A Computer-Guided 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]. The threat of not doing that 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 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. 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 overall framework of our research is that of design science [2]. 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.

The structure of the paper is as follows. 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 for its solution. Section 4 explains our approach to the solution of the problem. The approach has been implemented in a tool that has been very useful for testing and experimentation[1]. 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 review schema.org and introduce its UML [3] metamodel, which is shown in Fig. 1. As far as we know, this metamodel has not been published before (in any formal language).

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[2]. Each type has a name and a description. An object type may be an entity type, a data type or an enumeration. An enumeration consists of a set of literals.

A property may have one or more entity types as its domain and one or more object types as its range. For example, the property *creator* has as its domain *CreativeWork* and *UserComments*, and its range may be an *Organization* or a *Person*.

Types are arranged in a multiple specialization/generalization hierarchy where each type may be a *subtype* of multiple *supertypes*. For example, the entity type *LocalBusiness* is a subtype of both *Organization* and *Place*. The top of the hierarchy is the entity type *Thing*. All other object types are a direct or indirect subtype of it. A property may also be a subtype of another one, although this is used only in user extensions to schema.org. Enumerations may have subtypes. For example, *MedicalSpecialty* is a subtype of *Specialty*.

[1] A public preliminary version of the tool can be found at http://genweb.upc.edu/mpi/gmc-grup/eines/schemaorg/introduction

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

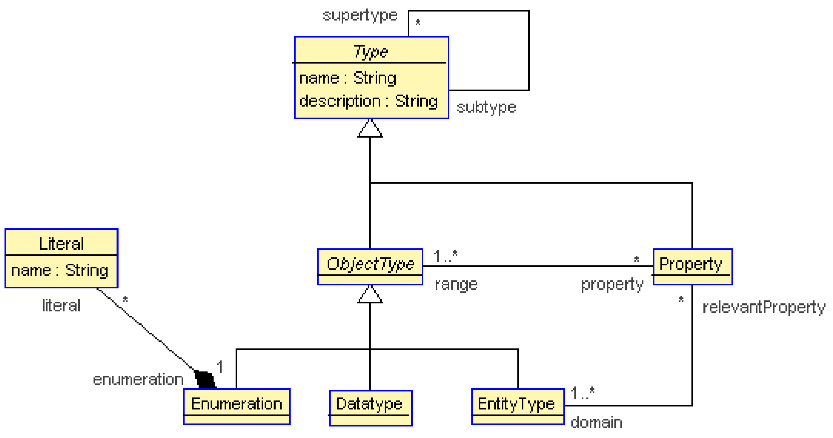


Fig. 1. The UML metamodel of Schema.org

3. Website Schema.org Design

In this section, we formalize the concept of website schema.org (3.1), we define the problem of designing that schema (3.2), and we 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 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.

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. As far as we know, this metamodel has not been published before (in any language). A website schema.org has one or more roots, which are instances of *Item*. We use here the term

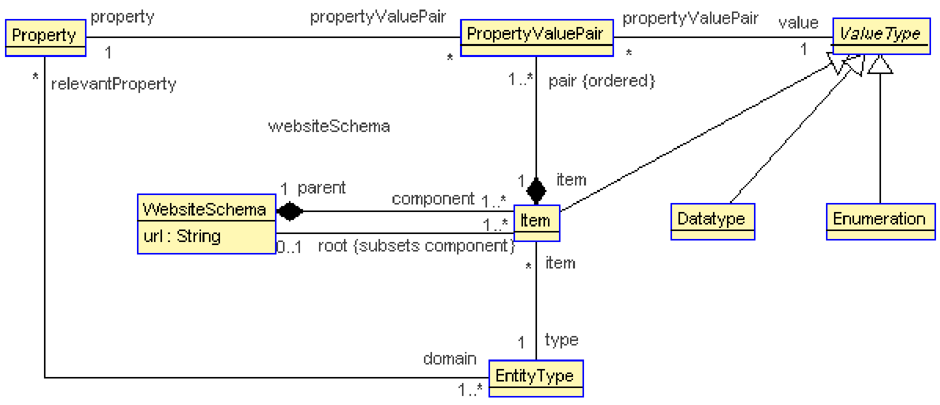


Fig. 2. The UML metamodel of a *website schema.org*

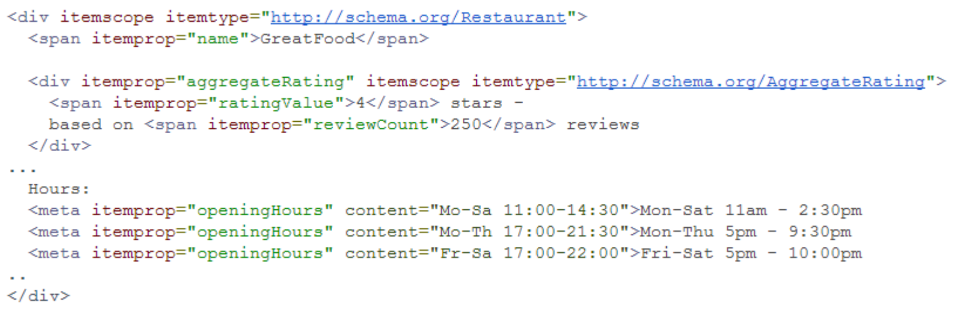
item with the same meaning as in the microdata model: a group of name-value pairs (that we call property-value pairs). An *Item* has a type, which is an *EntityType*. For example, the root of a restaurant website schema.org is an *Item* whose type is the *EntityType Restaurant*. An *Item* consists of an ordered set of at least one *PropertyValuePairs*.

Each instance of a *PropertyValuePair* has a *property* and a *value*. The property is an instance of *Property* and it must be one of the direct or indirect relevant properties of the type of the item. The value is an instance of the abstract class *ValueType*, which is an *Item*, a *Datatype* or an *Enumeration*. If the value is an *Item* then its *type* must be an *EntityType* that is in the range of the property, or a subtype of one of the ranges of the property.

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/Restaurant". There are three *Items*, with types *Restaurant* (the root), *AggregateRating* and *PostalAddress*. The first, has eight property-value pairs, two of which have as value an *Item*, and the other six have as value a *Datatype* (*Text* or *Duration*). *AggregateRating* has two property-value pairs, whose values are *Datatypes* (*Text* and *Number*). *PostalAddress* has three property-value pairs, whose values are also *Datatypes* (*Text*).

```
<Restaurant,name,Text>
<Restaurant, aggregateRating, AggregateRating>
    <AggregateRating, ratingValue,Text>
    <AggregateRating, reviewCount,Number>
<Restaurant, address, PostalAddress>
    <PostalAddress, streetAddress,Text>
    <PostalAddress, addressLocality,Text>
    <PostalAddress, addressRegion,Text>
<Restaurant, telephone,Text>
<Restaurant, url,Text>
<Restaurant, openingHours, Duration>
<Restaurant, servesCuisine,Text>
<Restaurant, priceRange,Text>
```

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



```
<div itemscope itemtype="http://schema.org/Restaurant">
  <span itemprop="name">GreatFood</span>

  <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
    <span itemprop="ratingValue">4</span> stars -
    based on <span itemprop="reviewCount">250</span> reviews
  </div>
...
  Hours:
  <meta itemprop="openingHours" content="Mo-Sa 11:00-14:30">Mon-Sat 11am - 2:30pm
  <meta itemprop="openingHours" content="Mo-Th 17:00-21:30">Mon-Thu 5pm - 9:30pm
  <meta itemprop="openingHours" content="Fr-Sa 17:00-22:00">Fri-Sat 5pm - 10:00pm
..
</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/Restaurant).

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[3].

3.3 Related Work

The task of web information extraction (WIE) could be seen as similar to website schema.org design, and therefore the work done on WIE systems [5] could be relevant to our problem. However, there are a few differences that make WIE systems inappropriate for website schema.org design. The input to a WIE system is a set of online documents that are semi-structured and usually generated by a server-side

[3] For example http://schema-creator.org/ or http://www.microdatagenerator.com/

application program. The extraction target can be a relation tuple or a complex object with hierarchically organized data. In our case, the target is a fragment of a schema, without the facts, and if the website is under design, the online documents are not available. On the other hand, 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]). In our case, this is unfeasible 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]. [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]. 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

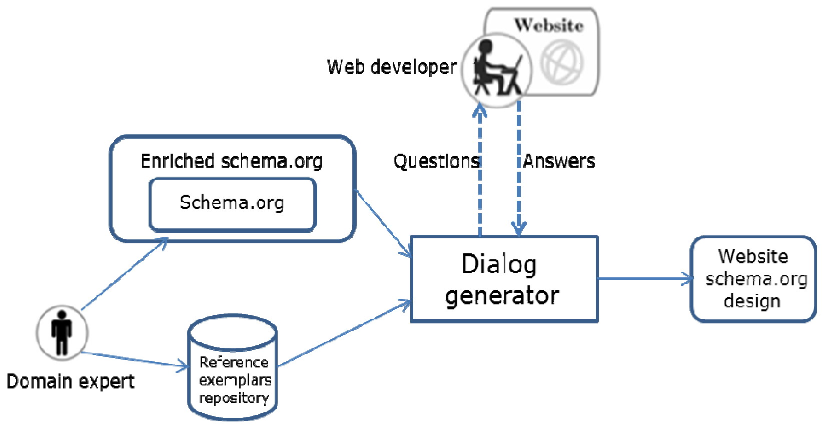


Fig. 5. Our approach to *website schema.org* design

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).

4.2 Enriching Schema.org

The dialogue generator can generate dialogues from the content of schema.org. However, if domain experts can provide additional knowledge then the generated dialogues can be 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 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*). If there is only one type and it is *Thing*, then the context is any entity type. For any given pair of *contextualizedProperty* and *type* there must be at most one *PIC*.

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 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 *verb*, and the

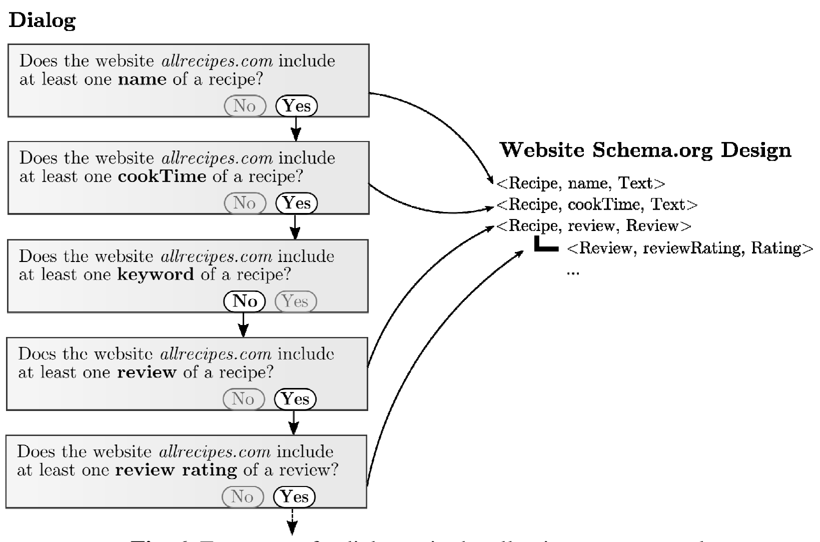

Dialog
Does the website allrecipes.com include at least one name of a recipe?
No
Yes
Does the website allrecipes.com include at least one cookTime of a recipe?
No
Yes
Does the website allrecipes.com include at least one keyword of a recipe?
No
Yes
Does the website allrecipes.com include at least one review of a recipe?
No
Yes
Does the website allrecipes.com include at least one review rating of a review?
No
Yes
Website Schema.org Design
<Recipe, name, Text>
<Recipe, cookTime, Text>
<Recipe, review, Review>
<Review, reviewRating, Rating>
...

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

specific name could be "*isWrittenInTheLanguage*". In this case, the *type* would be *Thing*.

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 as 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*.

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*).

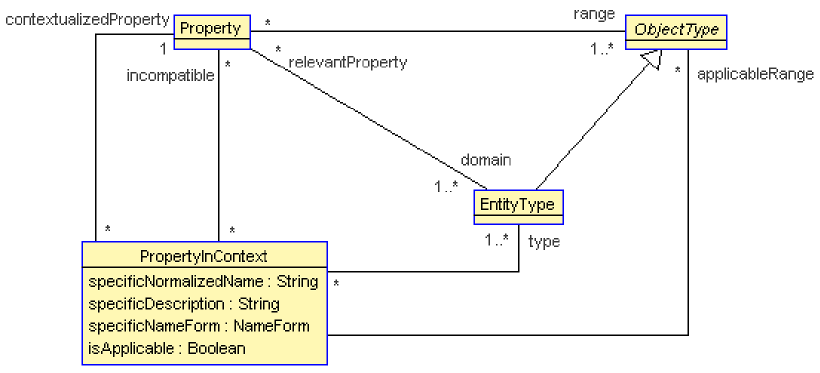


Fig. 7 Enrichment of the schema.org metamodel

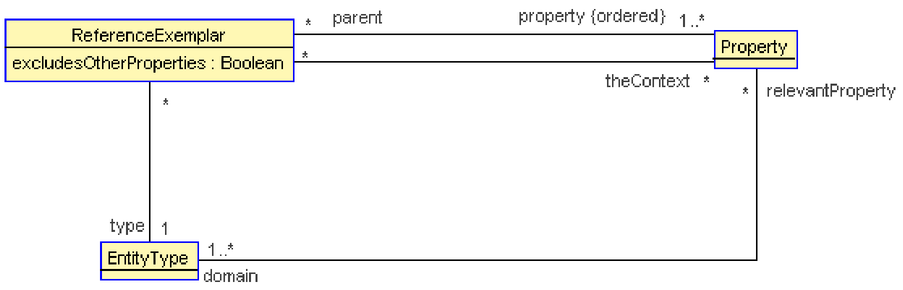


Fig. 8. Schema of reference exemplars

4.3 Reference exemplars

A basic approach to website schema.org design could be that the web developer first defines the 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 actually used in practice. For example, *Recipe* (a subtype of *CreativeWork*, which in turn is a subtype of *Thing*) has 67 properties (7 for *Thing*, 50 for *CreativeWork* and 10 for *Recipe*), but a representative website such as *allrecipes.com* only shows 13 of those properties. Clearly, if the dialog generator were able to select the 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. 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.

There must be a reference exemplar for the type *Thing*, which is used when other more specific exemplars are not available.

Root reference exemplars are defined by domain experts. 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[4]. Another possibility is to just adopt the recommendations from search engines[5]. An even better possibility, not explored further here, is to integrate the properties shown in several recommended websites. For example, if a domain expert recommends *food.com* as a reference exemplar for *Recipe*, the root reference exemplar of *Recipe* would comprise 16 properties in a given order. The properties shown by *allrecipes.com* are 12 of those, and only one new (*video*).

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 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.). The website *allrecipes.com* uses all of them, and adds a new one.

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.

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. As 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.root :=w;
3. i.parent := w;
4. i.type := e;
5. designSchemaForItem(i);
6. if i.pair -> isEmpty() then destroy i; end;
```

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

[5] For example. Google suggests the properties of *Recipe* indicated in https://support.google.com/webmasters/answer/173379?hl=en&ref_topic=1088474

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 (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.
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);
4. end;
5. if not ref.excludesOtherProperties and userWantsAllProperties then
6.     for each p in
               (i.type.hasProperty() - ref.property->asSet()) ->sortedBy(positionInHierarchy) do
7.             generatePairsForProperty(i,p);
8.     end
9.  end
```

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. If the property 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
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);
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);
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.

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 domains such as, for example, ecommerce or tourism. The approach presented in this paper is general.

Precision and recall are two classical criteria used in information retrieval contexts, which can be used here also. Now, instead of documents, we deal with schema.org properties that are relevant to a website and the properties that have been found. Therefore, in our case precision is the number of properties relevant to the website that have been found (true positives) divided by the total number of properties that have been found. Similarly, recall is the number of true positives, divided by the total number of relevant properties. Ideally, both precision and recall should have the value one.

In our approach, assuming that the web developer correctly identifies the root entity types (such as *Recipe*) of a website, the value of precision is always one, because the approach only considers those properties that are relevant and, therefore, all properties found are necessarily relevant. The value of recall is also one if the web developer chooses the complementary dialog and he correctly identifies the properties proposed by the system that are relevant to the website. The value of recall is less than one only when the web developer indicates that one or more proposed properties are not relevant to the website when, in fact, they are.

The human effort criterion evaluates the amount of effort the use of the approach requires to the web developers. That effort would be null 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 when the website is under design. In our approach, web developers have to answer one question for each potentially relevant property. Questions are simple, and their answer should be easy in most cases.

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.

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 our approach gets reasonable good results in the six proposed evaluation criteria.

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 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 by domain experts. 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). For the purposes of testing and experimentation, we have implemented our approach in a prototype tool.

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 several directions. First, the approach should be tested in the development of several 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. Third, when 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.

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/