An Approach to Website Schema.org Design

Albert Tort and Antoni Olivé

Department of Service and Information System Engineering
Universitat Politècnica de Catalunya – Barcelona Tech
{atort,olive}@essi.upc.edu

Abstract. Schema.org offers to web developers the opportunity to enrich a website's content with microdata and schema.org. For large websites, implementing microdata can take a lot of time. In general, it is necessary to perform two main activities, for which we lack methods and tools. The first consists in designing what we call the *website schema.org*, which is the fragment of schema.org that is relevant to the website. The second consists in adding the corresponding microdata tags to the web pages. In this paper, we describe an approach to the design of a website schema.org. The approach consists in using a human-computer task-oriented dialogue, whose purpose is to arrive at that design. We describe a dialogue generator that is domain-independent, but that can be adapted to specific domains. We propose a set of six evaluation criteria that we use to evaluate our approach, and that could be used in future approaches.

Keywords. Schema.org, Microdata, Ontologies, Conceptual Modeling

1. Introduction

Google, Bing and Yahoo's initiative to create schema.org for structured data markup has offered an opportunity and at the same time has posed a threat to many web developers. The opportunity is to transform the website's content to use HTML microdata and schema.org, so that search engines can understand the information in web pages and, as a consequence, they can improve the accuracy and the presentation of search results, which can translate to better click through rates and increased organic traffic [1,15]. Google, for example, uses schema.org markup to display rich snippets in the search results it produces, and in Custom Search[1], a service that enables the creation of search engines for a website or a collection of websites. The threat of not doing the website transformation is just the opposite: not reaping the above benefits that other websites may gain. This is the reason why many web developers are considering, or will consider in the near future, the schema.org markup of their web pages.

For large websites, implementing microdata can take a lot of time and require some big changes in the HTML source code [1]. In general, that implementation requires

[1] http://googlecustomsearch.blogspot.com

two main activities. The first consists in designing what we call the *website schema.org*, which is the fragment of schema.org that is relevant to the website. The second consists in adding the microdata tags to the web pages, using the previously designed website schema.org.

In this paper, we describe an approach to website schema.org design. Our approach consists in a human-computer task-oriented dialogue, whose purpose is to design a website schema.org. The dialogue uses the directive mode, in which the computer has complete control [22]. In each dialogue step, the computer asks a question to the web developer about the website content. Depending on the answer, a fragment of schema.org is or is not added to the website schema.org. The dialogue continues until the design is finished.

The methodology of our research is that of design science [2], which is defined in [18] as "the scientific study and creation of artifacts as they are developed and used by people with the goal of solving practical problems of general interest." The problem we try to solve is the design of a website schema.org. The problem is significant because it is (or will be) faced by many developers and, due to the novelty of the problem, they lack the knowledge and the tools required for solving it. In this paper we present an approach to the solution of that problem. As far as we know, this is the first work that explores the problem of website schema.org design.

According to [18], the main activities in design science research are: explicate problem, define requirements, design and develop artifact, demonstrate artifact and evaluate artifact. Many design science projects focus on one or two of the activities, while the others are treated more lightly. In our project we have focused in the first three activities: we formulate the problem to be solved, we outline a solution to that problem in the form of an artifact, and we create an artifact that addresses the problem[2].

The structure of the paper is as follows[3]. Next section describes schema.org and presents its metamodel. Section 3 defines the problem of website schema.org design and reviews the relevant previous work to its solution. Section 4 explains our approach to the solution of the problem. Section 5 presents the evaluation of the approach. Finally, section 6 summarizes the conclusions and points out future work.

2. Schema.org

In this section, we briefly introduce schema.org and present its UML [3] metamodel, which is shown in Fig. 1.

Schema.org is a large conceptual schema (or ontology) [4] comprising a set of types. A type may be an object type or a property[4]. Each type has a name and a description. An object type may be an entity type, a data type or an enumeration.

---

[2] A free, public version of the tool can be found at http://mpi.upc.edu/gmc-en/tools/schemaorg/introduction?set_language=en

[3] This paper is a revised and extended version of [25]

[4] At the time of writing, there are 428 object types and 581 properties, with a significant increase over time.

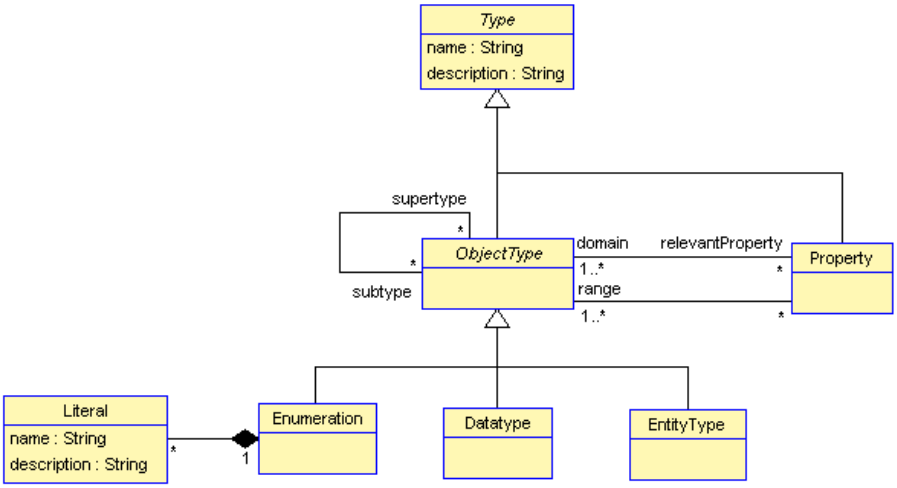

Fig. 1. The UML metamodel of Schema.org

Object types are identified by their *names*, and are arranged in a multiple specialization/generalization hierarchy where each object type may have several *supertypes*. For example, the entity type *LocalBusiness* is a *subtype* of both *Organization* and *Place*. The top of the hierarchy is the entity type named *Thing*. All other object types are a direct or indirect subtype of it.

An enumeration consists of a set of literals. Enumerations may have subtypes. For example, *MedicalSpecialty* is a subtype of both *Specialty* and *MedicalEnumeration*, which are in turn a subtype of *Enumeration*. The literals of an enumeration have different names.

Schema.org includes a predefined set of data types comprising *Text*, *Number*, *Boolean* and others. Data types may have subtypes too. For example *Integer* is a subtype of *Number*.

Properties are identified by their *name*. Properties are similar to UML attributes or binary associations, but with three important differences:

- The domain of a property may be one or more object types[5]. The property may be a relevant property of any of these types. For example, the domain of the property *height* may be a *MediaObject*, a *Person* or a *Product*. In UML, this property would normally correspond to three distinct attributes. The alternative of representing that property as a single UML attribute would require an artificial entity type that is a generalization of *MediaObject*, *Person* and *Product.*
- The range of a property may be one or more object types. The value(s) of the property should be instances of at least one of these types. For example, the range of the property *creator* may be an *Organization* or a *Person*. The UML equivalent of that property would be two binary associations

[5] The domain of almost all properties is an entity type. However, there are a few exceptions in which the domain is an enumeration.

for each possible type of its domain (two in this case: *CreativeWork* and *UserComents*).

- Schema.org neither defines nor takes into account the two cardinalities that could be defined for a property.

In principle, a property might also be a subtype of another one, although this is possible only in user extensions, and therefore it has not been considered in the metamodel of Fig. 1.

The main additional constraint of the metamodel is that a property cannot be redefined in a subtype. In OCL, this can be formalized by the following invariant:

```
context ObjectType
inv doesNotRedefineProperties:
  self.relevantProperty->
       forAll(p|not self.inheritedProperties()->includes(p))
```

where *inheritedProperties* is an operation defined in the same context as:

```
inheritedProperties():Set(Property) =
  self.allParents()->collect(p|p.relevantProperty)->asSet())
```

and

```
allParents():Set(ObjectType) =
  self.supertype->
              union(self.supertype->collect(p|p.allParents()))->asSet
```

Given that schema.org allows multiple specializations, in principle it is possible that an object type inherits the same property from two different paths. This happens, for example, in *LocalBusiness* whose supertypes are *Place* and *Organization*, and both have *review* as relevant property.

3. Website Schema.org Design

In this section, we formalize the concept of website schema.org (3.1), define the problem of designing that schema (3.2), and review the relevant previous work (3.3). Our approach to the solution of that problem will be presented in the next section.

3.1 Website Schema.org

In general, the web pages of a website include the representation of many facts, some of which are an instance of concepts (object types and properties) defined in schema.org while others are an instance of concepts that are not defined in schema.org. We call *website schema.org* of a website the set of concepts of schema.org that have (or may have) instances represented in its web pages.

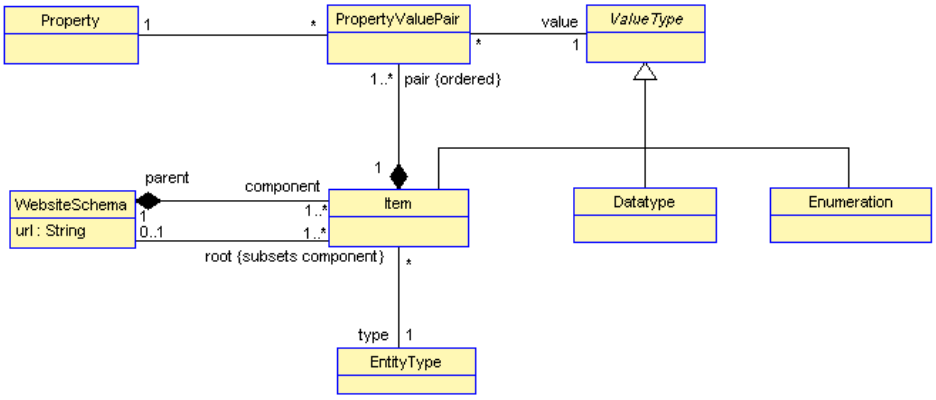

Fig. 2. The UML metamodel of a website schema.org

However, a website schema.org is not simply a subset of the schema.org concepts, because there are facts of a concept that are represented in a context of the website, but not represented in another one of the same website. For example, consider a website that represents instances of the entity type *Offer*, including values for the properties *seller* and *itemOffered*, among others. The value of *seller* is an *Organization*, for which the website shows only its *name*, *address* an *email*. On the other hand, the value of *itemOffered* is a *Product*, for which the website may show its *manufacturer*, which is also an *Organization*. However, for manufacturing organizations the website only shows their *name*, and not their *address* and *email*. The website schema.org of this example must indicate that the address and email of an organization are shown only for sellers.

Figure 2 shows the metamodel in UML of a website schema.org, which must be seen as an extension to the metamodel shown in Fig. 1. A website schema.org consists of a set of one or more *components*, which are instances of *Item*. We use here the term *item* with the same meaning as in the microdata model: a group of name-value pairs (that we call property-value pairs) [17]. An *Item* has a *type*, which is an *EntityType*.

A website schema.org has a set of one or more *roots*. The *type* of a *root* is an entity type that is the main subject of one or more pages of the website. For example, a root of a recipe website schema.org is an *Item* whose type is *Recipe*. Another root of the same website may be *Book* if there are web pages whose main subject is information about books. The roots of a website schema.org are a subset of its components.

An *Item* consists of an ordered set of at least one *PropertyValuePair*. Each instance of a *PropertyValuePair* has a *property* and a *value*. The *property* is an instance of *Property* and the value is an instance of the abstract class *ValueType*, which is a generalization of *Item*, *Datatype* and *Enumeration*.

We use a textual notation for defining a website schema.org (that is, an instance of the metamodel shown in Fig. 2). Figure 3 shows the example corresponding to the restaurant presented in “schema.org/Recipe”. There are two *Items*, with types *Recipe* (the *root*) and *NutritionInformation*. The first has twelve property-value pairs, one of

which (*nutrition*) has as value an *Item*, and the other eleven have as value a *Datatype* (*Text, Date, URL* or *Duration*). *NutritionInformation* has two property-value pairs, whose values are *Datatypes* (*Text*).

```
<Recipe, name,Text>
<Recipe, author, Text>
<Recipe, datePublished, Date>
<Recipe, image,URL>
<Recipe, description,Text>
<Recipe, prepTime,Duration>
<Recipe, cookTime,Duration>
<Recipe, recipeYield,Text>
<Recipe, nutrition, NutritionInformation>
   <NutritionInformation, calories,Text>
   <NutritionInformation, fatContent,Text>
< Recipe, ingredients,Text>
< Recipe, recipeInstructions,Text>
< Recipe, interactionCount, Text>
```

Fig. 3. A website schema.org example, using a textual notation

```
<div itemscope itemtype="http://schema.org/Recipe">
  <span itemprop="name">Mom's World Famous Banana Bread</span>
  By <span itemprop="author">John Smith</span>,
  <meta itemprop="datePublished" content="2009-05-08">May 8, 2009
  <img itemprop="image" src="bananabread.jpg"
       alt="Banana bread on a plate"  />
  <span itemprop="description">This classic banana bread recipe comes
  from my mom -- the walnuts add a nice texture and flavor to the banana
  bread.</span>
  Prep Time: <meta itemprop="prepTime" content="PT15M">15 minutes
  Cook time: <meta itemprop="cookTime" content="PT1H">1 hour
  Yield: <span itemprop="recipeYield">1 loaf</span>
  <div itemprop="nutrition"
    itemscope itemtype="http://schema.org/NutritionInformation">
    Nutrition facts:
    <span itemprop="calories">240 calories</span>,
    <span itemprop="fatContent">9 grams fat</span>
  </div>
  Ingredients:
  - <span itemprop="ingredients">3 or 4 ripe bananas, smashed</span>
  - <span itemprop="ingredients">1 egg</span>
  - <span itemprop="ingredients">3/4 cup of sugar</span>
  ...
  Instructions:
  <span itemprop="recipeInstructions">
  Preheat the oven to 350 degrees. Mix in the ingredients in a bowl. Add
  the flour last. Pour the mixture into a loaf pan and bake for one hour.
  </span>
  140 comments:
  <meta itemprop="interactionCount" content="UserComments:140" />
  From Janel, May 5 -- thank you, great recipe!
  ...
</div>
```

Fig. 4. Example of microdata markup using the website schema.org of Figure 3.

Once the website schema.org is known, the web developer can add the corresponding microdata to the webpages. Figure 4 shows an example (an excerpt from the example shown in schema.org/Recipe).

There are five important additional constraints in the metamodel of Fig. 2, which we formalize in the following. The first is that for a given item there cannot be two property-value pairs with the same *property* and *value*. Formally:

```
context Item
inv hasUniquePropertyValuePairs:
       self.pair->isUnique(Tuple{p:property,v:value})
```

The second is that the *property* of a property-value pair must be one of the direct or inherited relevant properties of the *type* of the item associated with that pair. Formally:

```
context Item
inv includesRelevantProperties:
       self.pair->forAll(self.type.allProperties()->includes(property))
```

where *allProperties* is an operation defined as:

```
context ObjectType
  allProperties():Set(Property) =
       relevantProperty->union(inheritedProperties())
```

The third is that the *value* of a property-value pair must be one of the *ObjectType* that are the *range* of the corresponding *property,* or a subtype of one of those *ObjectType*. For example, consider the property *citation* of a recipe, whose *range* may a *CreativeWork* (a supertype of *Recipe*) or a *Text*. For a particular recipe shown in a web page, a value of citation may be an instance of:

- *CreativeWork*
- Any subtype of *CreativeWork*
- *Text*
- *URL* (the subtype of *Text*)

Formally:

```
context PropertyValuePair
inv hasAValidValue:
let valueofPair:ObjectType =
if self.value.oclIsTypeOf(Datatype) then self.value.oclAsType(Datatype)
else
  if self.value.oclIsTypeOf(Enumeration) then
       self.value.oclAsType(Enumeration)
  else self.value.oclAsType(Item).type
  endif
endif
in
self.property.range -> exists(
       r|Set{valueofPair}->closure(supertype)->includes(r))
```

The fourth constraint states that a root item cannot be the value of a property-value pair. Formally,

```
context Item
inv isUnnestedIfRoot:
       websiteSchema->notEmpty implies self.propertyValuePair->isEmpty
```

The last constraint states that a non-root item must be the value of one and only one property-value pair. Formally,

```
context Item
inv isNestedIfNonRoot:
       websiteSchema->isEmpty implies self.propertyValuePair->size()= 1
       and not self.propertyValuePair.item->includes(self)
```

3.2 Problem definition

Once we have defined what we mean by website schema.org, we can now state the problem we try to solve in this paper: the design of the website schema.org of a given website. The problem can be formally defined as follows:

Given:
- A website *W* consisting of a set of web pages. The website *W* may be fully operational or under design.
- The current version *S* of schema.org

Design:
- The *website schema.org WS* of *W*.

A variant of the problem occurs when the input includes a database *D* that is the source of the data displayed in *W*. A subvariant occurs when the database is not fully operational yet, and only its schema *DS* is available. Usually, *DS* will be relational.

All web developers that want to markup the web pages with schema.org microdata are faced with this problem. Once *WS* is known, the developers can add the corresponding markup in the web pages. Tools that illustrate how to add microdata once *WS* is known start to appear in the market[6].

3.3 Related Work

As far as we know, there have not been reported attempts to solve the design of a website schema.org. However, there is some previous work that is relevant to our problem, and that we briefly review in the following.

The task of web information extraction (WIE) could be seen as similar to website schema.org design, and therefore the work done in WIE systems [5, 19] could be relevant to our problem. The input to a WIE system is a set of online documents that are semi-structured and usually generated by a server-side application program. The extraction target can be a relation tuple or a complex object with hierarchically organized data. In these systems users must program a wrapper to extract the data (as in W4F [6] or DEQA [7]) or to show (examples of) the data to be extracted (as in Thresher [8]). There are a few differences that make WIE systems inappropriate for website schema.org design. In our case, the target is a fragment of a schema, without

6 For example http://schema-creator.org/ or http://www.microdatagenerator.com/

the facts, and if the website is under design, the online documents are not available. On the other hand, it is unfeasible to build wrappers because web developers do not know what to extract.

The table interpretation problem is a specialization of WIE focused on extracting data from HTML tables [9, 19]. [10] describes one of the more recent systems, which is an example of the ontology-guided extraction approach. In this case, the ontology is the universal probabilistic taxonomy called Probase, which contains over 2.7 million concepts. The system uses that ontology to determine the concepts corresponding to the rows of a table, and to its columns, from the names of the table headers and the values of the columns. This approach cannot be used in our case because in general web pages display many facts in a non-table format, and on the other hand the web pages may not be available.

Another related problem is schema matching, which deals with finding semantic correspondences between elements of two schemas or ontologies [11, 12, 20, 21, 23]. Schema matching may be relevant to our problem when the source of the website is a database and we know its schema [13]. Assuming the database is relational, in our context the correspondences are between table attributes and schema.org properties. There exist a large spectrum of possible matchers (see [14] for a recent review) but in our context they would require the involvement of users who know both the database schema and schema.org.

4. Our Approach to Website Schema.org Design

In this section we describe our approach to the design of a *website schema.org*. We start with an overview of the approach (sect. 4.1) and then we continue with a detailed explanation of its main components (sect. 4.2-4.4). Throughout this section we use examples from the websites *allrecipes.com* and *food.com*, which deal with cooking recipes [15]. Users publish their recipes in those websites, including for each of them its name, a description, the ingredients, nutritional information, cooking time, preparation videos, and so on.

4.1 Overview

Our approach to the design of a website schema.org is based on a computer-controlled dialogue (see Fig. 5). The dialogue is automatically generated (see sect. 4.4) from schema.org, enriched with domain knowledge by domain experts (as indicated in sections 4.2 and 4.3). In most cases, the dialog asks simple yes/no questions in natural language to the web developer. Figure 6 shows a fragment of that dialogue in our example. The answer to a question requires the web developer to know only the contents of the website. Prior knowledge on schema.org is not needed. Note that in our approach the website could be under design and that we do not need to know the schema of the website source database (if it exists).

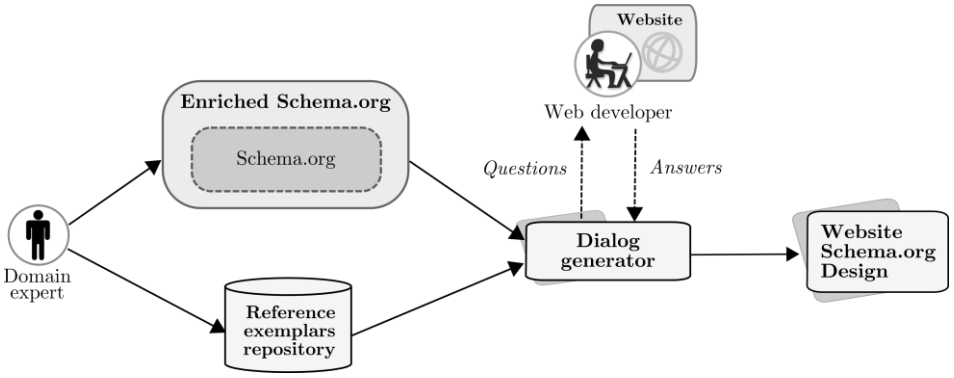

Fig. 5. A dialog approach to *website schema.org* design

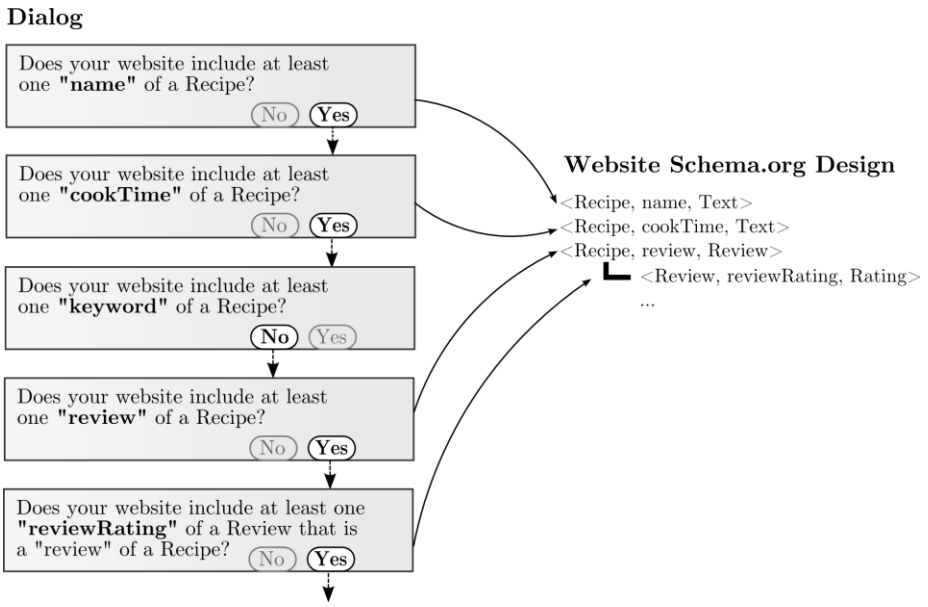

Fig. 6. Fragment of a dialogue in the allrecipes.com example

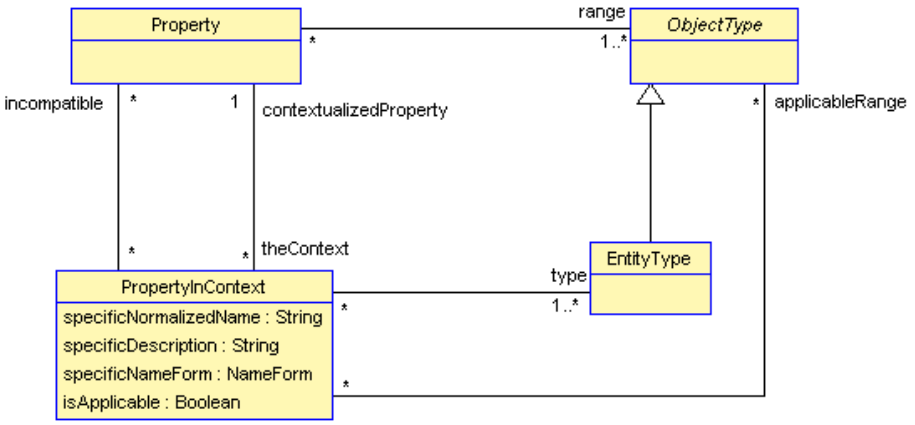

Fig. 7 Enrichment of the schema.org metamodel

4.2 Enriching Schema.org

The dialogue generator can generate dialogues from the content of schema.org. However, if domain experts provide additional knowledge then the generated dialogues can be some times more understandable (by improving the phrasing of the questions) and more selective (by asking only the most relevant questions). Figure 7 shows the enrichment of the metamodel of schema.org (Fig.1) that allows defining that additional knowledge.

The dialog generator deals with a property *P* always in a context. The context is an entity type that has *P* as a direct or indirect relevant property. In absence of additional knowledge, the dialog generator deals with *P* taking into account only the "official" names and descriptions of the involved types.

However, domain experts may add new knowledge by means of instances of *PropertyInContext* (*PIC*). An instance of that type has a few attributes and links that are useful when the dialog generator deals with a property in a particular context.

A *PIC* contextualizes a property (*contextualizedProperty*). The context in which a *PIC* is applicable is a set of one or more *EntityTypes* (*type*). For example, there may be a *PIC* for property *inLanguage* in the context of *CreativeWork*.

For any given pair of *contextualizedProperty* and *type* there must be at most one *PIC*. This is a constraint of the metamodel shown in Fig.7, which can be formally expressed by:

```
context PropertyInContext
inv hasUniquePropertyAndType:
       PropertyInContext.allInstances->
       isUnique(Tuple{p:contextualizedProperty,t:type})
```

The three first attributes of a *PIC* are the specific name form, normalized name and description of the contextualized property. The specific name form indicates the

grammatical form of the name, which may be a noun in singular form or a verb in third-person singular form. By default, it is assumed to be a noun. The specific normalized name and description may be used in the cases where the original name and description defined in schema.org can be improved in a given context. Such improvements (which are the equivalent of the "semantic label normalizations" performed in a different context [24]) allow the dialog generator to generate "better" questions.

For example, *CreativeWork* includes the property *inLanguage*. A *PIC* could specify a better name for this *contextualizedProperty*. The specific name form could be a *verb*, and the specific name could be "*is written in the language*". In this case, the *type* would be *CreativeWork*. As another example, *Recipe* includes the property *prepTime*. A *PIC* could specify that a more expressive name for that property in the context of *Recipe* is *preparation time*.

The last attribute of a *PIC* is *isApplicable*. The attribute may be used to indicate that a property is not applicable in a given context. For example, a domain expert can define that the property *genre* of *CreativeWork* is non-applicable for *Recipe*.

The *applicableRange* of a *PIC* may be used to restrict the set of ranges for the contextualized property. For example, the property *author* of *CreativeWork* has the range {*Organization, Person*}. If we want to specify that for *Recipes* the *author* must be a *Person*, then we create a link between the corresponding *PIC* and *Person* as its *applicableRange*. The applicable range must be a subset of the range of the corresponding property. This is formally expressed by:

```
context PropertyInContext
inv hasAValidRange:
       self.contextualizedProperty.range->includesAll(applicableRange)
```

Finally, there are properties that cannot be defined in a particular context if another one has previously been defined. For example, *author* is a property of *CreativeWork*, and *creator* is a property of *CreativeWork* and *UserComments*. However, in the context of *CreativeWork* only one of the two should be defined. We can then indicate in the corresponding *PIC*s that *author* and *creator* are *incompatible* with each other in the context of *CreativeWork* (*type*).

The definition of properties in context is a task that must be done by domain experts. However, a tool (like ours) may make it easy to define at any time a new enrichment, ensure that it satisfies all relevant constraints, and make it available for all future dialogue generations.

4.3 Reference exemplars

A basic approach to website schema.org design could be that the web developer first defines a root of the website (such as *Recipe*), then the dialogue generator automatically determines the schema.org properties that could be relevant, and finally the system asks the web developer which of those properties are relevant for the website.

However, that approach would not be practical, for two main reasons. The first is that there can be many schema.org properties for a given root, but not all of them are

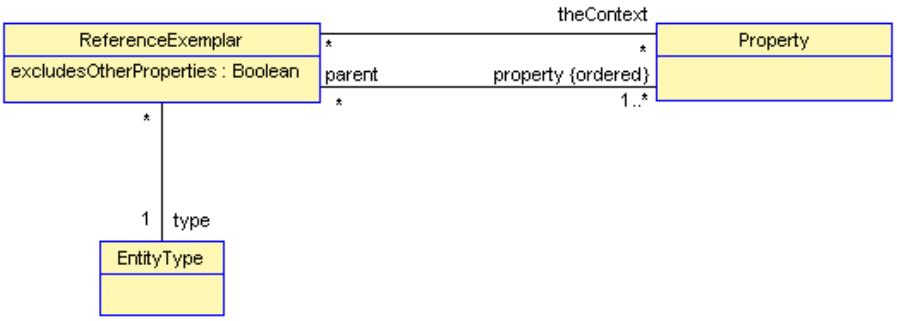

Fig. 8. Schema of reference exemplars

actually used in practice. For example, *Recipe* (a subtype of *CreativeWork*, which in turn is a subtype of *Thing*) has 81 properties (8 for *Thing*, 63 for *CreativeWork* and 10 for *Recipe*), but a representative website such as *allrecipes.com* only shows 14 of those properties (3 from *Thing*, 4 from *CreativeWork* and 7 from *Recipe*). Clearly, if the dialog generator were able to select a subset of properties that might be of interest for a given website, the system would ask much less questions to the web developer.

The second reason why the simple approach described above would not be practical is that the system would ask questions without any particular order, mixing questions belonging to different topics. For example, the system could ask about the presence of property *prepTime* (of *Recipe*), followed by *aggregateRaing* (of *CreativeWork*), *name* (of *Thing*) and then *cookTime* (again of *Recipe*). Clearly, such approach would confuse the web developer. Ideally, the questions posed by the system should be grouped by topic and unfold in a logical order, as required in, for example, questionnaire design [16].

Our solution to those problems is what we propose to call *reference exemplars*. Figure 8 shows their schema (an extension to the metamodel shown in Fig. 1). There are two kinds of reference exemplars: root and dependent. A root reference exemplar of a given *type* (which is an *EntityType*) is an ordered set of one or more properties that are shown in recommended websites of the given root. The order of the properties of the set is the order in which those properties are usually displayed in those websites. A root reference exemplar can be seen as a recommended practice for the schema.org markup of websites of a given root.

For example, if a domain expert recommends the properties displayed in *food.com* as a reference exemplar for *Recipe*, then the root reference exemplar of *Recipe* would comprise a set of 15 ordered properties. Other schema.org properties could be added to this set, if so desired. For example, given that a popular website such as *allrecipes.com* also displays the schema.org *video* and *review* properties, such properties could be added to the reference exemplar.

There must be a root reference exemplar for the type *Thing*, which is used when other more specific exemplars are not available. Moreover, for any given *EntityType* there may be at most one root reference exemplar. Formally, this is expressed by:

```
context EntityType
inv hasAtMostOneRootReferenceExemplar:
  self.referenceExemplar->select(r|r.theContext->isEmpty)->size()<= 1
```

Note that root reference exemplars are those that do not have context.

The properties of a reference exemplar must be a subset of those of its type. Formally:

```
context ReferenceExemplar
inv includesRelevantProperties:
       self.type.allProperties()->includesAll(property)
```

A dependent reference exemplar of a given type *E* and property *P* (*theContext*) is an ordered set of one or more properties that are usually shown in current websites of the given type *E* when it is the value of the property *P*. As before, the order of the properties of the set is the order in which those properties are usually displayed in recommended websites. A dependent reference exemplar can also be seen as a recommended practice. The same dependent reference exemplar can have several properties in its *context* meaning that it applies to any of them.

For any given *EntityType* (*type*) and *Property* (*theContext*) there may be at most one dependent reference exemplar. This is formally captured by the expression:

```
context Property
inv andATypeHaveAtMostOneDependentReferenceExemplar:
       self.referenceExemplar->isUnique(type)
```

For example, *food.com* includes the property *nutrition* of *Recipe*, whose value is the entity type *NutritionInformation*. For this type, nine properties are shown (*calories*, etc.). A domain expert that wishes to recommend these properties in that context could define a dependent reference exemplar, with *type NutritionInformation* and *theContext* = {*nutrition*}.

Reference exemplars have the boolean attribute *excludesOtherProperties*. We use it to indicate whether or not the dialog generator should consider other properties of the *type* beyond those indicated by the reference exemplar. For example, *Energy* has seven properties (all of *Thing*), but when used as a property of *calories*, only one of those properties are likely to be used (a text of the form <Number> <Energy unit of measure>). We could define a dependent reference exemplar for the type *Energy* and property *calories*, consisting of a single property (*name*) and excluding other properties. In this way, the dialogs can be highly simplified.

Reference exemplars are defined by domain experts. A tool (like ours) may make it easy to define at any time a new reference exemplar, ensure that it satisfies all relevant constraints, and make it available for all future dialogue generations. In the simplest case, a domain expert indicates a recommended website, from which the properties and their order can be automatically extracted using tools such as the Google Rich Snippet tool[7]. Another possibility is to just adopt the recommendations

[7] https://www.google.com/webmasters/tools/richsnippets

from search engines[8]. An even better possibility, not explored further here, is to integrate the properties shown in several recommended websites.

4.4 Dialog generation and execution

In the following, we describe the main steps of the process needed to design the schema of a website using our approach (see Fig. 5). The starting point is the creation of an instance *w* of *WebsiteDesign* (see Fig. 2), followed by the determination (by the web developer) of a root entity type *e* of *w*, and the invocation of the procedure *designSchema* indicated in Algorithm 1. The procedure is executed for each root entity type of *w*. As it can be seen, the procedure creates a root item *i* of *w* and then invokes (in line 5) the procedure *designSchemaForItem i*.

Algorithm 1. designSchema
input: An instance *w* of *WebsiteDesign*; an instance *e* of *EntityType.*
output: The complete design of website schema.org for *w*.
1. i := new Item;
2. i.websiteSchema :=w;
3. i.parent := w;
4. i.type := e;
5. designSchemaForItem(i,null,null);
6. if i.pair -> isEmpty() then destroy i; end;

Note that in line 6 of the above algorithm, the item is deleted if no property-value pairs have been found for it. This may happen when the website does not represent any fact about the schema.org properties of the root entity type *e*.

The procedure for the design of the schema for an item *i* is indicated in Algorithm 2. We first determine the nearest (root) reference exemplar *ref* for *i* (there is always one), and then we generate and execute two dialogs: the reference and the complementary dialogs. The first (lines 1-4) is based on the reference exemplar *ref* and considers only the properties of *ref*, and in their order. The second (lines 6-8) is performed only if *ref* does not exclude other properties and the web developer wants to consider all remaining properties. These properties are presented in the order of their position in the hierarchy of schema.org.

Algorithm 2. designSchemaForItem
input: An instance *i* of *Item,* its parent *Item parent_i* and its parent *Property parent_prop*
output: The complete design of the fragment corresponding to *i*.
1. ref:= determineReferenceExemplarForItem(i);
2. for each p in ref.property do
3. generatePairsForProperty(i,p,parent_i,parent_prop);
4. end;
5. if not ref.excludesOtherProperties and userWantsAllProperties then
6. for each p in (i.type.allProperties() - ref.property->asSet()) ->sortedBy(positionInHierarchy) do
7. generatePairsForProperty(i,p,parent_i,parent_prop);
8. end
9. end

[8] For example, in https://support.google.com/webmasters/topic/4598337?hl=en&ref_topic=3309300, Google suggests properties for 11 entity types. In particular, it suggests 14 schema.org properties for recipes.

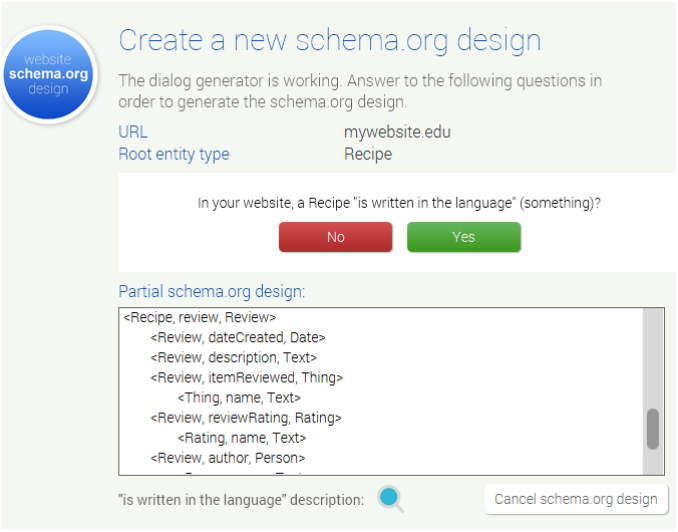

Fig. 9. Example of a question on the presence of property inLanguage of Recipe

The procedure *generatePairsForProperty* (algorithm 3) generates the property-values pairs of a property, if it is applicable and it is not incompatible with previously defined ones (see Fig. 7).

In line 2, the system asks the user whether or not the property *p* of item *i* is shown in the website, as illustrated in the examples of Fig. 6. The paraphrasing of the question uses the name and description indicated in the corresponding property in context, if it exists. A question may take one of the following forms (shown as examples):

- Does your website include at least one "description" of a Recipe? This form is used for properties (*description*) of root items (*Recipe*) whose name form is a noun.
- In your website, a Recipe "is written in the language" (something)? This form is used for properties (*is written in the language*) of root items (*Recipe*) whose name form is a verb.
- Does your website include at least one "name" of a Person that is an "author" of a Recipe? This form is used for properties in noun-form (*name*) of non-root items (*Person*) that are the value of a property (*author*) of a type (*Recipe*).
- In your website, a Review that is the "review" of a Recipe, "is written in the language" (something)? This form is used for properties in verb-form (*is written in the language*) of non-root items (*Review*) that are the value of a property (*review*) of a type (*Recipe*).

Figure 9 shows an example question in our tool. The system displays in the bottom the schema.org design that has been done already. The user may access to the complete description of the requested property by clicking the lens icon.

If the property *p* of item *i* is present in the website, the system determines its possible ranges, taking into account what is indicated in the corresponding *PropertyInContext* (Fig.7) or, if any, the definition of the property in schema.org (Fig. 1). If the range is not unique, then the operation asks the user the possible ranges of the property (one or more). If one of the possible ranges of *p* is an instance *E* of *EntityType*, then that operation asks whether the range of *p* is *E* or one of its subtypes. For example, the possible ranges of *author* are *Person* and *Organization*. If the user selects *Organization* as a possible range, then the system asks whether the range is *Organization* or one of its subtypes (there are no subtypes of *Person* in schema.org).

Algorithm 3. generatePairsForProperty
input: An instance *i* of *Item;* a property *p*; its parent *Item parent_i* and its parent *Property parent_prop*
output: The property value pairs of *p* for item *i*.

```
1. if isApplicable(i,p) and not incompatible(i,p)
2.     ranges := askQuestion(i,p,parent_i,parent_prop);
3.     for each r in ranges do
4.         pvp := new PropertyValuePair;
5.         pvp.property := p;
6.         pvp.item := i;
7.         if r is an EntityType then
8.             inew := new Item;
9.             pvp.value := inew;
10.            inew.parent := i.parent;
11.            inew.type := r;
12.            designSchemaForItem(inew,i,p);
13.            if inew.pair -> isEmpty() then destroy inew; end;
14.        else
15.            pvp.value = r;
16.        end;
17.    end
```

For each range, a property value pair is created (line 4), and if its value is an instance of *EntityType*, then the corresponding instance of *Item* is created (*inew*, line 8), and it is requested to generate its design by recursively invoking the operation *designSchemaForItem* in line 12. The execution of this operation now uses dependent reference exemplars. The process always ends because the depth of the compositions (Fig. 2) is finite in all practical websites.

5. Evaluation

As far as we know, ours is the first approach that has been proposed in the literature for solving the problem of website schema.org design, and therefore we cannot evaluate our proposal with respect to others. We propose in the following a set of six evaluation criteria that could be used to evaluate future new approaches to that problem, and we provide an evaluation of our approach with respect to those criteria. The criteria are: generality, precision, recall, human effort, cohesiveness and computation time.

Generality. Solutions may be general or domain specific. A general solution is applicable to any website, while a domain specific one is applicable to only one or more websites in domains such as, for example, online shops or recipes. The approach presented in this paper is general because it can deal with any root entity type defined in schema.org.

Precision and Recall. Precision and recall are two classical criteria used in information retrieval [26], which can be used here also. In our context, the information request is an entity type E for which we want to know its schema.org properties in a website W, and the answer is the set A of schema.org properties that have been generated in the design of a website schema.org WS for W. Let P be the set of all properties of E defined in schema.org, and P_W the subset of P that is relevant to W. For example, E = *Recipe*, P is the set comprising the 81 relevant properties to *Recipe* defined in schema.org, P_W is the subset of those properties that is relevant to W, and A is the set of properties that have been obtained in the design WS.

Precision is the fraction of the generated properties (the set A) which is relevant, and recall is the fraction of the relevant properties (the set P_W) which has been generated. Formally,

$$\text{Precision} = |P_W \cap A| \,/\, |A|$$
$$\text{Recall} = |P_W \cap A| \,/\, |P_W|$$

In our approach, if a complete dialog is performed (algorithm 2), and the web developer correctly identifies the relevant properties, then $P_W = A$ and therefore both precision and recall have the value one.

When only the reference dialog is performed (first dialog of algorithm 2), and the web developer correctly identifies the relevant properties, then $A \subseteq P_W$ and therefore the precision is still one. However, in this case the recall may be less than one if W displays properties not included in the reference exemplar. For example, assume that the dialog is based on the reference exemplar comprising the 16 properties from food.com and that W displays 13 of those properties and two more not included in the reference exemplar. In this case, we would have $|P_W| = 15$, $|A| = 13$, $|P_W \cap A| = 13$, and therefore Precision = 1 and Recall = 0.87. This is the reason why we would recommend to perform always the two dialogs of algorithm 2.

In the above evaluation, we have assumed that the web developer correctly identifies the relevant properties. That is, we assume that when the system asks whether or not a property is present in a website, its web developer provides the correct answer.

Human effort. This criterion evaluates the amount of human effort required by an approach. In general, two kinds of effort may be required: system administration and design. System administration effort may be needed to update the system whenever schema.org changes, or to provide any additional data required by the system. This effort is not necessary if the update is automatic and the system does not need additional data, or it is automatically captured. Design effort is the effort required by a web developer to design his website schema.org. This effort would not be necessary if an approach were completely automated, but it is difficult to see that such approach is possible and, if it were, it would not be applicable in the problem variant in which the website is under design.

In our approach, there is a significant system administration effort. Updating the system with the latest version of schema.org is not a problem, because changes are basically additions (new entity types and properties). The problem may be the effort required by domain experts to enrich the schema.org (Sect. 4.2) and to define reference exemplars (Sect. 4.3). Our approach may work without such enrichments and reference exemplars, but then the design effort is greater.

Thanks to the enrichment and to the reference exemplars, in our approach the design effort is small. Web developers have to answer one question for each potentially relevant property. Questions are simple, and the answer is easy in most cases.

Cohesiveness. Approaches that, like ours, are based on a human-computer dialog in a directive mode, face the problem of dialog cohesiveness. Intuitively, we define cohesiveness as the degree in which the questions posed by the system are grouped by topic and unfold in a logical order, as required in questionnaire design [16]. The lowest value would correspond to dialogs in which questions are randomly selected.

In our approach, we achieve maximum cohesiveness when the dialog is based only on reference exemplars, because then the order of the questions is the same as (or based on) the order used in recommended practices. However, if the web developer chooses a complementary dialog, then the overall cohesiveness may decrease, because the additional properties considered are presented in a top-down order, which should be better than random, but not necessarily the most logical.

Computation time. The computation time criterion evaluates the amount of time required by the computer. We conjecture that this time will normally be small and insignificant, because the number of schema properties relevant to a website is normally small, and the design must be performed only once. In our tool, the computation time has been less than one second per question.

In summary, we believe that in general our approach gets reasonable good results in the six proposed evaluation criteria. An exception may the criterion of system administration effort, although it remains to be seen if it is possible to get similar overall results with less system administration effort.

6. Conclusions

We have seen that the creation of schema.org for structured data markup has posed a problem to the (many) developers of websites that want to implement it in their web pages. We have formally defined that problem, which we call the problem of designing the *website schema.org* of a given website. We have identified two variants of the problem.

We have presented an approach to that design, consisting in a human-computer dialogue. The dialogue is automatically generated from schema.org, possibly enriched with domain knowledge. In the dialogue, the system asks simple questions in natural language to the web developer. The answer to a question requires the web developer

to know only the contents of the website. Prior knowledge on schema.org is not needed. In our approach the website could be under design, and we do not need to know the schema of the website source database (if it exists). We have implemented our approach in a prototype tool that is publicly available.

We have proposed a set of six criteria for the evaluation of possible solutions to the design of website schema.org, and we have evaluated our approach with respect to those criteria. Due to the novelty of the problem, there are not comparable alternative solutions yet. We believe that our approach will be useful to web developers because –among other things- it is easy to use, and it provides a systematic method to discover all schema.org microdata that could be added to the web pages.

The work reported here can be extended in many directions. First, the approach should be tested in the development of industrial websites in order to experimentally confirm its usefulness in practice. The experiment should be performed using our tool (or a professional version of it), fully loaded with relevant domain knowledge (properties in context and reference exemplars). Second, the approach could be extended to automatically generate examples of microdata markup from the design. Those examples could be useful to the web developers. An even better solution would be to provide an effective support to the web developer in adding the microdata tags to the web pages, using the previously designed website schema.org. Third, in the variant of the design problem in which the website is operational, it could be interesting to analyze the existing web pages in order to guess the presence of potential schema.org properties, which could then be suggested to the web developer. Finally, it would be interesting to develop a (semi-) automatic way of obtaining reference exemplars by integrating several recommended websites.

Acknowledgments. We thank the anonymous reviewers for their valuable comments to the previous version of the paper. We would also like to thank to Martin Menes for the work he has done in the development of the new version of the tool. This work has been partly supported by the Ministerio de Economía y Competitividad and FEDER under project TIN2008-00444, Grupo Consolidado.

References

1. Seochat: Schema.org and microdata markups for SEO (May 2013) http://www.seochat.com/c/a/search-engine-optimization-help/schema-org-and-microdata-markups-for-seo/.
2. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly (2004) 75-105
3. OMG. UML Superstructure v.2.4.1 (2011) http://www.omg.org/spec/UML.
4. Olive, A.: Conceptual Modeling of Information Systems. Springer, Berlin (2007)
5. Chang, C.H., Kayed, M., Girgis, R., Shaalan, K.F.: A survey of web information extraction systems. IEEE Transactions on Knowledge and Data Engineering 18(10) (2006) 1411-1428
6. Sahuguet, A., Azavant, F.: Building intelligent web applications using lightweight wrappers. Data & Knowledge Engineering 36(3) (2001) 283-316.
7. Lehmann, J., Furche, T., Grasso, G., Ngomo, A.C.N., Schallhart, C., Sellers, A.,Unger, C., Buhmann, L., Gerber, D., Honer, K.: DEQA: Deep web extraction for question answering. In: ISWC 2012, Springer (2012) 131-147.

8. Hogue, A., Karger, D.: Thresher: Automating the unwrapping of semantic content from the World Wide Web. In: WWW2005, ACM (2005) 86-95.
9. Embley, D.W., Hurst, M., Lopresti, D., Nagy, G.: Table-processing paradigms: A research survey. IJDAR Journal 8(2-3) (2006) 66-86.
10. Wang, J., Wang, H., Wang, Z., Zhu, K.Q.: Understanding tables on the web. In: ER 2012, Springer (2012) 141-155.
11. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4) (2001) 334-350.
12. Bellahsene, Z.: Schema Matching and Mapping. Springer (2011)
13. An, Y., Borgida, A., Mylopoulos, J.: Discovering the semantics of relational tables through mappings. Journal on Data Semantics VII (2006) 1-32.
14. Shvaiko, P., Euzenat, J.: Ontology matching: State of the art and future challenges. IEEE Transactions on Knowledge and Data Engineering 25(1) (2013) 158-176.
15. Krutil, J., Kudelka, M., Snasel, V.: Web page classification based on schema.org collection. In: CASoN 2012, IEEE (2012) 356-360
16. Pew Research Center: Question Order. http://www.people-press.org/methodology/questionnaire-design/question-order/
17. W3C: HTML microdata. http://www.w3.org/TR/microdata/
18. Johannesson, P., Perjons E.: An Introduction to Design Science. Springer 2014.
19. Embley, D.W, Liddle, S.W., Londsdale, D.W.: Conceptual Modeling Foundations for a Web of Knowledge. In Handbook of Conceptual Modeling, Springer 2011, pp. 477-516.
20. Bergamaschi, S., Beneventano, D., Guerra, F. Orsini, M.: Data Integration. In: Handbook of Conceptual Modeling, Springer 2011, pp. 441-476.
21. Shvaiko, P., Euzenat, J.: Ten Challenges for Ontology Matching. OTM Conferences (2) 2008: 1164-1182.
22. Smith, R.W., Hipp, D.R., Biermann, A.W.: A dialog control algorithm and its performance. In Proceedings of the third conference on Applied natural language processing (ANLC '92). Association for Computational Linguistics, Stroudsburg, PA, USA, 9-16., 1992 DOI=10.3115/974499.974502
23. Arnold, P., Rahm, E.: Enriching ontology mappings with semantic relations. Data Knowl. Eng. 93: 1-18 (2014)
24. Sorrentino, S., Bergamaschi, S., Gawinecki, M., Po, L.: Schema label normalization for improving schema matching. Data Knowl. Eng. 69(12): 1254-1273 (2010).
25. Tort, A., Olivé, A.: A Computer-Guided Approach to Website Schema.org Design. ER 2014: 28-42.
26. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval - the concepts and technology behind search. Second edition. Pearson Education Ltd., Harlow, England 2011.