OntoPair: Towards a Collaborative Game for Building OWL-Based Ontologies

Peyman Nasirifard, Slawomir Grzonkowski and Vassilios Peristeras

Digital Enterprise Research Institute
National University of Ireland, Galway
IDA Business Park, Lower Dangan, Galway, Ireland
firstname.lastname@deri.org

Abstract. Collective Intelligence takes advantage of collaboration, competition and integration. It often uses mixed groups of humans and computers to research in new unexplored ways. Ontologies, which are the main building block of the Semantic Web, are usually prepared by domain experts. We introduce a novel approach, which employs Collective Intelligence, towards building simple domain ontologies through a game called *OntoPair*, an entertaining web-based game that is able to build simple OWL-based ontologies based on collected information from players. The game collects properties and common-sense facts regarding an object by means of some fixed templates and translates them into OWL representation by aid of a mediator/mapper and builds simple domain ontologies after refinement in several iterations. We define the game and preform a small experiment that proves our idea.

1 Introduction

Ontologies are the main building blocks of the Semantic Web technologies. They try to define a specific domain in a systematic way. The can be expressed using different standards and languages like RDFS [3] and OWL [10]. One of the main concerns of Semantic Web researchers is building domain ontologies and collect sufficient instances for them. Because building domain ontologies is not an entertaining task, they are usually build by domain experts

In computer science, human-based computation is a technique in which a computational process performs its function via outsourcing certain steps to humans [16]. In other words, there are some tasks that most humans can do easily, but current computers can not perform them in a logical time (e.g. CAPTCHA [13]).

Collective Intelligence is a form of intelligence that emerges from the collaboration and competition of many individuals [17]. This phenomenon has been observed by many researchers for years. Among them, Pierre Levy [6] described its potential for the internet technologies. He pointed out that rapid and open data exchange would coordinate the intelligence in new unexpected ways.

In this paper we present a game called OntoPair which aims at harnessing the benefits of the Collective Intelligence phenomenon to create ontologies. We show

how to create them by a number of human-human competitions. We describe how computers should proceed and integrate the obtained results in a way that leads us to obtain well-defined ontologies.

2 OntoPair Structure

OntoPair is a two- or one-player game which provides an interactive environment between anonymous players to play and build simple ontologies. The game is based on traditional word guessing games [15] with some fixed templates which have been carefully chosen to be translated into OWL by means of a mediator/mapper. The game is composed of two different phases which are separated from each other, but the result of first phase is the input of next phase. Roughly speaking, these two main phases can be called *Collecting properties* and *Collecting common sense facts* about an object by means of some fixed templates. The game is mainly for two players, but it can be also played in single mode. The players do not know each other, they can not communicate, and they are randomly paired. The game can be played in two main modes: *graphic-based* and *text-based*. In the graphic mode, the players look at a same image and play, whereas in the text mode, the players look at a same text-based word or keyword and they play; e.g. in the graphic mode, the players may look at an image of a *car*, a *house* or a *bicycle*, but in text mode, they will see the explicit words of a *car*, a *house*, or a *bicycle*. In next sections, we describe each phase in a more detailed manner.

2.1 Collecting Properties

In first phase of *OntoPair*, collecting properties, the main goal is collecting properties, components, and characteristics of a specified object. This phase is very similar to ESP Game [7] and Google Image Labeler [5], but there exist several cruicial differences. The main difference is that in ESP game or Google Image Labeler, the players try to annotate an image and catch the objects that are located in images, whereas in this phase of OntoPair, players play to catch the properties and characteristics of a specific object in text-based or graphic-based mode. The other main difference is that ESP game and Google Image Labeler work only in graphic mode and text-based ESP game does not make sense; whereas in OntoPair, as we mentioned earlier, the game can be played in both graphic and text mode.

The graphic mode of this phase should be based on ESP game or Google Image Labeler, as we need the explicit name of objects that are located in the image. In other words, the result of ESP game or Google Image Labeler can be used in this phase of graphic mode of OntoPair. In graphic mode of OntoPair, the players will look at the same image and they have a *hint* which is actually the name of one of the objects that is located in image and the players should mention properties of that object and agree upon a property. In the text-based

mode, again both players will look at the same word which is fetched from a database of objects.

The result of this phase is actually a collection of properties of different objects. We store these properties in a data store and link them to the object. To clarify what we are looking for, we give some hints to the players. These hints are two general questions: *What does an object X contain/have?*, and *Which parts/components/characteristics does object X have?*. in these templates, *X* is replaced with the name of the object; e.g. *What does a car contain?* and *Which parts/components/characteristics does a car have?*

One of key concepts in games is *points*. Games without points do not make sense and players will lose their motivation to play after a while. In this phase of the game, we also give points to the players. After agreement of the players upon a property, both players get points and the game continues and shows another image or text, depending on game's mode. The game continues until one player quits or time is up, as the game is played in time intervals. If one player quits during the game, we try to find another player randomly. If it takes a long time, as the number of players is not always even, the game can go through single player version. The single player version is actually playing with a log file from previous games with the same object. Actually we store all properties of an object during each game. This will also help to evaluate the data source of the properties. It is obvious that at startup of the game with an empty knowledge base, the game can not be played in single player version and there should be always an even number of players. To avoid the game being boring, the players can skip current image/text and continue to see next random image/text, if the players find an object boring and they can not agree upon a property.

It is obvious that there exist some properties in an object that most players often mention, e.g. most players will say that a book has title or author, but probably a few of them will say about ISBN, price, or publishing date. To fight with these issues, we detect these often-used words and we prevent users to mention such words again and again by indicating them as *prohibited words*. The prohibited words which are assigned to the objects are calculated based on the number of each property which has been mentioned in previous plays. There is no doubt that a single object is played as long as the players are not able to detect a new property in it and they always skip the object. The prohibited words make the game more difficult, but more fun. The other advantage of the prohibited words is that these words can be used as hints for players to guess what information we are looking for as a property, part, characteristic, or component.

As a concrete example, suppose that both players are looking at an image of a *book*. They should play and mention the properties and different parts of a book, one after the other. First player says *title*, the second player says *author*; game continues: the first player says *chapter*, the second player says *ISBN*; game continues: the first player says *publisher*, the second player says *title*. At this point, both players agreed upon *title*, so both players get points and the game continues by showing another random image or word. It is obvious that

after several times of playing and putting common words to prohibited list, we catch, say, a complete collection of book's properties like author, chapter, ISBN, publisher, etc.

2.2 Collecting Triples and Common Sense Facts

The second phase of OntoPair, collecting triples and common sense facts, is also totally separated from the previous phase. Note that this phase is played by different players which are not necessarily the same as players in the first phase. The result of the first stage is used in this phase. Like the previous phase, there are also two players in this phase, who do not know each other and they can not communicate. The players are randomly paired. In this phase, we collect some pieces of information which are called common sense facts about an object by means of some fixed templates. Informally, a common-sense fact is a true statement about the world that is known to most humans [8]: "a book has one title", "a human has two legs", etc. As we mentioned earlier, collecting these common sense facts is done through fixed templates and based partially on properties that we have collected in the previous phase. This phase of OntoPair is a word guessing game that one player (narrator) should guide the other player (guesser) to guess a word which is actually the object that we are trying to find some common sense facts about it. In other words, in this stage, an image or a word is assigned to one player and he/she should complete pre-defined templates to guide the next player to guess the word. As soon as one template is completed, it will be sent to next player and as soon as the next player could come up with the right word, both players get points, the role of players will switch and game continues by showing another randomly-chosen image or word. These pre-defined templates have been chosen for a purpose: To translate them simply into OWL using a mediator/mapper. The explicit templates that we present to the players are listed below:

- It has at least ___ Y: The Y will be replaced by a list of real properties of the item that comes actually from the knowledge base of properties that we have collected in previous phase and the player can choose arbitrary property from a combo box. Here we catch the minimum cardinality of the property, if and only if it makes sense.
- It has at most ___ Y: The Y will be replaced by a list of real properties of the item that comes actually from the knowledge base of properties that we have collected in previous phase and the player can choose arbitrary property from a combo box. Here we catch the maximum cardinality of the property, if and only if it makes sense.
- It is kind of ___: With this template, we catch hierarchical information of the item.
- It could be either ___ or ___ (or more): From one perspective (see also next template), this template provides different types of the item. The player can extend this template by adding more items.

- It could be union of ___ and ___ (and more): From the other perspective (see also previous template), this template provides different types of the item. The player can extend this template by adding more items.
- It is complement of the ___: This template provides complement objects/-concepts of the item.
- It is disjoint with (opposite of) ___: This template provides the objects/concepts that are disjoint with the item.
- It is equivalent to the ___: This template provides equivalent objects/concepts to the item.

Note that in this phase we have also the notion of prohibited statements. Prohibited statements are actually those statements that most players decide to choose first. we are not interested to collect these statements all time, so we do not give the opportunity to the player (narrator) to use them.

However, the players should use these templates, as we build OWL ontologies by aid of these templates, but we give also the option to the narrator to build arbitrary sentences as well, if the templates can not be useful. These arbitrary sentences will build comments for the generated ontology.

3 Generating OWL-based Ontology

In this section, we introduce the translation mechanism that we use to generate OWL-based ontologies. The Ontology will be created for the object that the players are playing, e.g. book, computer, car. After every play using an object (item), we collect some common sense facts about that item and we can build an ontology for that. The first iteration of generating ontologies is draft and can not be considered as a complete ontology. In other words, the ontology is created during several iterations and not at the first time.

3.1 Concepts

For the approved properties, i.e. the properties that their frequencies are more than a threshold, a *owl:class* is generated. These classes are actually the transformation of properties into OWL representation using a mediator/mapper which is simply able to generate classes and their properties. Suppose a domain like a *book*: For every approved property or concept, a class and a link will be generated to associate this class to main concept which in our example is a book. Figure 1 illustrates the mapping between some selected properties and their OWL representations. As we mentioned earlier, the properties will be stored in a knowledge base (KB) and as soon as they are *mature* enough to be linked, the mapper will translate them into OWL and link them to the main concept. In the following sections, we provide a more detailed description.

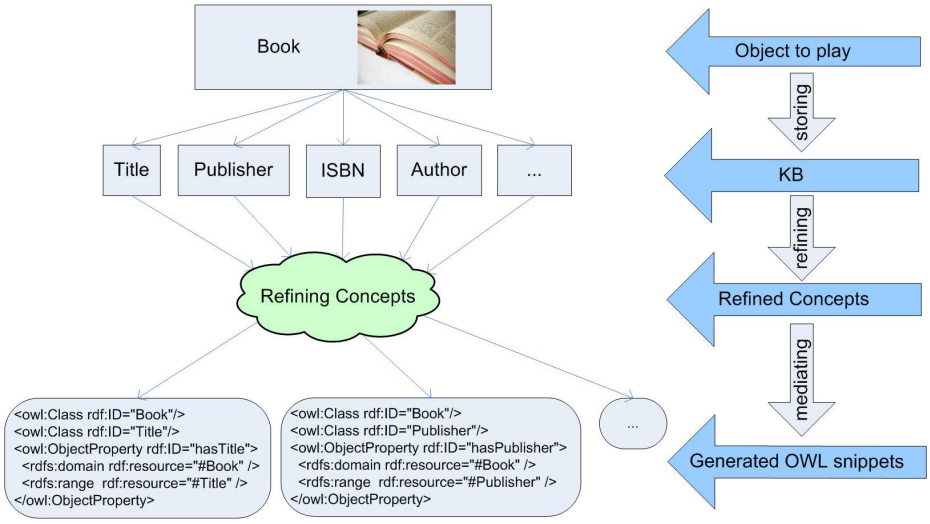

Fig. 1. Generating OWL for Properties Using a Mapper/Mediator

Pre-Refinement of Concepts (Refining Before Mediation). As we mentioned earlier, the concepts need to be refined. The refinement process is as follows: Because a specific object can be played more than once, we assign a counter to every object and the counter increases if the players are playing that object. We call this counter *objectCounter* in which the word *object* will be replaced with the explicit name of the object. A counter is also assigned to every property that the players agree upon that during the game and after further agreement by other players, the counter increases. We call this counter *objectPropertyCounter* which *object* will be replaced with the explicit name of the object and *property* will be replaced with the explicit name of the property of the object. The *variance* is defined for each property and is calculated by *objectCounter* minus *objectPropertyCounter*. If the result is greater than *threshold1*, the property will be moved to prohibited list, as many pairs agreed upon that property and if it is less than *threshold2*, the property will be deleted, as only very few pairs agreed upon that property. Note that, we do not care about uppercase and lowercase of alphabetic letters. Listing 1.1 demonstrates the pseudocode of this refinement.

Listing 1.1. Pseudocode of Refining Concepts

```
1  if (object is selected) then
2          objectCounter++;
3
4  if (objectProperty is selected) then
5          objectPropertyCounter++;
6
7  variance(objectProperty) = objectCounter - objectPropertyCounter;
8
9  if (normalize(variance(objectProperty)) > threshold1) then
10         move objectProperty to prohibited list;
11
12 if (normalize(variance(objectProperty)) < threshold2) then
13         delete objectProperty;
```

Concept Mediator/Mapper. Concept mediator/mapper is simply a mapper that gets the property or concept as input and generates OWL statements as output. The OWL statement contains also the link that associates the property to the main object. Figure 1 demonstrates some sample inputs and outputs of the mediator/mapper. However, in this step, we do not have our ontology and we have just gathered only properties and built their links. The ontology will be created after gathering sufficient facts about the object.

Post-Refining (Refinement After Mediation). After generating OWL representations of properties, they need also to be purified. Refining statements is an iterative task and tries to build a summarized version of statements based on resource URIs. Figure 2 demonstrates a sample of this post-refinement.

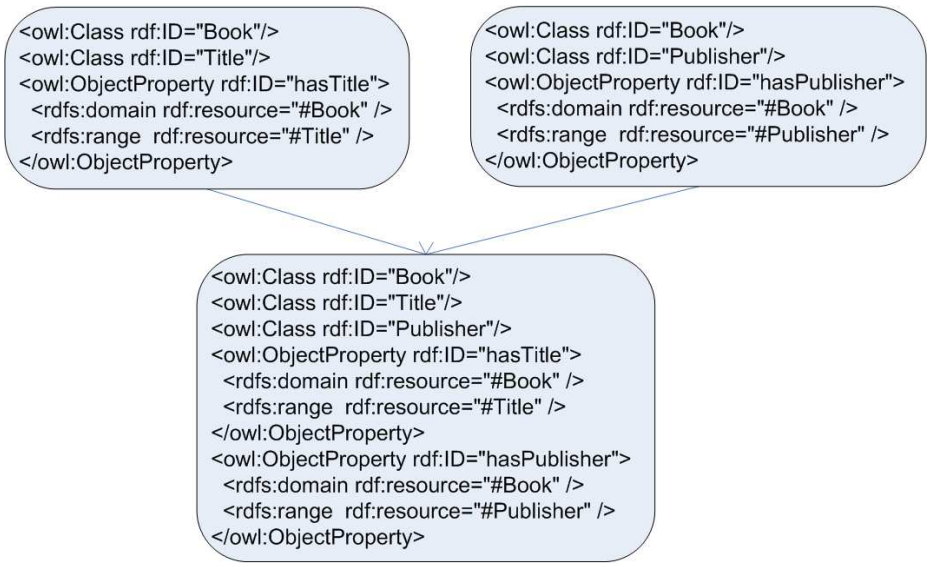

Fig. 2. Properties Refinement Sample

3.2 Statements

In the previous section, we presented the fixed templates that we use to gather common sense facts about objects. As we mentioned, those templates were carefully chosen for two main purposes: First, to be able to be translated into OWL using a mediator/mapper and second, to avoid the game being boring, as we need to entertain players, instead of assigning tasks to them. Table 1 demonstrates the general translation of templates. Note that *&xsd;* refers to XSD namespace which is actually *xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"*. To avoid a huge messy table, we decided to use acronyms.

Table 1: Templates and Their OWL Representations

Template	**Generated OWL**
X has at least ___ *Y*	<owl:Restriction> <owl:onProperty rdf:resource = "#hasY" /> <owl:minCardinality rdf:datatype = "&xsd;nonNegativeInteger"> *some value* </owl:minCardinality> </owl:Restriction>
X has at most ___ *Y*	<owl:Restriction> <owl:onProperty rdf:resource = "#hasY" /> <owl:maxCardinality rdf:datatype = "&xsd;nonNegativeInteger"> *some value* </owl:maxCardinality> </owl:Restriction>
X is kind of ___	<owl:Class rdf:ID = "*some value*"/> <rdfs:Class rdf:resource = "#X"> <rdfs:subClassOf rdf:resource = "#*some value*"/> </rdfs:Class>
X could be either ___ or ___ *(or more)*	<owl:Class rdf:ID = "*some concept*"/> <owl:Class rdf:ID = "*other concept*"/> <owl:Class rdf:ID = "*more concept*"/> <owl:Class rdf:ID = "X"> <owl:intersectionOf rdf:parseType = "Collection"> <owl:Class rdf:about = "#*some concept*"/> <owl:Class rdf:about = "#*other concept*"/> <owl:Class rdf:about = "#*more concept*"/> </owl:intersectionOf> </owl:Class>
X could be union of ___ and ___ *(and more)*	<owl:Class rdf:ID = "*some concept*"/> <owl:Class rdf:ID = "*other concept*"/> <owl:Class rdf:ID = "*more concept*"/> <owl:Class rdf:ID = "X"> <owl:unionOf rdf:parseType = "Collection"> <owl:Class rdf:about = "#*some concept*"/> <owl:Class rdf:about = "#*other concept*"/> <owl:Class rdf:about = "#*more concept*"/> </owl:unionOf> </owl:Class>
X is complement of ___	<owl:Class rdf:ID = "*some concept*"/> <owl:Class rdf:ID = "X"> <owl:complementOf> <owl:Class rdf:about = "#some concept"/> </owl:complementOf>

Continued on next page

Table 1 – continued from previous page

Template	Generated OWL
	</owl:Class>
X is disjoint with (opposite of) ___	<owl:Class rdf:ID = "*some concept*"/> <owl:Class rdf:ID = "X"> <owl:disjointWith> <owl:Class rdf:about = "#some concept"/> </owl:disjointWith> </owl:Class>
X is equivalent to ___	<owl:Class rdf:ID = "*some concept*"/> <owl:Class rdf:ID = "X"> <owl:equivalentClass> <owl:Class rdf:about = "#some concept"/> </owl:equivalentClass> </owl:Class>

Pre-Refinement of Statements (Refinement Before Mediation). The main goal of *Pre-Refinement* is to select the statements that can be translated into correct OWLs. The process is as follows: Like previous refinement, we assign a counter to an object. we call this counter *objectCounter2*. We assign also a counter to every instance of a template related to object. We call this counter *objectTInstanceCounter*. We log all instances that will be sent to guesser. If the instance was helpful and the guesser could guess the word correctly, we increase the *objectTInstanceCounter*, but if the instance was not useful and the guesser was not able to guess the word, we decrease the *objectTInstanceCounter*. We compare the *objectTInstanceCounter* with some thresholds and then we decide whether to keep, delete or move it into the prohibited list. Note that in this refinement, we do not care about uppercase and lowercase of alphabetic letters. Listing 1.2 demonstrates the pseudocode of this refinement.

Listing 1.2. Pseudocode of Refining Instances

```
if (object is selected) then
        objectCounter2++;

if (objectTInstance was helpful) then
        objectTInstanceCounter++;
else
        objectTInstanceCounter--;

variance(objectTInstance) = objectCounter2 - objectTInstanceCounter;

if (normalize(variance(objectTInstance)) > threshold3) then
        move objectTInstance to prohibited list;
if (normalize(variance(objectTInstance)) < threshold4) then
        delete objectTInstance;
```

Statement Mediator/Mapper. Statement mediator/mapper is simply a mapper that gets the template instance as input and generates OWL statements as output. The OWL statements also contain all necessary links to the main object.

Table 1 demonstrates the OWL translation of some fixed templates. Note that the italic words are those variable words that are used by the narrator.

Post-Refinement (Refinement After Mediation) and Ontology Assembler. After generating OWL representations, they need to be purified. Refining statements is an iterative task that tries to build a summarized version of statements based on resource URIs. Figure 3 demonstrates a sample of statement refinement.

As we mentioned earlier, the fixed templates are just highly-recommended proposals to be used. If they are not helpful for the narrator to help the guesser, he/she may simply use English sentences. As these sentences have no structure, we keep them as comments for the ontology, if they were helpful for guesser.

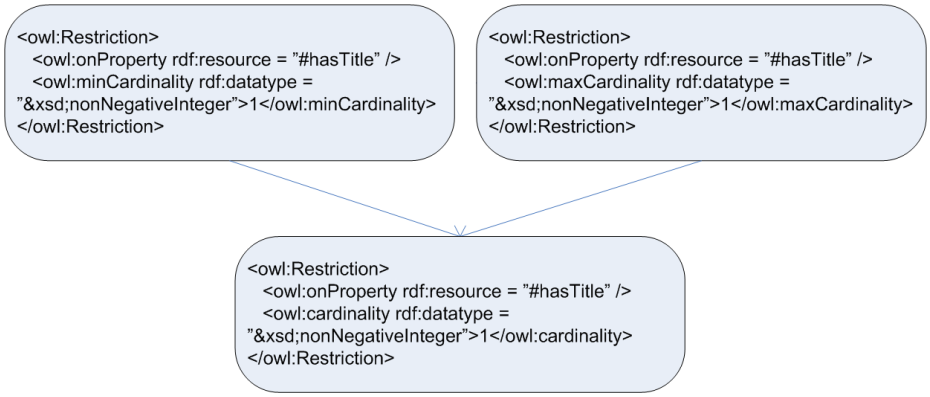

Fig. 3. Statement Refinement Sample

After all these processes, the general assembler is able to merge these statements and build the first version of the ontology. This is an iterative task and the ontology will be completed after several plays. Every Ontology has a version track using *owl:versionInfo* that enables us to keep the history of generated ontologies. Figure 4 demonstrates the iterative life cycle of generating ontologies.

4 Experimental Results

To evaluate the quality of the generated ontologies, we have checked how they change with an increasing number of rounds. To make our presentation feasible, we have reduced the number of rounds to ten and the number of concepts to two (tree and book).

In the first round (see Section 2.1), the properties *color*, *height*, and *age* were collected for the word *tree*. After ten rounds, we additionally collected *leaves* and *species*. The same test performed for the word *book* resulted in five properties:

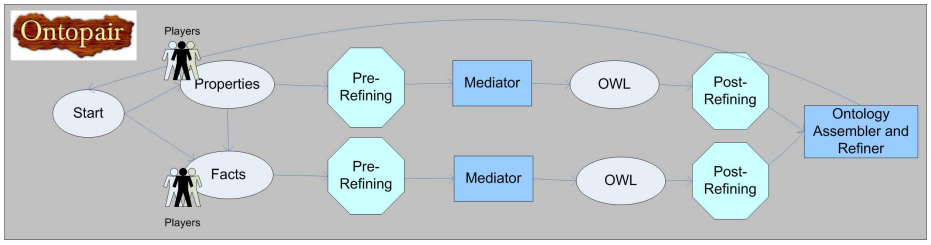

Fig. 4. Iterative Life Cycle of Generating Ontologies

author, *language*, *publisher*, *title*, and *year of publishing*. Five more rounds gave us additionally three more properties: *number of pages*, *language* and *index*. Tables 2 and 3 present the results that we have collected; we show both the words that affected the created ontology and the words that were rejected. However, the rejected words can become properties of the ontology, if we perform more rounds.

By analyzing more and more examples, we noticed that the number of properties does not grow linearly with the number of rounds. Additionally, some of the players were using plural versions of the words. This problem can be solved, however, by using dictionaries. Moreover, the results provided by the native speakers were much more accurate and they responded faster. We suggest using the lists of forbidden words; such lists impose users, specially non-English-spoken players, to use more and more sophisticated vocabularies, otherwise they stop getting points at some time. Hence, they have to learn new vocabularies.

The next part of our experiment was to evaluate the second phase (see Section 2.2), in which each person was asked a set of questions related to the common sense facts. Again we used the same words: *tree* and *book*. For the word *tree*, there were just three questions that let the players to successfully complete a round: *it is a kind of a plant*; *it has at least 1 height*; *it could be either oak or larch*. Five more rounds introduced additionally two more facts to our knowledge base: *it is disjoint with animals*; and *it has at least 1 root*. The same example for the word *book* resulted in three common sense facts in five rounds: *it has at least 1 edition*; *it has at least 1 language*; *it could be either hard-copy or electronic*. Five more rounds resulted in two new statements: *it has at least 1 author*; and *it has at least 1 title*. Again we note that more and more rounds are necessary to improve the quality of the ontologies.

Table 2: Results of Phase 1: Tree

Rounds	**Accepted Words**	**Rejected Words**
5	Color, Height, Age	Bark, Animals, Location, Kind, Fruit, Root, Branches, Green, Flower, Species, Width, Status, Leaves Falling, Seeds,

Continued on next page

Table 2 – continued from previous page

Rounds	Accepted Words	Rejected Words
		Kind
10	Color, Height, Age, Leaves, Species	Bark, Animals, Location, Kind, Fruit, Root, Branches, Green, Flower, Width, Status, Type, Name, Leaves Falling, Seeds, Kind

Table 3: Results of Phase 1: Book

Rounds	Accepted Words	Rejected Words
5	Author, Language, Publisher, title, year of publishing	Pages, Chapters, Words, Paragraph, Index, Foreword, Thickness, audience age, ISBN, Wtext, abstract, color
10	Author, Language, Publisher, title, year of publishing, number of pages, publishing, Language	Pages, Chapters, Words, Paragraph, Index, Foreword, Thickness, audience age, ISBN, text, abstract, color, cover type, domain

5 Discussions

The aim of the *OntoPair* game is to build simple ontologies for different objects that are located in images or even text-based objects in a short time. Our main concern is that the game should be entertaining to encourage people to play it. For this reason, we should avoid complex domains to be *played*. Some complicated concepts like business categorizations can be out of scope of this game, as these complicated domains may make the game boring and players will not come back again. The other point is that the generated ontologies may not contain all information regarding a domain, as the players are very ordinary people and not from Semantic Web domain. This is the main advantage of the game, as it cleverly uses people from different domains to help the Semantic Web domain experts and scientists. However, we believe that ontologies will be complicated after each play.

Even though we proposed that the players should be randomly paired, there exist some cheating potentials; players could agree to login at the same time to be paired together and maliciously annotate the objects. To avoid this case, based on previous plays, at some random times, we propose presenting specific images or texts that we know exactly the properties of objects in them and if we notice that the players are not playing honestly, we let them play as long

as they want. The same solution is foreseen for second phase of the game. As we mentioned, to increase certainty, we only assign properties and statements to objects, if and only if a certain amount of players agreed upon that. As an example, if only two players agreed upon *a car has wing* among other players, we give a low ranking to *wing* and after filtering the properties using a threshold, we omit the *wing*.

Statistics and our experiences show that word guessing games are played by many people as these games are entertaining. Many people from non-English speaking countries play these game to improve their English.

For evaluating the generated ontologies, the game can be played in single mode and the single player will play against already-generated ontologies. If generated ontologies contain sufficient knowledge, the guesser should be able to guess the correct words, otherwise a low ranking will be assigned to the generated ontology. The other approach towards evaluating OntoPair is comparing the generated ontologies with ontologies that have been created by domain experts; e.g. we can compare two ontologies for a domain like *book*, one from OntoPair repository and the other which has been generated by hand.

6 Related Works

In [11], the authors present an approach for building ontologies using a game called OntoGame. They use Wikipedia articles as conceptual entities, present them to the players, and have the users judge the ontological nature and find a common abstractions for a given entry [11]. Our approach is different, as we do not build a tree structure for objects. In two phases, we gather properties and cardinalities plus different instances of an object.

There exist also some efforts towards building a knowledge base by means of computer-based games. These games have been designed mostly for two players. The ESP game [7] tries to annotate images by enforcing players to come up with the exact objects located in images. Peekaboom [9] is another game which tries to come up with approximate location of objects in an image. Verbosity [8] is a word guessing game which composes of two players: narrator and guesser; The former should guide the latter to come up with the word that he is looking for by using some fixed templates for this purpose. Common Consensus [4] is very similar to Verbosity [8], but it has its own templates which begin mostly with *Wh** questions. Phetch [12] is another game which is composed of two players: narrator and guesser; the narrator should give guesser some keywords to help him/her to select the right image from a list of images. In other words, Phetch's main goal is finding a specific image in a bunch of similar images.

There exist also some other efforts in this general direction mostly for designing single player games. *Labelme* [2] is one example which assigns you an image for annotation. *Cyc* [1] is an artificial intelligence project that attempts to assemble a comprehensive ontology and database of everyday common sense

[1] http://www.cyc.com/

knowledge, with the goal of enabling AI applications to perform human-like reasoning [14]. Cyc offers a web-based game called *FACTory* [2] which gives the single player several sophisticated common sense facts regarding different domains and the player should mark them as true or false statements in a short time period.

At the beginning of 1980s Wille [18] initiated his work on a theory known as Formal Concept Analysis. The aim of the theory is to analysis data and identify conceptual structures among data sets. This work rapidly expanded several years later and has been successfully applied for some specific domains, e.g. bio-medicine [1]. However, such an approach often requires domain experts to approve the results.

7 Conclusion and Future Works

We have presented our work towards OntoPair, a game that uses Collective Intelligence for building OWL-based ontologies. OntoPair collects properties and common sense facts about an object in an entertaining environment and builds simple domain ontologies. We described how players should compete and how computers should process and integrate results. We also performed a simple experiment showing now our knowledge base grows. Our prototype implementation is still being implemented[3] and it needs some work in the data and user management areas. Moreover, the future work will include a reputation model that will give more impact to users who are given high esteem. Linking different ontologies together can be also considered as next phase. As an example, if we build an ontology for a *wheel*, and we have a common sense fact indicating that *a car has wheel*, we may link the car and wheel ontologies. Furthermore, we would like to perform more experiments to research how long would it take for a domain expert and ontology engineer to build an equivalent ontology. We also would like to test OntoPair in more specific domains.

Acknowledgments. The authors would like to thank Dr. Axel Polleres for his valuable comments. This work is partially supported by Ecospace (Integrated Project on eProfessional Collaboration Space) project: FP6-IST-5-35208, Lion project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I-131, and Enterprise Ireland under Grant No. *ILP/05/203*.

References

1. Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In Manuela M. Veloso, editor, *IJCAI*, pages 230–235, 2007.
2. Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: a database and web-based tool for image annotation. In *MIT AI Lab Memo AIM-2005-025*, 2005.

[2] http://207.207.9.186/

[3] `http://sourceforge.net/projects/OntoPair`

3. Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification. http://www.w3.org/TR/rdf-schema/, February 2004.
4. Henry Lieberman, Dustin Smith, and Alea Teeters. Common Consensus: A Web-based Game for Collecting Commonsense Goals. In *Workshop on Common Sense for Intelligent Interfaces, ACM International Conference on Intelligent User Interfaces (IUI-07)*, Honolulu, Hawaii, USA, 2007. ACM Press.
5. Google Inc. Google Image Labeler. http://images.google.com/imagelabeler/, 2007. Online; accessed 3-May-2007.
6. Pierre Levy. *Collective Intelligence*. Plenum Publishing Corporation, January 1997.
7. Luis von Ahn, and Laura Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the 2004 conference on Human factors in computing systems*, pages 319–326. ACM Press, 2004.
8. Luis von Ahn, Mihir Kedia, and Manuel Blum. Verbosity: a game for collecting common-sense facts. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78, New York, NY, USA, 2006. ACM Press.
9. Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM Press.
10. Sean Bechhofer, and Frank van Harmelen, and Jim Hendler, and Ian Horrocks, and Deborah L. McGuinness, and Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, February 2004. Online; accessed 2-May-2007.
11. Siorpaes Katharina, and Martin Hepp. OntoGame: Towards Overcoming the Incentive Bottleneck in Ontology Building. In *3rd International IFIP Workshop On Semantic Web and Web Semantics (SWWS '07), co-located with OTM Federated Conferences, Vilamoura, Portugal*, pages 1222–1232, 2007.
12. Luis von Ahn, Shiry Ginosar, Mihir Kedia, Ruoran Liu, and Manuel Blum. Improving accessibility of the web with a computer game. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 79–82, New York, NY, USA, 2006. ACM Press.
13. Wikipedia. Captcha — wikipedia, the free encyclopedia, 2007. [Online; accessed 14-December-2007].
14. Wikipedia. Cyc — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Cyc&oldid=125786119, 2007. [Online; accessed 7-May-2007].
15. Wikipedia. Guessing game — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Guessing_game&oldid=116214370, 2007. Online; accessed 6-May-2007.
16. Wikipedia. Human-based computation — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Human-based_computation&oldid=122965665, 2007. [Online; accessed 7-May-2007].
17. Wikipedia. Collective intelligence — wikipedia, the free encyclopedia, 2008. [Online; accessed 17-March-2008].
18. R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered Setsand in I. Rivals (Ed.)*, volume 23, 1982.