL-RDF/S vocabularies/ontologies describing the input data can be loaded. Next, we describe the basic components of the rdf:SynopsViz.

In the preprocessing phase, the *Data and Schema Handler* parses the input data and inferes schema information (e.g., properties domain(s)/range(s), class/property hierarchy, type of instances, type of properties, etc.). *Facets Generator* generates class and property facets over input data. *Statistics Generator* computes several statistics regarding the schema, instances and graph structure of the input dataset. *Metadata Extractor* collects dataset metadata. Note that, the model construction does not require any preprocessing, it is performed online, according to user interaction.

During runtime the following components are involved. *Hierarchy Specifier* is responsible for managing the configuration parameters of our hierarchy model, e.g., the number of hierarchy levels, the number of nodes per level, and providing this information to the Hierarchy Constructor. *Hierarchy Constructor* implements our tree structure. Based on the selected facets, and the hierarchy configuration, it determines the hierarchy of groups and the contained triples. *Statistics Processor* computes statistics about the groups included in the hierarchy. *Visualization Module* allows the interaction between the user and the back-end, allowing several operations (e.g., navigation, filtering, hierarchy specification) over the visualized data. Finally, the *Hierarchical Model Module* maintains the in-memory tree structure for our model and communicates with the Hierarchy Constructor for

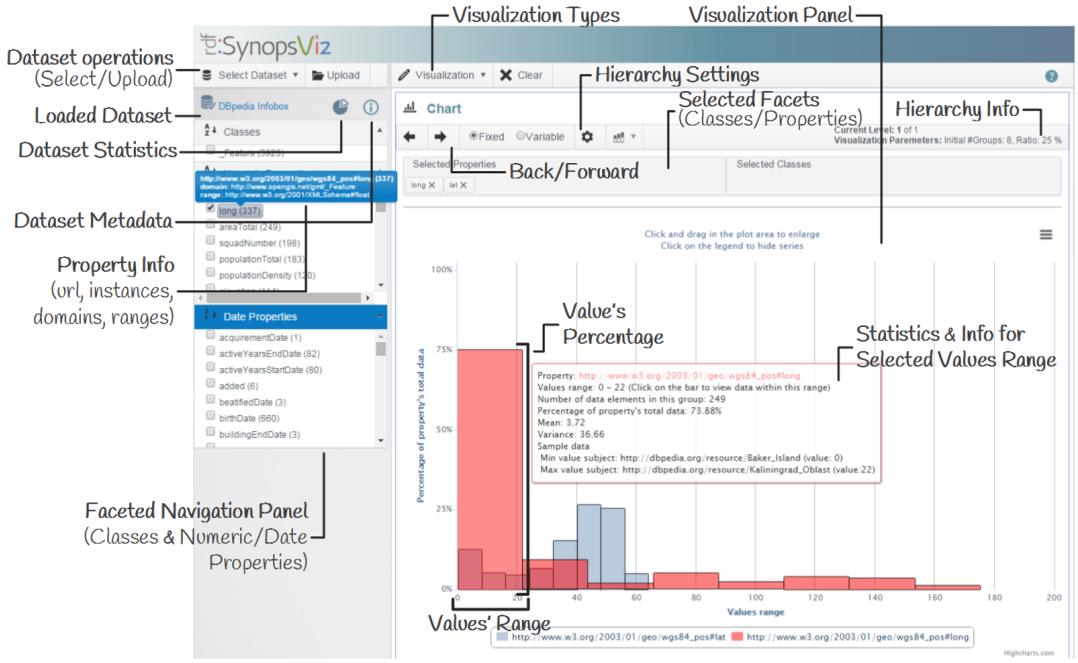

Fig. 6. Web user interface

the model construction, the Hierarchy Specifier for the model customization, the Statistics Processor for the statistics computations, and the Visualization Module for the visual representation of the model.

3.2. *rdf:SynopsViz In-Use*

In this section we outline the basic functionality of rdf:SynopsViz prototype. Figure 6 presents the web user interface of the main window. rdf:SynopsViz UI consists of the following main panels: *Facets panel*: presents and manages facets on classes and properties; *Input data control panel*: enables the user to import and manage input datasets; *Visualization panel*: is the main area where interactive charts and statistics are presented; *Configuration panel*: handles visualization settings.

Initially, users are able to select a dataset from a number of offered real-word LD datasets (e.g., DBpedia, Eurostat) or upload their own. Then, for the selected dataset, the users are able to examine several of the *dataset's metadata*, and explore several *datasets's statistics*.

Using the *facets panel*, users are able to navigate and *filter* data based on classes, numeric and date properties. In addition, through facets panel several information about the classes and properties (e.g., number of instances, domain(s), range(s), IRI, etc.) are provided to the users through the UI.

Users are able to visually explore data by considering properties' values. Particularly, *area charts* and *timeline-based area charts* are used to visualize the resources considering the user's selected properties. Classes' facets can also be used to *filter* the visualized data. Initially, the top level of the hierarchy is presented providing an *overview* of the data, organized into top-level groups; the user can interactively *drill-down* (i.e., zoom-in) and *roll-up* (i.e., zoom-out) over the group of interest, up to the actual values of the raw input data (i.e., RDF triples). At the same time, statistical information concerning the hierarchy groups as well as their contents (e.g., mean value, variance, sample data, range) are presented through the UI (Figure 6). Regarding the most detailed level (i.e., RDF triples), several visualization types are offered; e.g., area, column, line, spline, areaspline, etc. (Figure 7).

In addition, users are able to visually explore data, through class hierarchy. Selecting one or more classes, users can interactively navigate over the class hierarchy using treemaps. Properties' facets can also be used to filter the visualized data. In rdf:SynopsViz the treemap visualization has been enriched with schema and statistical information. For each class, schema metadata (e.g., number of instances, sub-

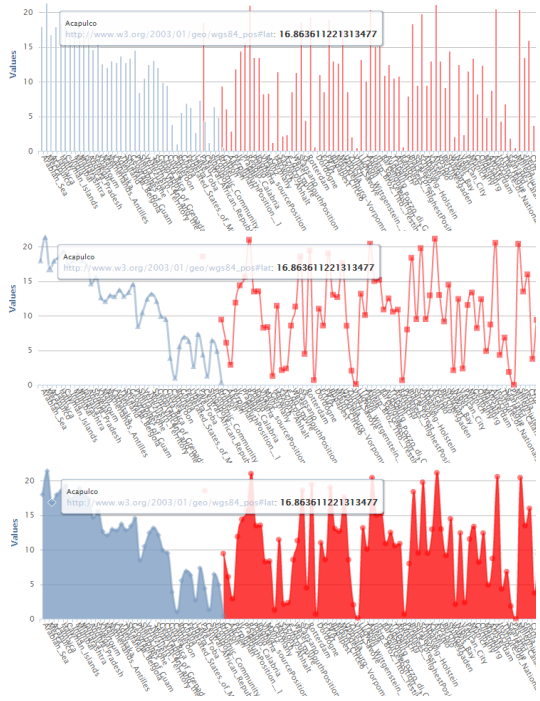

Fig. 7. RDF data object (i.e., most detailed level) visualization example

classes, datatype/object properties) and statistical information (e.g., the cardinality of each property, min, max value for datatype properties) are provided (Figure 8).

Finally, users can interactively modify the hierarchy specifications. Particularly, they are able to increase or decrease the level of abstraction/detail presented, by modifying both the number of hierarchy levels, and number of nodes per level.

A video presenting the basic functionality of our prototype is available at `youtu.be/n2ctdH5PKA0`. Finally, a demonstration of rdf:SynopsViz tool is presented in [14].

3.3. Implementation

rdf:SynopsViz is implemented on top of several open source tools and libraries. The back-end of our system is developed in Java, Jena framework[7] is used for RDF data handing and Jena TDB is used for disk-based RDF storing. The front-end prototype, is developed using HTML and Javascript. Regarding visualization libraries, we use Highcharts[8], for the area, column, line, spline, areaspline and timeline-based charts and Google Charts[9] for treemap and pie charts.

[7] jena.apache.org
[8] www.highcharts.com

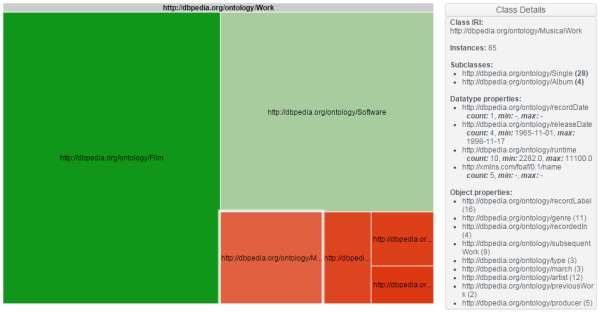

Fig. 8. Treemap enriched with schema & statistical information

4. Related Work

This section reviews related work on LD visualization and exploration (Section 4.1), LD statistical analysis (Section 4.2) and finally hierarchical visualization techniques (Section 4.3).

4.1. Linked Data Visualization & Exploration

A large number of works studying issues related to LD visual exploration and analysis have been proposed in the literature [23,53]. In what follows, we classify these works into six categories: (1) Browsers, (2) Generic visualization tools, (3) Domain, vocabulary & environment-specific visualization tools, (4) Graph-based visualization tools, (5) Ontology visualization tools, and (6) Visualization libraries.

Browsers. *LD browsers* have been the first tools developed for LD utilization and analysis. Similarly to the traditional ones, LD browsers provide the functionality for link navigation and user-friendly representation of LD resources and their properties; thus enabling browsing and exploration of LD in a most intuitive way. LD browsers mainly use tabular views and links to provide navigation over the LD resources. In this category we can mention the following LD browsers, *VisiNav* [36], *LESS* [7], *Explorator* [3], *Tabulator* [13], *Noadster* [59], *Haystack* [57], *Dipper*[10], *Disco*[11], *graphite*[12], *Information Workbench*[13], *mar-*

[9] developers.google.com/chart
[10] api.talis.com/stores/iand-dev1/items/dipper.html
[11] www4.wiwiss.fu-berlin.de/bizer/ng4j/disco
[12] graphite.ecs.soton.ac.uk/browser
[13] iwb.fluidops.com

bles[14], *OpenLink Data Explorer*[15], *URI Burner*[16] and *Zitgist*[17].

Generic Visualization Tools. In the context of LD visual exploration, there is a large number of *generic visualization frameworks*, that offer a wide range of visualization types and operations. In what follows we outline the most widely known tools in this category.

Rhizomer [19] provides LD exploration based on a overview, zoom and filter workflow. Rhizomer offers various types of visualizations such as map, timeline, treemap and chart. *Payola* [45] is a generic framework for LD visualization and analysis. The framework offers a variety of domain-specific analysis and visualization plugins (e.g., graphs, tables, etc.). In addition, Payola offers collaborative features in which users can create and share analyzers. Finally, in Payola the visualizations can be customized according to ontologies used in the resulting data. The *Linked Data Visualization Model* (LDVM) [18] provides an abstract visualization process for LD datasets. LDVM allows to connect different datasets with various kinds of visualizations in a dynamic way. The visualization process follows a four stages workflow: Source data, Analytical abstraction, Visualization abstraction, and View. LDVM considers several visualizations techniques, e.g., circle, sunburst, treemap, etc. *Balloon Synopsis* [62] provides an easy-to-use LD visualizer based on HTML and JavaScript. It adopts a node-centric visualization approach in a modern tile design. Finally, it supports automatic information enhancing, similarity analysis and ontology templates. *Vis Wizard* [72] is a Web-based visualization tool which exploits data semantics to simplify process of setting up visualizations. In addition, Vis Wizard is able to analyse multiple datasets using brushing and linking methods. Similarly, *Linked Data Visualization Wizard* (LDVizWiz) [4] provides a semi-automatic way for the production of possible visualization for LD datasets. *SemLens* [39] is a visual tool that combines scatter plots and semantic lenses, offering visual discovery of correlations and patterns in data. Objects are arranged in a scatter plot and are analysed using user-defined semantic lenses. *Hide the stack* [24] proposes an approach for visualizing LD for mainstream end-users. Semantic Web technologies (e.g., RDF, SPARQL) are utilized, but are "hidden" from the end-users. Particularly, a template-based visualization approach is adopted, where the information for each resource is presented based on its rdf:type.

Domain, Vocabulary & Environment-specific Visualization Tools. Several tools have been developed to provide *domain-specific* visualization operations. For example, *Map4rdf* [50], *Facete* [68], *SexTant* [12] and *LinkedGeoData browser* [67] focus on visualizing geo-spatial data. Furthermore, *VISU* [2] consider university data.

There are also some *vocabulary-based* visualization tools, which are based on specific vocabularies, such as the following works focusing on multidimensional LD modelled in the DataCube vocabulary: *CubeViz* [30], *Payola Data Cube Vocabulary* [40], *OpenCube* [43], *CSV2DataCube* [60] and *Linked Data Cubes Explorer*[18]. Similarly, *FOAF.vix*[19] and *FoaF Explorer*[20] target the FOAF vocabulary.

Finally, there are efforts for tools that are designed for specific environments. For example, *Who's Who* [21] and *DBpedia Mobile* [9] focus on mobile environments.

Graph-based Visualization Tools. A large number of tools visualize LD datasets adopting a *graph-based* (a.k.a., node-link) approach. In this category, we can mention *IsaViz* [56], *Fenfire* [37], *Lodlive* [20], *RelFinder* [38], *LODWheel* [70], *Trisolda* [26,27], *PGV* [25], *LODeX* [11], *Welkin*[21], *RDF-Gravity*[22], *VisualRDF*[23] and *RDF graph visualizer* [61].

Ontology Visualization Tools. The problem of *ontology visualization and exploration* have been extensively studied for many years [32,28,35,49,44]. In existed works, several approaches have been adopted. For example, *KC-Viz* [54], *VOWL2* [51], *GLOW* [42], *SOVA* [15], *FlexViz* [31] and *GrOWL* [47] follow the node-link paradigm, *CropCircles* [74] uses geometric containment, *Knoocks* [46] combines containment-based and node-link approaches and *OntoTrix* [8] uses node-link and adjacency matrix representations.

Visualization Libraries. Finally, there is a variety of Javascript libraries which allow LD visualiza-

[14] mes.github.io/marbles
[15] lod.openlinksw.com/ode
[16] linkeddata.uriburner.com
[17] dataviewer.zitgist.com
[18] km.aifb.kit.edu/projects/ldcx
[19] foaf-visualizer.gnu.org.ua
[20] xml.mfd-consult.dk/foaf/explorer
[21] simile.mit.edu/welkin
[22] semweb.salzburgresearch.at/apps/rdf-gravity
[23] github.com/alangrafu/visualRDF

tions to be embedded in Web pages. *Sgvizler* [66] is a JavaScript wrapper for visualizing SPARQL results. Sgvizler allows users to specify SPARQL Select queries directly into HTML elements. Sgvizler uses Google Charts to generate the output, offering numerous visualizations types such as chart, treemap, graph, timeline, ect. *Visualbox* [33] provides an environment where users can build and debug SPARQL queries in order to retrieve LD; then, a set of visualization templates is provided to visualize results. Visualbox uses several visualization libraries like Google Charts and D3 [16], offering 14 visualization types.

In contrast to the aforementioned approaches, our work does not focus solely on proposing techniques for LD visualization. Instead, we introduce a model for building, visualizing and interacting with hierarchically organized numeric and temporal LD. The proposed model can be adopted by the existing visualization techniques, in order to offer hierarchical visualizations. Additionally, we present a prototype system that adopts the introduced hierarchical model, offering hierarchical visual exploration over LD datasets.

4.2. Linked Data Statistical Analysis

Statistical information can be considered particularly important in the data visual exploration and analysis context. In this respect, the following tools are considered to provide statistical analysis features of LD datasets. *RDFStats* [48] calculates statistical information about RDF datasets. *LODstats* [6] is an extensible framework, offering scalable statistical analysis of LD datasets. *RapidMiner LOD Extension* [58,55] is an extension of the data mining platform RapidMiner[24], offering sophisticated data analysis operations over LD. *SparqlR*[25] is a package of the R[26] statistical analysis platform. SparqlR executes SPARQL queries over SPARQL endpoints and provides statistical analysis and visualization over the SPARQL results.

In comparison with these tools, our work does not focus on new techniques for LD statistics computation and analysis. We are primarily interested on enhancing the visualization and user exploration functionality by providing statistical properties of the visualized datasets and objects, making use of existing computation techniques. Also, we demonstrate how in the proposed structure, computations can be performed on-the-fly and enrich our hierarchical model.

4.3. Hierarchical Visualization Techniques

The wider area of data and information visualization has provided a variety of approaches for hierarchical analysis and presentation of large datasets. In this section we present the most relevant to our approach.

Treemaps [63] visualize tree structures using a space-filling layout algorithm based on recursive subdivision of space. Rectangles are used to represent tree nodes, the size of each node is proportional to the cumulative size of its descendant nodes. Additionally, several treemaps variations have been proposed. For example, *Cushion Treemaps* [73] and *Squarified Treemaps* [17] use shades in order to provide insight in the hierarchical structure. *Ordered Treemaps* [65] ensures that items near each other in the given (i.e., input) order, will be near each other in the treemap layout. Finally, *Quantum Treemaps* [10] have been proposed for laying out images within the generated rectangles.

Moreover, there is some graph-based (a.k.a. node-link) hierarchical visualization techniques. These techniques generate hierarchies over graphs by hierarchically aggregate graph's nodes and/or edges. *ASK-GraphView* [1] and *Tulip* [5] use clustering techniques to cluster nodes into a visual aggregate hierarchy. *NodeTrix* [41] combines graphs with adjacency matrices. Subsets of a graph are clustered and rendered as adjacency matrices connected to other matrices.

In the context of online analytical processing (OLAP), there are some approaches that provide hierarchical visual exploration. [52] proposes a class of OLAP-aware hierarchical visual layouts; similarly, [71] uses OLAP-based hierarchical stacked bars. *Polaris* [69] offers visual exploratory analysis of data warehouses with rich hierarchical structure. Finally, as aforementioned, some of the techniques proposed in the context of ontology visualization (Section 4.1) follow a hierarchy-based model; e.g., CropCircles, Knoocks, etc.

In contrast to above approaches, our work does not introduce a new hierarchical visualization technique, instead it proposes a model that can be adopted by the existing non-hierarchical visualization techniques, in order to provide multilevel visualizations. Finally, compared to the OLAP-based approaches, we propose a generic model for the hierarchical organization of numeric and temporal data in the LD context.

[24] rapidminer.com
[25] cran.r-project.org/web/packages/SPARQL/index.html
[26] www.r-project.org

Table 2
Numeric Properties Characteristics

Property	#Triples	Range Type	Min	Max	Mean	Variance	Dataset
areaWater	58	xsd:double	3000	1.2E+11	3.1E+09	2.3E+20	DBpedia Infobox
areaLand	66	xsd:double	3.8E+06	1.3E+12	8.2E+10	5.2E+22	DBpedia Infobox
length	72	xsd:double	1	3.7E+06	2.4E+05	3.8E+11	DBpedia Infobox
runtime	80	xsd:double	210	1.3E+04	5.4E+03	8.0E+06	DBpedia Infobox
elevation	114	xsd:double	1	4.1E+03	5.9E+02	8.8E+05	DBpedia Infobox
populationDensity	120	xsd:double	0	3.3E+06	3.1E+04	9.0E+10	DBpedia Infobox
populationTotal	183	xsd:integer	162	3.9E+09	2.8E+07	8.4E+16	DBpedia Infobox
squadNumber	198	xsd:integer	1	9.4E+01	2.3E+01	3.6E+02	DBpedia Infobox
areaTotal	249	xsd:double	2.0E+05	2.2E+13	2.0E+11	2.1E+24	DBpedia Infobox
obsValue	960	xsd:decimal	3	1.0E+02	4.8E+01	4.0E+02	Eurostat

Table 3
Temporal Properties Characteristics

Property	#Triples	Range Type	Min	Max	Dataset
releaseDate	62	xsd:date	08/02/1922	28/12/2007	DBpedia Infobox
activeYearsStartDate	80	xsd:date	30/01/1648	25/09/1995	DBpedia Infobox
activeYearsEndDate	82	xsd:date	14/10/1768	03/01/2011	DBpedia Infobox
foundingDate	114	xsd:date	18/06/618	10/06/1999	DBpedia Infobox
deathDate^{I}	519	xsd:date	04/03/306	29/02/2012	DBpedia Infobox
birthDate^{I}	660	xsd:date	08/02/412	11/12/1981	DBpedia Infobox
timePeriod	960	xsd:date	01/01/2010	01/01/2010	Eurostat
deathdate^{P}	3505	xsd:date	19/10/14	30/07/2012	DBpedia Persondata
birthDate^{P}	4396	xsd:date	31/08/12	05/03/2002	DBpedia Persondata

5. Experimental Evaluation

In this section we present the results of the evaluation we have conducted on our system. The goal is to study the performance of the proposed model, as well as the behaviour of our tool in terms of response time, using LD datasets.

Section 5.1 describes evaluation setup and Section 5.2 presents the datasets used. Then, Section 5.3 outlines the evaluation scenario, and finally Section 5.4 presents the evaluation results.

5.1. Setting

Our backend system is hosted on a quad-core server with 4GB RAM, running Linux. We have used a 2GHz CPU with 4G RAM client machine, running Linux, Google Chrome and ADSL2+ internet connection, for evaluating the tool response performance on the Web.

5.2. Datasets

In our experiments, we employ three real LD datasets. Particularly, we use the following two datasets from *DBpedia* 3.9[27]: *Infobox* (Info) which contains information that has been extracted from Wikipedia infoboxes; and *Persondata* (Person) which contains information about persons. Finally, we use one dataset from *Eurostat*[28] (Stat) which contains finance observations regarding business growth.

5.3. Scenario

In our evaluation scenario, the numeric and temporal properties induced in the employed datasets, are visualized using our hierarchical model. In order to study the performance of our model, we measure the time required for the tree construction, as well as the tool response time. Note that, in the evaluation we consider the properties that are contained in at least 50 triples.

We use ten numeric properties from the employed datasets. Particularity, nine of the numeric properties are included in the Info dataset, and one numeric property is from Stat dataset. Table 2 summarize the basic characteristics of the numeric properties used in the evaluation process, sorted by the number of triples. The table contains: (1) the number of triples in which the property is included (*#Triples*); (2) the data type of the objects (i.e., property's range) in these triples

[27] wiki.dbpedia.org

[28] eurostat.linked-statistics.org

Table 4
Numeric Properties Visualization using HETree Structures

Property	Tree Characteristics			HETree-C			HETree-R		
	#Leaves	Degree	Height	#Nodes	Construction Time (msec)	Response Time (msec)	#Nodes	Construction Time (msec)	Response Time (msec)
areaWater	9	3	2	13	8.7	336.5	13	8.2	343.2
areaLand	9	3	2	13	8.8	332.8	13	8.3	336.6
length	9	3	2	13	9.2	330.2	13	8.3	336.9
runtime	9	3	2	13	10.4	335.7	13	9.8	342.3
elevation	9	3	2	13	10.5	340.9	13	11.8	339.3
populationDensity	9	3	2	13	13.7	341.9	13	12.0	346.3
populationTotal	9	3	2	13	18.1	350.3	13	16.2	349.1
squadNumber	9	3	2	13	17.9	352.9	13	17.1	346.4
areaTotal	16	4	2	21	38.4	362.1	17	35.5	360.0
obsValue	81	3	4	121	69.5	404.3	110	69.6	400.6

Table 5
Temporal Properties Visualization using HETree Structures

Property	Tree Characteristics			HETree-C			HETree-R		
	#Leaves	Degree	Height	#Nodes	Construction Time (msec)	Response Time (msec)	#Nodes	Construction Time (msec)	Response Time (msec)
releaseDate	9	3	2	13	9.2	343.0	13	9.5	330.1
activeYearsStartDate	9	3	2	13	9.6	335.2	13	9.7	343.2
activeYearsEndDate	9	3	2	13	10.0	330.9	13	10.8	333.5
foundingDate	9	3	2	13	12.6	337.9	13	12.0	339.9
deathdate^{I}	27	3	3	40	46.0	377.9	34	47.9	376.1
birthDate^{I}	27	3	3	40	64.4	398.3	32	64.7	393.6
timePeriod	81	3	4	121	81.4	410.1	121	84.6	417.9
deathdate^{P}	243	3	5	364	268.2	600.9	351	267.4	599.2
birthDate^{P}	243	3	5	364	346.6	668.4	359	340.9	665.3

(*Range Type*); (3) the minimum object value (*Min*); (4) the maximum object value (*Max*); (5) the mean of the objects values (*Mean*); (6) the variance of the objects values (*Variance*); and (7) the dataset in which the property is included (*Dataset*).

Additionally, we use nine temporal properties. Particularity, six of the temporal properties are included in the Info dataset, two are included in the Person dataset, and one in the Stat. Table 3 summarize the main characteristics of the employed temporal properties, sorted by the number of triples. Since the birthDate and deadDate properties are included in both Info and Person datasets, superscripts indicating the dataset are used in properties names.

The above properties, are visualized by our prototype using both HETree-C and HETree-R structures. In our experiment, we are measuring the following:

Construction Time: the time required to build the HETree structure. This time includes (1) the time for sorting the triples; (2) the time for building the tree; and (3) the time for the statistics computations.

Response Time: the time required to render the charts, starting from the time the client sends the request. This time includes (1) the *Construction Time*; (2) the communication cost between the client and server; and (3) the time required by Highchart library to create and render the charts.

Note that through user interaction, the server sends to the browser only the data required for rendering the current visualization level. Hence, when a user requests to generate a visualization we have the following workflow. Initially, our system constructs the tree; then, the data regarding the top-level groups (i.e., root node children) are sent to the browser which renders the result. Afterwards, based on user interactions (i.e., drill-down, roll-up), the server retrieves the required data from the tree and sends it to the browser. As a result, the tree is constructed the first time a visualization is requested for the given input dataset; for any further user navigation over the hierarchy, the response time includes only the communication cost and the rendering time.

In our experiments, as response time we measure the time required by our tool to provide the first response (i.e., render the top-level groups), which corresponds to the slower response in our visual exploration scenario.

The tree parameters (i.e., number of leaves, degree and height) are automatically specified following the approach described in Section 2.5. For all input datasets, the lower and the upper bound for the objects rendered in the most detailed level have been set to $\lambda_{min} = 10$ and $\lambda_{max} = 50$, respectively. Note that, differences in the tree parameters do not actually affect the performance, as the major amount of runtime is depends on sorting (in case of Construction Time) and on communications (in case of Response Time).

5.4. Results

Table 4 and Table 5 present the evaluation results regarding the numeric and the temporal properties, respectively. Each table contains the characteristics of the constructed HETree structure (i.e., number of leaves, degree, and height). Additionally, for each of HETree-C and HETree-R structure the number of nodes, the construction time and the response time are presented. Note that, the number of nodes between the HETree-C and the HETree-R may differ (e.g., obsValue), because in the latter case there were empty leaf nodes that were removed during the tree construction. The presented time measurements are the average values from 500 executions.

Regarding the time required for the construction of the HETreee structure. As we can observe from Tables 4 & 5, the performance of both HETtree structures is almost similar for all examined properties. As the number of input triples increases, the construction time slightly increases, too. From Tables 4 & 5, we can see that the *areaWater* property requires the minimum construction time (8.7 msec), in HETree-R case; while *birdDate*$^{\text{P}}$ requires the maximum time (346.6 msec). Overall, the HETree structure takes reasonable time, even for properties with 4396 triples, allowing real-time user interaction.

Furthermore, from the Tables 4 & 5, we can observe that the response time follows a similar trend as the construction time. This is expected since the communication cost, as well as the time required for rendering are almost the same for all the cases. Particularly, in our setting, the Highchart requires approximately 90 msec for creating and rendering the charts.

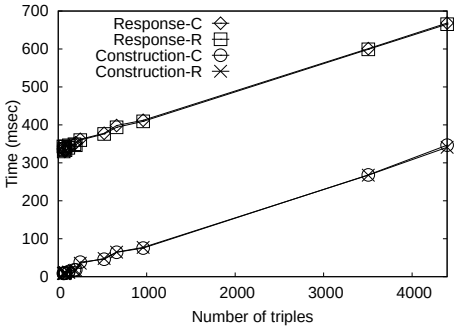

Fig. 9. Construction & Response Time w.r.t. the number of triples

From the above it seems that the response time is dominated by the communication cost. Particularly, for the properties involving a small number of triples (i.e., areaWater, 58 triples), 71% of the response time is due spent on communication cost, 27% on rendering the chart, and 2% on constructing the HETree. Similarly, for a property with a medium number of triples (i.e., squadNumber, 198 triples), 65% of response time is spent on communication cost, 25% on rendering, and 5% on constructing. Finally, for the property with the largest number of triples (i.e., birthDate$^{\text{P}}$, 4396 triples), we have 35% for communications, 13% for rendering and 52% for HETree construction. In average, considering all the properties, the 66% of the response time is spent on communications, 25% for rendering and only 9% for constructing the HETree.

Figure 9 summarizes the results from Tables 4 & 5, presenting the construction and the response time from both HETree structures w.r.t. the number of triples. In Figure 9, Response-C and Construction-C denote the response and construction time, respectively, when a HETree-C is employed. Similarly, Response-R and Construction-R stand for the HETree-R case. As we can observe, the response and the construction time are almost the same for both structures, in all cases. Additionally, the difference between the response and the construction time is almost stable for all cases. An important observation is that, in practice both the construction and the response time, and thus the overall performance of our tool grows linearly to the number of triples.

Overall, our tool exhibits a very good time performance, handling properties with more than 4K objects in less than 0.7 sec. Thus, it can offer real-time capabilities for visual exploration and analysis of large and dynamic datasets that are retrieved online by remote sites.

6. Conclusions

In this paper we have presented *HETree*, a generic model that combines user-customized hierarchical exploration with online analysis of Linked Data (LD). Our model is built on top of a lightweight tree-based structure, which can be easily constructed on-the-fly for a given set of data. We have presented two variations for constructing our model: the *HETree-C* structure organizes input data into fixed-size groups, whereas the *HETree-R* structure organizes input data into fixed-range groups. In that way the users can customize the exploration experience, allowing them to organize data into different ways, by parametrizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, etc. We have also provided a way for efficiently computing statistics over the tree, as well as a method for automatically deriving from the input dataset the best-fit parameters for the construction of the model. To this end, we have developed *rdf:SynopsViz*, a web-based prototype system, based on the introduced model, offering hierarchical visual exploration and analysis over LD. Finally, we have experimentally evaluated and demonstrated the efficiency of the presented approach over LD datasets.

Some possible insights for future work include the support of sophisticated methods for data organization in our approach (e.g., organize data in a way, so specific statistics properties hold for the resulted groups). Additionally, the extension of our approach in order to effectively handle multidimensional numeric and temporal data, as well as offer "multidimensional-based" visual exploration operations. Regarding the rdf:SynopsViz tool, we are planning to extend the graphical user interface, so our tool to be able to use data resulted from SPARQL endpoints, as well as to offer more sophisticated filtering techniques (e.g., SPARQL-enabled browsing over the data). Finally, we are interested in including more visual techniques and libraries.

References

[1] J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Trans. Vis. Comput. Graph.*, 12(5), 2006.

[2] M. Alonen, T. Kauppinen, O. Suominen, and E. Hyvönen. Exploring the linked university data with visualization tools. In *Extended Semantic Web Conference (ESWC)*, 2013.

[3] S. F. C. Araújo, D. Schwabe, and S. D. J. Barbosa. Experimenting with explorator: a direct manipulation generic rdf browser and querying tool. In *Visual Interfaces to the Social and the Semantic Web*, 2009.

[4] G. A. Atemezing and R. Troncy. Towards a linked-data based visualization wizard. In *Workshop on Consuming Linked Data*, 2014.

[5] D. Auber. Tulip - a huge graph visualization framework. In *Graph Drawing Software*, Mathematics and Visualization. 2004.

[6] S. Auer, J. Demter, M. Martin, and J. Lehmann. Lodstats - an extensible framework for high-performance dataset analytics. In *Knowledge Engineering and Knowledge Management*, 2012.

[7] S. Auer, R. Doehring, and S. Dietzold. LESS – template-based syndication and presentation of linked data. In *Extended Semantic Web Conference (ESWC)*, 2010.

[8] B. Bach, E. Pietriga, and I. Liccardi. Visualizing populated ontologies with ontotrix. *Int. J. Semantic Web Inf. Syst.*, 9(4), 2013.

[9] C. Becker and C. Bizer. Exploring the geospatial semantic web with dbpedia mobile. *J. Web Sem.*, 7(4), 2009.

[10] B. B. Bederson. Photomesa: a zoomable image browser using quantum treemaps and bubblemaps. In *ACM symposium on User interface software and technology (UIST)*, 2001.

[11] F. Benedetti, L. Po, and S. Bergamaschi. A visual summary for linked open data sources. In *International Semantic Web Conference (ISWC)*, 2014.

[12] K. Bereta, C. Nikolaou, M. Karpathiotakis, K. Kyzirakos, and M. Koubarakis. Sextant: Visualizing time-evolving linked geospatial data. In *International Semantic Web Conference (ISWC)*, 2013.

[13] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *International Semantic Web User Interaction*, 2006.

[14] N. Bikakis, M. Skourla, and G. Papastefanatos. rdf:synopsviz - a framework for hierarchical linked data visual exploration and analysis. In *Extended Semantic Web Conference (ESWC) (Demo)*, 2014.

[15] T. Boinski, A. Jaworska, R. Kleczkowski, and P. Kunowski. Ontology visualization. In *Conference on Information Technology (ICIT)*, 2010.

[16] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, 17(12), 2011.

[17] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Joint Eurographics and IEEE TCVG Symposium on Visualization*, 1999.

[18] J. M. Brunetti, S. Auer, R. García, J. Klímek, and M. Necaský. Formal linked data visualization model. In *International Conference on Information Integration and Web-based Applications & Services, (IIWAS)*, 2013.

[19] J. M. Brunetti, R. Gil, and R. García. Facets and pivoting for flexible and usable linked data exploration. In *Interacting with Linked Data Workshop*, 2012.

[20] D. V. Camarda, S. Mazzini, and A. Antonuccio. Lodlive, exploring the web of data. In *Conference on Semantic Systems (I-SEMANTICS)*, 2012.

[21] A. E. Cano, A. Dadzie, and M. Hartmann. *Who's Who* - A linked data visualisation tool for mobile environments. In *Extended Semantic Web Conference (ESWC)*, 2011.

[22] A. Dadzie, V. Lanfranchi, and D. Petrelli. Seeing is believing: Linking data with knowledge. *Information Visualization*, 8(3),

2009.

[23] A. Dadzie and M. Rowe. Approaches to visualising linked data: A survey. *Semantic Web*, 2(2), 2011.

[24] A. Dadzie, M. Rowe, and D. Petrelli. *Hide the Stack:* toward usable linked data. In *Extended Semantic Web Conference (ESWC)*, 2011.

[25] L. Deligiannidis, K. Kochut, and A. P. Sheth. RDF data exploration and visualization. In *Workshop on CyberInfrastructure: Information Management in eScience*, 2007.

[26] J. Dokulil and J. Katreniaková. Visualization of large schemaless rdf data. In *Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2007.

[27] J. Dokulil and J. Katreniaková. Using clusters in rdf visualization. In *Advances in Semantic Processing*, 2009.

[28] M. Dudás, O. Zamazal, and V. Svátek. Roadmapping and navigating in the ontology visualization landscape. In *Conference on Knowledge Engineering and Knowledge Management EKAW*, 2014.

[29] N. Elmqvist and J. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Trans. Vis. Comput. Graph.*, 16(3), 2010.

[30] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked open data statistics: Collection and exploitation. In *Knowledge Engineering and the Semantic Web*, 2013.

[31] S. Falconer, C. Callendar, and M.-A. Storey. A visualization service for the semantic web. In *Knowledge Engineering and Management by the Masses*. 2010.

[32] B. Fu, N. F. Noy, and M.-A. Storey. Eye tracking the user experience - an evaluation of ontology visualization techniques. *Semantic Web Journal (to appear)*, 2014.

[33] A. Graves. Creation of visualizations based on linked data. In *Conference on Web Intelligence, Mining and Semantics*, 2013.

[34] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM Conference on Management of Data (SIGMOD)*, 1984.

[35] F. Haag, S. Lohmann, S. Negru, and T. Ertl. Ontovibe: An ontology visualization benchmark. In *Workshop on Visualizations and User Interfaces for Knowledge Engineering and Linked Data Analytics*, 2014.

[36] A. Harth. Visinav: A system for visual search and navigation on web data. *J. Web Sem.*, 8(4), 2010.

[37] T. Hastrup, R. Cyganiak, and U. Bojars. Browsing linked data with fenfire. In *World Wide Web Conference (WWW)*, 2008.

[38] P. Heim, S. Lohmann, and T. Stegemann. Interactive relationship discovery via the semantic web. In *Extended Semantic Web Conference (ESWC)*, 2010.

[39] P. Heim, S. Lohmann, D. Tsendragchaa, and T. Ertl. Semlens: visual analysis of semantic data with scatter plots and semantic lenses. In *Conference on Semantic Systems (I-SEMANTICS)*, 2011.

[40] J. Helmich, J. Klímek, and M. Necaský. Visualizing RDF data cubes using the linked data visualization model. In *Extended Semantic Web Conference (ESWC)*, 2014.

[41] N. Henry, J. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6), 2007.

[42] W. Hop, S. de Ridder, F. Frasincar, and F. Hogenboom. Using hierarchical edge bundles to visualize complex ontologies in GLOW. In *ACM Symposium on Applied Computing, SAC*, 2012.

[43] E. Kalampokis, A. Nikolov, P. Haase, R. Cyganiak, A. Stasiewicz, A. Karamanou, M. Zotou, D. Zeginis, E. Tambouris, and K. A. Tarabanis. Exploiting linked data cubes with opencube toolkit. In *International Semantic Web Conference (ISWC)*, 2014.

[44] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. G. Giannopoulou. Ontology visualization methods - a survey. *ACM Comput. Surv.*, 39(4), 2007.

[45] J. Klímek, J. Helmich, and M. Necaský. Payola: Collaborative linked data analysis and visualization framework. In *Extended Semantic Web Conference (ESWC)*, 2013.

[46] S. Kriglstein and R. Motschnig-Pitrik. Knoocks: New visualization approach for ontologies. In *Conference on Information Visualisation*, 2008.

[47] S. Krivov, R. Williams, and F. Villa. Growl: A tool for visualization and editing of OWL ontologies. *J. Web Sem.*, 5(2), 2007.

[48] A. Langegger and W. Wöß. Rdfstats - an extensible RDF statistics generator and library. In *Database and Expert Systems Applications*, 2009.

[49] M. Lanzenberger, J. Sampson, and M. Rester. Visualization in ontology tools. In *International Conference on Complex, Intelligent and Software Intensive Systems, CISIS*, 2009.

[50] A. d. Leon, F. Wisniewki, B. Villazón-Terrazas, and O. Corcho. Map4rdf- faceted browser for geospatial datasets. In *Using Open Data: policy modeling, citizen empowerment, data journalism*, 2012.

[51] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Vowl 2: User-oriented visualization of ontologies. In *Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2014.

[52] S. Mansmann and M. H. Scholl. Exploring OLAP aggregates with hierarchical visualization techniques. In *ACM Symposium on Applied Computing (SAC)*, 2007.

[53] N. Marie and F. L. Gandon. Survey of linked data based exploration systems. In *Workshop on Intelligent Exploration of Semantic Data (IESD)*, 2014.

[54] E. Motta, P. Mulholland, S. Peroni, M. d'Aquin, J. M. Gómez-Pérez, V. Mendez, and F. Zablith. A novel approach to visualizing and navigating ontologies. In *International Semantic Web Conference (ISWC)*, 2011.

[55] H. Paulheim. Generating possible interpretations for statistics from linked open data. In *Extended Semantic Web Conference (ESWC)*, 2012.

[56] E. Pietriga. Isaviz: a visual environment for browsing and authoring rdf models. In *World Wide Web Conference (WWW)*, 2002.

[57] D. A. Quan and R. Karger. How to make a semantic web browser. In *World Wide Web Conference (WWW)*, 2004.

[58] P. Ristoski, C. Bizer, and H. Paulheim. Mining the web of linked data with rapidminer. In *International Semantic Web Conference (ISWC)*, 2014.

[59] L. Rutledge, J. van Ossenbruggen, and L. Hardman. Making RDF presentable: integrated global and local semantic web browsing. In *World Wide Web Conference (WWW)*, 2005.

[60] P. E. R. Salas, F. M. D. Mota, K. K. Breitman, M. A. Casanova, M. Martin, and S. Auer. Publishing statistical data on the web. *Int. J. Semantic Computing*, 6(4), 2012.

[61] C. Sayers. Node-centric rdf graph visualization, 2004. Technical Report HP Laboratories.

[62] K. Schlegel, T. Weißgerber, F. Stegmaier, C. Seifert, M. Granitzer, and H. Kosch. Balloon synopsis: A modern node-centric RDF viewer and browser for the web. In *Extended Semantic*

Web Conference (ESWC), 2014.

[63] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1), 1992.

[64] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, 1996.

[65] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *IEEE Symposium on Information Visualization (INFOVIS)*, 2001.

[66] M. G. Skjæveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *Extended Semantic Web Conference (ESWC)*, 2012.

[67] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4), 2012.

[68] C. Stadler, M. Martin, and S. Auer. Exploring the web of spatial data with facete. In *World Wide Web Conference (WWW)*, 2014.

[69] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. In *ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2002.

[70] M. Stuhr, D. Roman, and D. Norheim. Lodwheel - javascript-based visualization of RDF data. In *Workshop on Consuming Linked Data*, 2011.

[71] K. Techapichetvanich and A. Datta. Interactive visualization for OLAP. In *Computational Science and Its Applications (ICCSA)*, 2005.

[72] G. Tschinkel, E. E. Veas, B. Mutlu, and V. Sabol. Using semantics for interactive visual analysis of linked open data. In *International Semantic Web Conference (ISWC)*, 2014.

[73] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *IEEE Symposium on Information Visualization (INFOVIS)*, 1999.

[74] T. D. Wang and B. Parsia. Cropcircles: Topology sensitive visualization of OWL class hierarchies. In *International Semantic Web Conference (ISWC)*, 2006.

[75] A. Zaveri, A. M. Anisa Rula, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment methodologies for linked open data. *Semantic Web Journal (Under Review)*. Available at: *www.semantic-web-journal.net/content/quality-assessment-methodologies-linked-open-data*.