ProSWIP: Property-Based Data Access for Semantic Web Interactive Programming

Silviu Homoceanu, Philipp Wille, and Wolf-Tilo Balke

Institute for Information Systems,
Technische Universität Braunschweig, Germany

Abstract. The Semantic Web has matured from a mere theoretical vision to a variety of ready-to-use linked open data sources currently available on the Web. Still, with respect to application development, the Web community is just starting to develop new paradigms in which data as the main driver of applications is promoted to first class status. Relying on properties of resources as an indicator for the type, property-based typing is such a paradigm. In this paper, we inspect the feasibility of property-based typing for accessing data from the linked open data cloud. Problems in terms of transparency and quality of the selected data were noticeable. To alleviate these problems, we developed an iterative approach that builds on human feedback.

1 Introduction

The amount of data available on the Web has considerably increased in the last few years. Despite huge efforts in the area of the Semantic Web to make such web data machine-processable, only a few applications have been developed that can take full advantage of this data. Besides the general sparseness of semantic data, this behavior is currently explained by the different representation formalisms of semantic data and application programming languages causing a problem of data-model/programming language interoperability. Most semantic data stores provide data in Resource Description Framework (RDF) graphs or ontologies represented in the OWL Web Ontology Language. Accessing such data from state-of-the-art object oriented (OO) programming languages requires mappings from entities and ontology categories to programming structures like classes. Fortunately, similarly to Object Relational Mapping (ORM) [1] for relational databases, there are frameworks available that map RDF and/or OWL to programming structures by means of textual code generation. Well known frameworks include Jena [2] and RDFReactor [3].

However, generated code is often unintelligible, hard to customize and almost impossible to maintain. While some frameworks make customization and maintainability more convenient by including support for IDEs, compile-time meta-programming [4] represents a better technique to cope with the interoperability problem. With compile-time meta-programming, developers can programmatically generate required classes instead of providing them directly into the source code. One such approach was recently presented by Microsoft as a feature of F# 3.0 (http://msdn.microsoft.com

/en-us/library/hh156509.aspx). Called type provider, it's a component providing types, properties and methods for an external data source without having to write these types manually for each application. Type providers seem promising for accessing data from single data stores. But because each data source may have its own vocabulary, an RDFS type provider for the Linked Open Data cloud (LOD) would be useless without proper cleansing. With no global ontology to drive the cleansing and alternative solutions like automatic ontology alignment offering just average quality, such a type provider would require manual mapping during application development.

When writing an application, software engineers have some mental representation of "things" that are required for the application. It is common knowledge in cognitive psychology (imported in information science [5]) that concepts take the place of thoughts. They are represented through symbols (words, sounds, etc.), defined intensionally by a set of properties, and extensionally by a set of entities. The goal in programming with web data is to easily access the entities that correspond to a concept the software engineer thinks of. This concept may easily be expressed by its symbol, a word label. In the LOD cloud, entities are associated with concept labels by means of the rdf:type property. Detrimental to our purpose, types are provided in different granularities, e.g., Movie, Animation, FrenchFilm, etc. We found about 1,700 entity types for movies in the LOD cloud. Furthermore, for some entities, no type is provided. In consequence, accessing entities through their type labels is difficult.

We believe that a property-based data access model as recently sketched in [6, 7] is more suitable for programming with semantic data. The type information for such a programming approach is given by properties: A type is defined by a set of required properties, and every entity with at least those properties is part of that type. When designing an application, during the data modeling phase, developers usually think in terms of entities – no clear cut types, but concepts like Movie, Actor, etc. When writing code, these concepts are bound to properties which are required for the program logic. This way, concepts are extended to a minimal intensional definition comprising core properties (e.g. Movie ≡ {Title, Genre, Director}) required by the program. Similar to the case of structural subtyping or DuckTyping [8], this definition is used to identify entities that belong to the concept (in this case all entities providing values for Title, Genre, Director are considered to be Movies). We inspected the feasibility of such a programming paradigm for accessing data from the LOD cloud. Our experiments show that simple property-based data access can lead to selecting all kinds of entities. For example Music, Video-Games, and Books were also selected when trying to access Movies. The quality of the selected data is poor if properties describing the intended concept are not well chosen. Based on this observation, we propose ProSWIP (Property-based Semantic Web Interactive Programming), an approach which empowers property-based data access while maintaining quality under control. Part of a cloud-based centralized service for programming the Semantic Web, ProSWIP will be accessible from IDEs by means of plugins. Starting from properties provided by application developers, ProSWIP estimates the quality of the selected data and if necessary, identifies additional properties that have high positive impact on the quality. In an iterative process, it assists developers to extend the property-based type definitions while checking that the extended definition still matches their intentions.

The contribution of this paper can be summarized as follows: An extensive inspection of the property-based paradigm's feasibility for accessing data from the LOD cloud; the presentation and evaluation of a quality metric enabling transparency for this programming paradigm; and the presentation and evaluation of a property selection method for better data quality.

2 Property-Based Data Access - Use Case

To assess the feasibility of the property-based paradigm for accessing data from the Web, we conducted an experiment focused on developing applications related to movies: When writing such applications developers rely on variables that represent movie properties. These properties are used in the property-based paradigm as filters so that all entities from the LOD cloud which have values for those properties are considered to represent movies. By inspecting the selected entities to identify those that actually are movies, we get an impression of the quality of the property-based paradigm. But first, what are the properties developers require for programming applications concerning movies? We conducted an extensive analysis (involving about 6% of the pages on the Web) to find properties typically associated with movies.

Motivated by the improved Web visibility promised by rich snippets, application developers started to adopt the vocabulary provided by schema.org to semantically annotate data published on the Web. Schema.org was launched in 2011 as joint initiative of major search engine providers like Bing, Google, Yahoo and Yandex to provide a unified set of vocabularies which web masters and application developers can use to semantically annotate data published on the Web. The goal of the project was to ultimately empower semantic Web search. Currently, schema.org provides a collection of 406 hierarchically built schemata for various concepts ranging from organizations, persons and events to creative works like movies, music or books. On average, schemata comprise about 34 attributes representing properties of the corresponding concepts. Movie (http://schema.org/Movie), with a total of 62 attributes, belongs to the fewer schemata that are described in more detail. While any subset of these properties can theoretically be used by application developers to refer to the Movie concept, some properties may be preferred: To establish which of the 62 properties are mostly being used when referring to movies, we analyzed a crawl of 870 million websites. Known as ClueWeb12 (http://boston.lti.cs.cmu.edu/clueweb12/) this Web crawl is publicly available as a corpus and consists of only English language sites, which have been crawled between February and May 2012. More than a year after schema.org was introduced, only about 1.56% of the web sites from ClueWeb12 comprised data that was annotated with schema.org. Overall, only 192 schemata out of 406 from schema.org were used for annotating data. On average, annotations comprised 4.6 properties. For movies, we observed about 40,000 annotations. In Table 1 we present a list of the properties most frequently used for annotating movie data. For movies, annotations comprised on average 4.5 properties, with a minimum of 1 and a maximum of 13 properties. These numbers are rather low considering that the Movie schema comprises 62 properties. This observation is not particular to movies but has

Table 1. Top "Movie" properties (with frequency above 30%) from schema.org frequently used for annotating movie data on Web pages from the ClueWeb12 corpus

Property	Movies annotated with property
Title	78 %
Description	56 %
URL	44 %
Director	39 %
Genre	38 %
Actors	38 %
AggregateRating	33 %

been made for other schemata like events or organizations as well, indicating that the property-based approach may suffer from under-specification.

Assuming that, in part, annotated data published on the Web surfaced as a result of some Web application, most developers require on average 3 to 5 properties. These properties are most likely the ones that have frequently been annotated. For the Movie concept, the most probable property-based definitions are {Title, Description, URL}, {Title, Description, URL, Director}, etc.

According to the property-based paradigm, all entities from the LOD cloud fulfilling these properties represent movies. We rely on the Billion Triples Challenge 2012 (BTC) dataset to represent the LOD cloud. BTC comprises about 1.4 billion quads of the form (subject, predicate, object, source) crawled from major LOD data stores like Datahub, DBpedia, Freebase, and others during May and June 2012. Entities and properties are provided as unique identifiers (URIs) in the quad subjects and predicates respectively. Sources are not relevant for our approach and will be ignored in this paper. The process of selecting data for a set of properties provided in natural language works as follows: (i) Property URIs are identified for each property. For this purpose, all subjects from tuples of the form (*, rdfs:label, *p*) are selected for each property *p* (* is a wildcard that may be substituted by any URI). Synonym sets provided by WordNet or obtained through the owl:sameAs predicate are used to extend the coverage of each property (more details in Section 3.1). (ii) With *p'* as the URI of each property *p*, the entities to be selected are the set of all distinct subjects *s* for which there are tuples of the form (*s*, *p'*, *) in the BTC dataset (* is a wildcard that may be substituted by any URI or literal). An overview of the selectivity for different property sets is provided in Table 2(a). While Title, Description and URL are quite general (1,5 million entities), Director, Actors and especially Genre significantly reduce the number of relevant entities.

Precision and recall are the standard measures for evaluating the quality of retrieved information or, in our case, the quality of selected entities. Precision is for our scenario defined as the proportion of entities representing movies out of all selected entities, while recall is defined as the proportion of selected movies out of all movies present in the BTC dataset. Computing precision and recall is not trivial in this case since it requires recognizing entities that are movies. As the rdf:type property connects entities to different types that may be related to movies (e.g., Films, Animations, FrenchFilms etc.), such types are difficult to automatically map to the Movie type without a general movie taxonomy. Without claiming full completeness for this experiment, we extracted from the BTC data set a list of movie types by bootstrapping on a seed of movies from the Linked Movie Data Base (LMDB - linkedmdb.org/) and manually inspecting the resulting types. More details about this process are presented in Section 4. In total, we found 1,736 types expressing different kinds of movies. This surprisingly large number is mostly due to the very fine classification provided by YAGO. With these types we identified a total of 87,273 movies in the BTC dataset.

As shown in Table 2(b), the choice of properties has notable impact on the quality of the selected entities: Precision increases from a mere 0.02 to 0.78 by adding one single property to the definition of Movie. Precision values of 0.92 are possible if the "right" properties are chosen. Recall is, with 0.3 for the first three most frequent properties, quite low. The main reason is the sparseness of the data. This becomes extreme in the case of Genre with just a few movies having this property.

Table 2. Nr. of entities from the BTC data set fulfilling each property set (a). The corresponding precision and recall values (b).

Property Set	(a) Nr. of Entities from BTC	(b) Precision / Recall
{Title, Description, URL}	1,447,813	0.018/0.3
{Title, Description, URL, Director}	29,328	0.78/0.26
{Title, Description, URL, Director, Genre}	2,266	0.35/0.01
{Title, Description, URL, Director, Actors}	21,531	0.92/0.23

Overall, the property-based paradigm can lead to high quality/high precision entity selection if properties are well chosen. A major obstacle in the process is the lack of transparency: The application developer has no idea about the quality of the selected entities. Properties belonging to the concept definition are mandatory and values for these properties are required by the application. In consequence, none of the entities missing on any of these properties can be used. But this has a high impact on recall. Combined with the sparse nature of LOD, the more elaborate the definition, the smaller the number of selected entities. In this paper we focus on improving the quality of the selection throughout precision first, by extending the concept definition with a set of well chosen properties. We believe once high quality properties are found, we can tackle the recall problem by building on structural similarity focused on the extending properties, but leave this as the subject of future work.

3 System Description

Starting from a property-based type definition with properties expressed in natural language and a large collection of data representing facts from the LOD cloud, ProSWIP helps the user to keep data quality problems under control: Relying on a measure of property-based data homogeneity, it measures the quality of the entities that fulfill the property-based definition. If the quality is low, key properties contributing the most to better data quality are found. The user has to finally decide if those properties are part of the type or not. The definition of the intended type is extended to include the user feedback and the process is repeated until the quality reaches a satisfactory level. For this purpose, the following functionality is required: i) identify and select those entities that fulfill the property-based type definition, ii) compute the quality of a collection of entities, iii) find properties that, if added to the set of properties defining the type, significantly improve the quality of the selected data.

3.1 Property-Based Data Access

According to the property-based paradigm, the system selects all entities from the LOD cloud having all properties from a given set. But in the LOD cloud, properties are represented through URIs. Hence, a mapping between the properties in natural language and the URIs is necessary. For this mapping, we rely on the rdfs:label property, an instance of rdf:property providing a human-readable name for a resource. For better coverage, each property is automatically extended beforehand with a list of synonyms from WordNet.

Definition 1 (mapping): Given a property $p \in$ Properties, P_{SYN_p} its set of synonyms from WordNet (including p) and LOD a large set of 3-tuples of the form (subject, predicate, object), we define *map* as a function *map* : Properties $\longrightarrow \wp$(URIs) with:

$$p \longmapsto \{s \mid \exists p_i \in P_{SYN_p}: (s, \text{rdfs:label}, p_i) \in LOD\} \tag{1}$$

For some entities the rdfs:label property may be missing. Furthermore, the same property may be present in different data stores under different URIs, possibly connected to each other through the owl:sameAs property. In consequence, in a dictionary-like fashion, each property is actually mapped to a set of URIs all considered synonyms.

Mapping Expansion Algorithm:
With $\Delta_{p,1} := map(p)$ define

$$\begin{aligned}\Delta_{p,i+1} := \{s_j' \mid \exists s_j \in \Delta_{p,i}: (s_j, \text{owl:sameAs}, s_j')\\ \in LOD \lor (s_j', \text{owl:sameAs}, s_j) \in LOD\}\end{aligned} \tag{2}$$

$$Dictionary(p) := \bigcup_{i=1}^{\infty} \Delta_{p,i} \tag{3}$$

By repeatedly linking elements through synonyms, two or more properties from the definition set may end up being represented by the same set of URIs. This doesn't play any role in the process of selecting the appropriate entities but may surprise the user when accessing values for these properties. Such cases are reported to the user.

At the very core of the property-based paradigm, an entity is relevant with respect to a specific property if there is a statement or fact asserting that the entity has this property. In the context of linked open data, we define the binary relevance of an entity w.r.t. a property as a *hit* function:

Definition 2 (hit): Given some entity $e \in E$ represented by its URI, a property in natural language $p \in$ Properties and LOD defined as above, we define *hit* as a function *hit* : (URIs × Properties) → {0, 1} with:

$$hit(e,p) = \begin{cases} 1 & \text{iff } \exists\, p' \in Dictionary(p)\colon (e,p',*) \in LOD \\ 0 & otherwise \end{cases} \tag{4}$$

where * is a wildcard that may be substituted by any literal or URI.

According to the *semiotic triangle* from cognitive psychology [5], concepts are defined intensionally by a set of properties, and extensionally by a set of entities. Aiming for simple yet effective access to entities corresponding to a certain concept we define conceptual variable *types* in the sense of programming, as a set of properties that intensionally define a concept. This type definition may iteratively evolve based on user feedback. Because the user feedback may be negative w.r.t. to some properties (by negative we mean properties that all entities corresponding to the concept definitely shouldn't possess), we define a type as follows:

Definition 3 (type): Given a concept c, extensionally defined through the set of entities given by their URIs, E_c, we define the *type* of concept c denoted T_c as the set of properties $T_c = P_{c_+} \cup P_{c_-}$ with P_{c_+} the set of positive properties and P_{c_-} the set of negative properties ($P_{c_+} \cap P_{c_-} = \emptyset$), such that:

$$\begin{array}{ll} (i) & \forall e \in E_c, p \in P_{c_+}\colon hit(e,p) = 1 \\ \text{(ii)} & \forall e \in E_c, p \in P_{c_-}\colon hit(e,p) = 0 \\ (iii) & \forall e \notin E_c\ \exists p \in P_{c_+}\colon hit(e,p) = 0 \end{array} \tag{5}$$

While here all properties (initial as well as positive and negative extensions) are treated equivalently, the fact that not all properties extending the definition are required is a starting point for future work. As in the case of properties, in the LOD cloud the same entities may end up having multiple URIs. For the sake of simplicity, we refer to one entity as being uniquely identified by an URI.

More often than not, the number of properties employed to refer to some type of entity is much smaller than the number of properties that would completely define the entity type or intended concept. Actually, extensive experiments presented in Section 2 show that on average only 4.6 (out of an average of 34 existing) properties have been used to link entities to concepts. This suggests that the developer provides a sub-set of properties meant to represent the intended (to us hidden) type. This set of properties is one of the many possible super-types of the intended type. Starting from a property set that builds a type or a super-type for some concept, all entities having all these properties are selected as being relevant for the type or super-type:

Definition 4 (property-based data access): Given a set of properties $T_c = P_{c_+} \cup P_{c_-}$ representing either a type or super-type for a concept c as before, the set of entities selected according to the property-based data access paradigm E_c is the set of entity URIs that fulfill all properties from T_c:

$$E_c = \bigcap_{j=1}^{|T_c|} E_j \tag{6}$$

where $E_j = \{e | hit(e, p_j) = 1 \; if \; p_j \in P_{c_+} \wedge hit(e, p_j) = 0 \; if \; p_j \in P_{c_-}\}$

In the ideal case, for a concept c the set of properties T_c is the *type* of c (not a super-type). Then, the set of selected entities E_c also extensionally defines concept c and should perfectly satisfy the user needs. However, there is a high probability that a super-type is provided. Since the type intended by the application developer is hidden to the system and entities have no clear types, there is no trivial way for checking if the selected entities correspond to the intended concept. The developer also has no feedback whatsoever regarding how good the selected entities match the intended use. This has grave effects on the applicability of the property-based data access paradigm. Aiming for better transparency of the whole approach, in the next section we introduce a measure of quality for the selected entities.

3.2 Quality of the Selected Entities

We measure the quality of entities selected through the property-based model as a function of entity homogeneity. The basic assumption is that the application developer describes simple concepts (like “Movies” or “Books”) with all corresponding entities having the same or almost the same properties and not ad-hoc or composed concepts (like “all things having a geo-location”). Consider for example that the developer provides three properties: Title, Description and Genre. Based on these properties a set of eight entities is selected. Besides the three properties, each entity is described by other additional properties like in Table 3. Properties p_4, p_5 and p_6 may be, for instance, Duration, Actors and Director while p_7, p_8 and p_9 could represent ISBN, Pages and Editor. As you may have intuited, entities e_1, e_2, e_3 and e_4 represent movies while the remaining entities represent books. Properties in the LOD cloud may be missing. This is reflected also in this artificial example with movies e_1, e_3 and e_4 providing no values for properties p_4 and respectively p_6. Analogously, for the entities representing books. The rest of the missing values are attributed to the fact that properties p_4, p_5 and p_6 are proper to movies while p_7, p_8 and p_9 are proper to books.

More generally, starting from the set of properties, the system selects a set of entities as described in the previous section. In a relational sense, together with the union of all their corresponding properties (stop properties like rdfs:label, owl:sameAs, rdf:type, etc. are first removed) these entities form a relational schema (as in Table 3). Especially in the field of schema extraction and discovery, the number of null values has successfully been used for establishing the quality of the schema [9] – the better the schema, the fewer null values, the more homogeneous the data. Thus, if the data is homogeneous in terms of structure - their properties - these properties intensionally define a single concept. As a measure of homogeneity we measure the property-based

Table 3. On rows - the entities that are selected for the properties set $\{p_1, p_2, p_3\}$. On columns - all properties that describe any of the selected entities.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
e_1	✓	✓	✓	✗	✓	✓	✗	✗	✗
e_2	✓	✓	✓	✓	✓	✓	✗	✗	✗
e_3	✓	✓	✓	✓	✓	✗	✗	✗	✗
e_4	✓	✓	✓	✓	✓	✗	✗	✗	✗
e_5	✓	✓	✓	✗	✗	✗	✓	✓	✓
e_6	✓	✓	✓	✗	✗	✗	✓	✗	✓
e_7	✓	✓	✓	✗	✗	✗	✓	✗	✓
e_8	✓	✓	✓	✗	✗	✗	✓	✓	✗

similarity between all entities. But there is a problem: Entities may be selected from different data sources (DBpedia, LMDB, etc.). Entities with the same type and from the same source tend to share the same properties, usually due to the focus of each data store. Different sources have different sizes, and small data sources with many properties can introduce null values. These null values are artificially amplified by the size of the data source. To handle this problem, we reduce all entities having the exact same properties to just one *witness*. This way, for the example presented in Table 3, e_3 and e_4 are both represented by one witness: $w_{e_3 e_4}$ having the same properties as e_3 or e_4. The same for e_6 and e_7. The rest are their own witnesses. Based on this observation we define the quality of a set of entities as follows:

Definition 5 (quality): With the notations of T_c and E_c as above and W_c as the set of witnesses represented by URIs of entities from E_c, the quality of the selected entities is a function, $Q : \wp(\text{URIs}) \rightarrow [0, 1]$ with:

$$Q(E_c) = \frac{1}{C_2^n} \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Sim(w_i, w_j) \qquad (7)$$

$\forall w_i, w_j \in W_c, n=|W_c|$ and $sim(w_i, w_j) = \frac{|P_{w_i} \cap P_{w_j}|}{|P_{w_i} \cup P_{w_j}|}$ is the Jaccard similarity index [10]. P_{w_i} is the set of properties of w_i and P_{w_j} is the set of properties of w_j.

While the Jaccard index is most suitable for measuring structural similarity between entities, any other similarity measure may be used here.

For the example introduced in Table 3, the quality of the selected entities is 0.55. If additional information were provided, like the concept the application developer has in mind also has property p_5, or doesn't have property p_7, the entities selected by the property based model restrict to movies only (entities 1 to 4). The quality in this case increases to 0.78, the result being slightly affected by the noise (missing values) in the data. In the following subsection we present how to find properties better separating various types of entities in the result set.

3.3 Property Selection

Finding the list of properties best distinguishing different types is similar to the problem of induction of an optimal decision tree in data classification, which is a hard task. It has been shown that finding a minimal decision tree consistent with the set of labeled entities provided as data is NP-hard [11]. Consequently, greedy algorithms like the C4.5 are applied for solving this problem [12]. When it comes to selecting some property that better discriminates between different types of entities, *information gain* from the field of information theory is the standard measure for deciding the relevance of a property [13]. Generally speaking, the information gain is the change in information entropy from a prior state to a state that takes some information as given. Computing this entropy change is only possible for entities that have class labels (entity types) attached. Types are provided in the LOD cloud by means of the rdf:type property, however entities may have multiple types partly with different granularities e.g., the movie "Gangs of NewYork" has types owl:Thing, schema.org/CreativeWork, dbpedia-owl:Film, yago:VictorianEraFilms and 15 other types. For other movies, types owl:Thing, or schema.org/CreativeWork are missing. All these types are obviously related to each other but without an upper ontology or global type hierarchy, it's difficult to make use of the type property to compute the information gain.

But the type information strongly correlates with the entity properties [14]: In the example presented in Table 3, it's obvious that entities having properties Duration, Actors and Director on top of Title, Description and Genre are movies while entities having ISBN, Pages and Editor are books. The type information is latent in the properties. But the missing values for some entities, as well as the heterogeneity of data sources make it difficult to fold all movies together to just one witness – a property set representing the movie type. Actually what happens is that more witnesses, with more or less similar properties, exist for a single type. The problem of reducing similar witnesses to a dominant type is similar to the problem of dimension reduction.

Principal component analysis (PCA) is the best, in the mean-square error sense, linear dimension reduction technique [15]. In essence, PCA is a basis transformation that seeks to reduce the dimensionality of the data by finding a few orthogonal linear combinations (called principal components) of the original variables capturing the largest variance. Given E_c the set of entities selected according to the property-based data access paradigm, and W_c the set of witnesses of entities from E_c, let X be a $n \times p$ matrix, where n and p are the number of entity witnesses and the number of properties of all witnesses, respectively. Let the matrix decomposition of X be

$$X = UDV^T \tag{8}$$

$Y{=}UD$ are the principal components (PCs), where the $p \times p$ matrix U is the matrix of eigenvectors of the covariance matrix XX^T, matrix D is a $p \times n$ rectangular diagonal matrix of nonnegative real numbers on the diagonal with customary descending order, and the $n \times n$ matrix V is the matrix of eigenvectors of X^TX. The columns of V are called loadings of the corresponding principal components. Usually the first PCs (capturing the highest data variance) are chosen to represent the dominant dimensions.

For the example introduced in Table 3 (first all data is reduced to binary values and centered on the columns such that the mean of each column is equal to 0), the first PC shows the strongest variance of 1.16. The next two components show a variance of 0.2 and the rest are 0 or close to 0. With respect to the properties, the coefficients of the first PC are clustered together according to their variance (Table 4). For this example, the three property clusters that build on the most significant PC show the existence of two dominant types that differentiate in terms of properties p_4, p_5, p_6 and p_7, p_8, p_9. Showing no variance, properties p_1, p_2 and p_3 can be ignored since they belong to both dominant types.

Table 4. Property coefficients of the first three PCs

Props. \ PCs	PC_1	PC_2	PC_3
p_1	0.00	0.00	0.00
p_2	0.00	0.00	0.00
p_3	0.00	0.00	0.00
p_4	0.35	-0.71	0.00
p_5	0.50	0.00	0.00
p_6	0.35	0.71	0.00
p_7	-0.50	0.00	0.00
p_8	-0.35	0.00	0.71
p_9	-0.35	0.00	-0.71

In general, depending on the selected entity set, more PCs may be significant. To dynamically establish which of them show significant variance, we rely on the ISODATA algorithm, an automatic thresholding approach [16] that identifies thresholds in one dimensional spaces that best separate a set of data points. With the PCs that show variances above the threshold, one dimensional clusterings (agglomerative hierarchical clustering with average inter-cluster similarity) on the coefficients are built for each PC. This way each property is assigned to one cluster for each significant PC. Each set of properties belonging to the same clusters on all significant PCs are grouped together and represent abstract dominant types we will further refer to as *latent types*. For the example in Table 4, considering that only PC_1 is significant, the extracted latent types are $t' \equiv \{p_1, p_2, p_3, p_4, p_5, p_6\}$ and $t'' \equiv \{p_1, p_2, p_3, p_7, p_8, p_9\}$. With these types we can now label entities according to the property-based model. This way, e_2 will be labeled with t' and e_5 with t''. For the future, we plan to introduce a probabilistic approach to increase the labeling recall, but for now all entities missing some values are ignored in the typing process. In this manner a set of labeled entities is created. Entities that fulfill properties for multiple types (Audiobooks in the context of our example) are automatically associated with multiple labels.

With the set of labeled entities, the information gain for a property can be computed as follows:

Definition 6 (information gain): With the notations of T_c and E_c as previously defined and P_U the set of all properties of all entities from E_c, the *information gain* of a property $p \in P_U - T_c$ w.r.t. the entity selection E_c is:

$$Gain(p, E_c) = H(E_c) - \sum_{v \in \{0,1\}} \frac{|E_c|p^v|}{|E_c|} \cdot H(E_c|p^v) \tag{9}$$

where $E_c|p^v = \{e \in E_c | hit(e,p) = v\}$.

The entropy (denoted *H*) represents a measure of the amount of uncertainty in the data and is usually computed as follows:

$$H(E_c) = -\sum_{i=1}^{n} p(t_i) \log p(t_i) \tag{10}$$

where *n* represents the number of latent types and *p(t)* represents the probability (relative frequency) of latent type *t* in E_c.

However in our case, an entity may have multiple types. Known as the multi-label learning problem, this poses difficulties for most learning and classification methods. The information gain - entropy based approach from the C4.5 decision tree algorithm is no exception [17]. To overcome this problem, we employ a modified version of the entropy proposed in [18] that considers multiple labels by introducing the probability of an entity not belonging to a certain type:

$$H(E_c) = -\sum_{i=1}^{n} ((p(t_i) \log p(t_i)) + (q(t_i) \log q(t_i))) \tag{11}$$

with *n* and *p(t)* as before and $q(t_i) = 1 - p(t_i)$ the probability of not having type t_i.

4 Evaluation

The approach we present in this paper has two major objectives: To provide transparency regarding the quality of the data accessed through the property-based paradigm and to improve the quality of the selected data by iteratively, and with user feedback, extending the property-based type definition with chosen properties. To evaluate how well these objectives have been fulfilled we performed the following experiment: Starting from different concepts presented in structured form with schemata on schema.org, as in the use case presented in Section 2, we build an initial type definition for each concept. This initial definition embodies typical properties most application developers require in order to program with each concept. It comprises the first four properties that have been most frequently annotated in ClueWeb12 for the corresponding schema.org schemata. The property-based data access is applied to these four properties and a set of entities from the BTC data corpus is selected. The quality score, precision and recall are computed for the selected entities. If the quality score is lower than 0.65 (our experiments have shown that a threshold of 0.65 brings satisfying data quality), a property is chosen based on its information gain. The user is asked whether this property belongs to the concept or not. We simulate the user feedback by relying on information from schema.org: If the property with the highest information gain is part of the schema that describes the corresponding concept on schema.org (considering synonymy), then the user feedback is positive. The type definition for the concept is in this case extended with this property and all entities having this property are kept. If, however, the property with the highest information gain is not part of the schema, then it is considered a negative property and all entities not having this property are kept. The process is repeated until the quality score reaches the quality threshold. Using schema.org to simulate user feedback is convenient but it has some drawbacks that will be addressed in future work: Some properties that are part of schema.org may be irrelevant from a human perspective. At the same time, schema.org doesn't claim full completeness. In consequence one can't be sure that properties not being part of schema.org are negative properties.

In order to measure precision and recall, a gold standard is required. The gold standard represents, in this case, clear type information w.r.t. the concepts: In the context of movies, is a given entity a movie or not? We build the gold standard by bootstrapping on a set of 1000 seed entities that we know are of the concept type: We extract all rdf:type types for each of the seed entities. On average, about 500 types are found. Types that are not related to the concept or that are too general (e.g. owl:Thing or schema.org/CreativeWork) are manually pruned. In a second iteration, all entities having those types are selected and 100 entities are randomly chosen. Only those entities that, on manual inspection show the correct type are kept. Their rdf:type types are extracted, and unrelated or general types are again manually pruned. The process is repeated one more time. The resulting list of rdf:type values represents the description of a concept type according to the rdf:type property. Any entity that has one of the types in the list is considered to be of the respective type. Of course, only a subset of the actual expressions of a certain type is found. As a result, the precision and recall values computed on this gold standard underestimate the actual values.

Our system chooses key properties to improve the type definition based on information gain. As a baseline, we built Rand, a system choosing properties at random (without replacement). The randomization process is repeated 10 times for each property selection step. Average quality, precision and recall values are considered for each iteration. The property that is closest to the average scores of all 10 random picks is chosen to extend the definition for the next iteration.

We evaluated ProSWIP on multiple concepts from various fields, with different characteristics. For brevity reasons, in Table 5 we present the results on the example of three chosen concepts. The base iteration (0) is common to both systems and corresponds to the most frequent four properties used for annotating the corresponding schema in ClueWeb. For Movie, this iteration already produces good precision but it is quite restrictive in terms of recall. ProSWIP requires in this case 4 iterations to reach quality above 0.65 and perfect precision. With 0.93, precision is already very good after the first iteration. Further iterations isolate well defined movies from the ones with missing values. This in turn affects recall. Benefitting from high quality entity selection from the base iteration (78% of entities selected from the start are movies), the random approach is also able to obtain good results. Primarily guided by average scores and with high quality semantic feedback, the baseline method achieves 0.95 precision and a quality score of 0.59 after 4 iterations. Recall however is severely affected by the random choice of properties. For Music the base iteration is, with a precision of 0.44, of lower quality. Various types of entities are selected. The probability for the random selector to choose some irrelevant property is higher in this case. This is also reflected in the poor performance of Rand for Music. In contrast, ProSWIP achieves the desired level of quality after only two iterations. For Books, the base also has low precision with negative consequences on the performance of Rand. The quality metric we introduced is highly correlated to precision on all experiments (Pearson's linear correlation coefficient of 0.94) denoting its expressiveness for the quality of the data selection. Precision rapidly increases towards values above 90%, showing the success of the whole approach.

Table 5. Quality, Precision and Recall for three chosen concepts and multiple iterations

Iteration	**Quality(Q)**		**Precision**		**Recall**	
Movies	ProSWIP	Rand	ProSWIP	Rand	ProSWIP	Rand
0	0.49	0.49	0.78	0.78	0.26	0.26
1	0.57	0.5	0.93	0.78	0.25	0.26
2	0.55	0.51	0.91	0.74	0.12	0.03
3	0.58	0.53	0.96	0.89	0.11	0.03
4	0.65	0.59	1	0.95	0.07	0
Music						
0	0.34	0.34	0.44	0.44	0.82	0.82
1	0.58	0.34	0.99	0.43	0.82	0.78
2	0.67	0.34	0.99	0.43	0.62	0.78
Books						
0	0.21	0.21	0.37	0.37	0.71	0.71
1	0.32	0.21	0.83	0.38	0.07	0.07
2	0.52	0.22	0.93	0.39	0.07	0.07
3	0.59	0.25	0.89	0.43	0.04	0.07
4	0.65	0.25	1	0.43	0.03	0.07

From a technical perspective, ProSWIP is a component implemented in Scala (www.scala-lang.org/), which maps variable names to properties from the BTC data set. While classical relational databases are not suitable for querying on RDF data, graph databases like Neo4j (www.neo4j.org/) have limited performance for our approach. In comparison, Lucene (lucene.apache.org/) has proven much faster in both the time needed for initially loading the data (building the index) as well as in terms of querying. With an off-the-shelf commodity computer with Intel I5-3550 quad-core CPU with 3.3 GHz. 32 GB RAM and 8.5 ms access hard drive, the index creation for the complete BTC data set took about 39 hours (only one core was used). The resulting index was about 1T in size including data. One simple entity search takes about 16 seconds. But the complete process of property-based data access may take up to hours as multiple queries, entity and property retrievals are being performed. It was possible to speed up the process by introducing caching mechanisms, for instance for the property synonymy dictionaries. Computing the quality, principal components, latent types and information gain for all properties on large data samples takes under 2 seconds. Nonetheless, we believe that in order to realize all operations in real-time a Lucene-based distributed index leveraging Hadoop is necessary.

5 Related Work

Property-based data access and its suitability for programming the Semantic Web has recently been discussed in [7, 19]. Challenges and open questions concerning a property based approach are discussed in these papers. Sharing their view, we inspect the practical feasibility of such an approach and address one of the main challenges: The data quality problem.

Structural typing approaches are already employed in programming: Property-based interfaces have been studied for OO languages [20] or extensible record systems for different language settings [21, 22]. But additional challenges like discovering, comprehending and extending property sets to match the intended use arise in the context of linked open data.

From a broader perspective, systems like Tipalo [23] performing automatic typing for DBpedia entities are also relevant to our approach. Tipalo extracts types for entities based on their corresponding Wikipedia pages. But there are several entities in the LOD cloud having no article on Wikipedia that would hence remain untyped (there are about 14,199 diseases (International Statistical Classification of Diseases: http://www.who.int/classifications/icd/en/) most of them documented through PubMed but only about 3,000 of them featuring an actual article on Wikipedia). High precision knowledge bases like YAGO [24] relying on the Wikipedia category system and Infoboxes suffer from the same problem. In contrast, we build on structural similarity independent of all-encompassing information sources to find latent, contextually relevant types.

6 Conclusions and Outlook

We believe that property-based data access represents a cornerstone in programming with data from the Web. Our experiments show that such an approach suffers from quality problems that the end user is not even aware of. With an entity homogeneity-based quality metric and iterative feedback from the user on chosen properties, the level of quality for the selected data can be controlled. Being highly correlated to precision, the quality measure we introduced provides for transparency. With additional feedback on chosen properties, precision easily reaches values above 0.9, confirming the success of this approach.

The sparse nature of data in the LOD cloud severely affects recall. Leveraging high quality property-based definitions, the recall problem can be tackled: We plan to use properties that have been found suitable to extend the concept definition, not as filters, but as features for entity ranking on structural similarity. This should increase the robustness against missing values and have a positive effect on recall.

References

1. Barry, D., Stanienda, T.: Solving the Java Object Storage Problem. Computer 31, 33–40 (1998)
2. Carroll, J.J., Reynolds, D., Dickinson, I., Seaborne, A., Dollin, C., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: Proc. WWW, New York, USA, pp. 74–83 (2004)
3. Völkel, M.: RDFReactor – From Ontologies to Programmatic Data Access. In: Proc. ISWC (2005)

4. Tratt, L.: Compile-time meta-programming in a dynamically typed OO language. In: Proc of the Dynamic Languages Symposium (DLS), San Diego, California, USA (2005)
5. Stock, W.G.: Concepts and Semantic Relations in Information Science. Journal of the American Society for Information Science and Technology 61, 1951–1969 (2010)
6. Scheglmann, S., Gröner, G.: Property-based Typing for RDF-Access. In: Proc. of Workshop on Programming the Semantic Web, Rio de Janeiro, Brasil, pp. 4–7 (2012)
7. Scheglmann, S., Groener, G., Staab, S., Lämmel, R.: Incompleteness-aware programming with RDF data. In: Proc. of Workshop on Data Driven Functional Programming (DDFP), vol. 11 (2013)
8. Cardelli, L.: Structural subtyping and the notion of power type. In: Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL 1988, pp. 70–79. ACM Press (1988)
9. Cafarella, M.J., Etzioni, O.: Navigating Extracted Data with Schema Discovery. In: Proc. of Int. Workshop on Web and Databases (WebDB), Beijing, China (2007)
10. Jaccard, P.: Nouvelles recherches sur la distribution orale. Bulletin Societe Vaudoise des Sciences Naturelles 44, 223–270 (1908)
11. Hancock, T., Jiang, T., Li, M., Tromp, J.: Lower Bounds on Learning Decision Lists and Trees. Information and Computation 126, 114–122 (1996)
12. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
13. Quinlan, J.R.: Simplifying decision trees. Int. Journal of ManMachine Studies 27, 221–234 (1987)
14. Gottron, T., Knauf, M., Scheglmann, S., Scherp, A.: A Systematic Investigation of Explicit and Implicit Schema Information on the Linked Open Data Cloud. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 228–242. Springer, Heidelberg (2013)
15. Jolliffe, I.: Principal Component Analysis, 2nd edn. Springer Series in Statistics (2002)
16. Ball, G., Hall, D.: ISODATA: A novel method of data analysis and pattern classification (1965)
17. Tsoumakas, G., Katakis, I.: Multi Label Classification: An Overview. Int. Journal of Data Warehousing and Mining 3, 1–13 (2007)
18. Clare, A., King, R.D.: Knowledge Discovery in Multi-label Phenotype Data. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS (LNAI), vol. 2168, pp. 42–53. Springer, Heidelberg (2001)
19. Scheglmann, S., Scherp, A., Staab, S.: Declarative Representation of Programming Access to Ontologies. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 659–673. Springer, Heidelberg (2012)
20. Gil, J.Y.: Whiteoak: Introducing Structural Typing into Java, pp. 73–89 (October 2008)
21. Bracha, G., Lindstrom, G.: Modularity meets inheritance. In: Proc. of Int. Conf. on Computer Languages, pp. 282–290. IEEE (1992)
22. Kiselyov, O., Lämmel, R., Schupke, K.: Strongly typed heterogeneous collections. In: Proc. of the SIGPLAN Workshop on Haskell, pp. 96–107 (2004)
23. Gangemi, A., Nuzzolese, A.G., Presutti, V., Draicchio, F., Musetti, A., Ciancarini, P.: Automatic Typing of DBpedia Entities. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 65–81. Springer, Heidelberg (2012)
24. Suchanek, F.M., Weikum, G.: YAGO : A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In: Proc. of WWW, Banff, Canada (2007)