The Collections Ontology: creating and handling collections in OWL 2 DL frameworks

Editor(s): Giancarlo Guizzardi, Federal University of Espírito Santo (UFES), Brazil
Solicited review(s): Fernando Naufel do Amaral, Pólo Universitário de Rio das Ostras Universidade Federal Fluminense, Brazil, and Csaba Veres, University in Bergen, Norway

Paolo Ciccarese[a,b,]* and Silvio Peroni[c]
[a]*Massachusetts General Hospital, Department of Neurology, Mindinformatics, 65 Landsdowne Street, Cambridge, 02139 MA, USA*
[b]*Harvard Medical School, Boston, MA, USA*
[c]*Department of Computer Science And Engineering, University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna (BO), Italy*

Abstract. The RDF collections and containers is one of the most used features by RDF technicians and practitioners. Although some work has been published in past, there is not a standard and accepted way for defining collections within OWL DL frameworks. Here, we attempt to address this issue with the introduction of the Collections Ontology (CO) version 2.0. CO is an OWL 2 DL ontology developed for creating sets, bags and lists of resources, and for inferring collection properties even in the presence of incomplete information.

Keywords: OWL, collection, list, bag, set

*Corresponding author. E-mail: paolo.ciccarese@gmail.com.

1. Introduction

In mathematics and computer science, objects that group multiple elements into a single unit, e.g. sets and lists, are commonly known as collections. These entities may involve groups of non-repeatable entities (e.g., students of a particular class), unsorted and repeatable items (e.g., votes of the latest US presidential elections), and even ordered indexes of things (e.g., bibliographic reference lists of scientific papers).

The need to describe those items as belonging to particular collections occurs quite often when formalising real domains through ontologies. Semantic Web technologies (e.g., RDF [1], RDFS [2] and OWL [3]) allow the use of collections to some extent. However, problems arise when we want to define OWL 2 DL ontologies that include known constructs from underlying modelling languages (e.g., RDF sequences and bags).

Several well-known ontologies, e.g. BIBO [4], adopt the aforementioned approach. Of course, such a technique is not an option we want or need to strictly follow the formal constraints given by the OWL 2 DL specifications. In such case, a large amount of ontologies define their own structure for describing collections within OWL 2 DL frameworks. The alternative to this approach is the creation of different and interoperable ways of describing, handling, and querying upon entities defined as collections of items.

We envision two possible solutions to properly address collections modelling within OWL 2 DL:

- extend the OWL specification in order to explicitly define mechanisms for handling collections, as happened in RDF, or
- create a standard model for describing collections within OWL 2 DL frameworks, along the line of what has been proposed in [5].

We firmly believe that modifications to the OWL specification are not feasible in the short term. For this reason, here we introduce the *Collection Ontology* (*CO*), a model for creating collections and align different conceptualisations of them through classes and properties that describe sets, bags and lists of items.

Although this ontology has been first introduced in 2007 as part of the project *SWAN* (*Semantic Web Applications in Neuromedicine*) [6], it is been conceived as a separate orthogonal module with no dependencies from the rest of the SWAN ontology ecosystem.

We are here presenting the current version of the CO ontology (v. 2.0), which has been greatly improved capitalizing on the experience matured in the last four years within several projects and carefully updated to utilize many of the new features released in OWL 2 DL.

The rest of the paper is structured as follows. In Section 2 we introduce previous approaches to define collections in RDF and OWL. In Section 3 we present the Collections Ontology (CO), describing its main entities and features. Then we show its inference power (Section 4) and how to answer particular queries on CO collections through SPARQL (Section 5). In Section 6 we briefly introduce a Java API for creating and handling CO entities inside a Java application. Finally, we present projects that are making use of CO for describing different domains (Section 7) and we conclude the paper briefly discussing on future development of our work.

2. Related Works

A large amount of literature exists about Semantic Web models for handling collections of entities. In this section we discuss the most important techniques currently used to address this issue, namely: RDF containers, RDF collections, ontological patterns and OWL ontologies.

2.1. RDF Containers

RDF allows the usage of three kinds of containers[1]:

- **rdf:Bag.** A *bag* represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members. For example, a bag might be used to describe a group of part numbers in which the order of entry or processing of the part numbers does not matter.
- **rdf:Seq.** A *sequence* (or *seq*) represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant. For example, a sequence might be used to describe a group that must be maintained in alphabetical order.
- **rdf:Alt.** An *alternative* (or *alt*) represents a group of resources or literals that are alternatives (typically for a single value of a property). For example, an alt might be used to describe alternative language translations for the title of a book, or to describe a list of alternative Internet sites at which a resource might be found. An application using a property whose value is an alt container should be aware that it can choose any one of the members of the group as appropriate.

In order to show how to use these constructs, let us take into consideration the following natural language scenario:

> *The resolution was approved by the Rules Committee, having members Fred, Wilma, and Dino.*

We could describe the above scenario in RDF as follows[2]:

```
ex:resolution exterms:approvedBy
        ex:fred , ex:wilma , ex:dino .
```

However, in the above excerpt we are saying that the resolution is approved by each individual member rather than by the whole group.

Using RDF containers allows us to avoid this issue. In fact, we can use a bag for grouping people as a single unit and then saying that the group approved (property *approvedBy*) the resolution:

```
ex:resolution exterms:approvedBy ex:rules-committee .
ex:rules-committee a rdf:Bag
        ; rdf:_1 ex:fred
        ; rdf:_2 ex:wilma
        ; rdf:_3 ex:dino .
```

Of course RDF containers have some constraints. In particular, they only state that certain identified resources are members, but they cannot express whether other members that are part of the same container exist. It is not possible to

1 Please note that the current W3C Working Draft of RDF 1.1, dated June 5, 2012 (available at http://www.w3.org/TR/rdf11-concepts/), has deprecated such collections without offering an alternative solution.

2 The prefixes *ex* and *exterms* refer to fictional URLs that describe resources and vocabulary terms respectively.

exclude that there might be another graph somewhere that describes additional members.

2.2. RDF Collection

RDF provides support for describing groups containing only the specified members in the form of RDF collections. An RDF collection is a group of entities represented as a list structure (class *rdf:List*) in the RDF graph.

For instance, we can describe the group of people introduced in the example in the previous section as follows:

```
ex:resolution exterms:approvedBy ex:rules-committee .
ex:rules-committee rdf:first ex:fred
        ; rdf:rest [ rdf:first ex:wilma
        ; rdf:rest [ rdf:first ex:dino
        ; rdf:rest rdf:nil ] ] .
```

RDF imposes no "well-formedness" conditions on the use of the collection vocabulary – it is possible, for instance, to define multiple *rdf:first* elements). Thus, RDF applications that require collections to be well-formed should be written to check that the collection vocabulary is being used appropriately, in order to be fully robust.

Of course, both RDF/XML and Turtle provide compact syntaxes for describing collections that avoid the aforementioned "well-formedness" issue, as shown as follows:

```
ex:resolution exterms:approvedBy
        ( ex:fred ex:wilma ex:dino ) .
```

2.3. OWL and ordering

OWL has no support for ordering, and the natural constructs from the underlying RDF vocabulary (*rdf:List* and *rdf:nil*) are unavailable in OWL-DL because they are used in its RDF serialization. In principle, *rdf:Seq* is not illegal but it depends on lexical ordering and has no logical semantics accessible to a DL classifier.

In other terms, as stated in [7]:

- the elements in a container are defined using the relations *rdf:_1*, *rdf:_2*, and so on that have no formal definition in RDF. Using them for the purpose of reasoning will require us to define and enforce the properties of these relations;
- it is not possible to define a container that has elements only of a specific type.
- for updating a specific element in a container in a remote source, one is forced to transmit the whole container.
- it is not possible to associate provenance information with the elements in a container.

Since OWL has greater expressiveness than RDF - with constructs such as transitive properties - and reasoning capabilities - for checking the consistency and inferring subsumptions -, the idea of reasoning with sequential structures in OWL-DL looks appealing.

In [5], the authors proposed a way of representing sequential structures in OWL-DL. They argued that the representation of these structures "requires extensive rewriting, the relation of the resulting structures to the original lists is not intuitive and, more importantly, the resulting structures grow as the square of the length of the list". Then, they describe a general *list pattern* that they incorporated in the Semantic Web Best Practice Working Group's note on n-ary relations [8].

Similar patterns are introduced in [9] and are available as OWL ontologies at the Ontology Design Patterns portal[3]. Among them, the *sequence pattern* [10] seems to be particularly appropriate for describing sequential structures. In fact, it has been developed primarily for sorting time-dependant entities such as tasks, processes, spatially located objects and situations. Moreover, it defines transitive and intransitive object properties to link an entity of the sequence with its successors and predecessors.

Another not-so-logically-grounded technique for specifying order among entities makes use of literal indexes. The main idea is to aggregate entities in a collection where the order is specified by a value (usually, an integer) defined through data property assertions. For instance, the *Music Ontology* [11] uses this approach (through the data property *track_number*) to list the tracks in a record (linked to it with the object property *track*). Although this approach is very simple, it is very easy to introduce mistakes when modelling such a scenario, for example assigning the same track number to two different tracks of the same record. Usually, this technique prevents common OWL applications from checking automatically the consistency of the ontology unless implementing *ad hoc* codes.

3. The Collections Ontology (CO)

Our contribution in addressing the issue of defining and handling collections within OWL 2 DL frameworks consists in the latest version (2.0) of the Collections Ontology (CO)[4] or CO2, originally proposed as part of the SWAN Ontology Ecosystem [6]. As summarised in the Graffoo diagram[5] in Figure 1, this ontology defines classes and properties that allow one to define three different kinds of collection depending on the particular features that are requested. Namely, sets for describing collections of non-repeatable and unordered elements; bags for defining collections of repeatable and unordered elements; and lists for introducing collections of repeatable and ordered elements. However, before better defining and detailing such classes we would like to explain how CO relates to the mathematical definition of sets, multisets and sequence and to ontological theories about collectivities.

3.1. Set, multisets and sequences

According to Georg Cantor, a set is a gathering together into a whole of definite, distinct objects of our perception and of our thought – which are called elements of the set. A set can be described by extension by listing each member of the set. An extensional definition is denoted by enclosing the list of members in curly brackets:

$C = \{4, 2, 1, 3\}$

$D = \{\text{blue}, \text{white}, \text{red}\}$

---

3 http://www.ontologydesignpatterns.org

4 Available at http://purl.org/co.

5 This and all the following graphical representations of ontologies are drawn using Graffoo, the Graphical Framework for OWL Ontologies, available at http://www.essepuntato.it/graffoo. Yellow rectangles represent classes (solid border) and restrictions (dotted border), green parallelograms represent datatypes, arrows starting out of a filled circle refer to object property definitions, arrows starting out of an open circle refer to data property definitions, while other arrows represent assertions between resources.

Every element of a set must be unique; no two members may be identical and the order in which the elements of a set or multiset are listed is irrelevant.

A multiset (or bag) is a generalization of the notion of set in which members are allowed to appear more than once. The number of times an element belongs to the multiset is the multiplicity of that member. The total number of elements in a multiset, including repeated memberships, is the cardinality of the multiset. The bag {1,2} is also a set.

A sequence is an ordered list of objects (or events). Like a set, it contains members (also called elements or terms), and the number of terms (possibly infinite) is called the length of the sequence. Unlike a set, order matters, and exactly the same elements can appear multiple times at different positions in the sequence.

In general, the principle of identity operates on the elements of a collection and, if handled, on their order rather then on the collection seen as proper artefact. This means, for instance, that in mathematics two sets containing the same group of elements are the same set, two lists contains the same elements in the same order are the same list, and so on.

In CO we decided not to model the sets, multisets and sequence as extensional objects (in the mathematical sense). We introduced a superclass *co:Collection* and to split its subclasses in two disjoint groups according to their ability to consent (i.e., *co:Bag* and *co:List*) or not (i.e., *co:Set*) the repetitiveness of their elements. We therefore defined asserted – manually defined – classes that are not mapping one-to-one to the mathematical classes.

The relationships between the above mathematical entities and those defined by Collections Ontology – and detailed in the following sub-sections of the paper – can be defined as follow:

co:Set $\sqsubseteq$ Set

co:Bag $\sqsubseteq$ Bag

co:Set $\sqcap$ co:Bag = $\varnothing$

co:List = co:Bag $\sqcap$ Sequence

3.2. Part-whole relations and collectives

In order to use CO when modelling scenarios describing "collections in terms of the constructive boundaries of those plural entities that form themselves a whole" [13], we intentionally did not model the mathematical principle of identity. Therefore, it is possible to consider two sets of people (actually, *collectives* of people), composed exactly by the same people, as two different research groups without contradictions. A more extensive example of this use is shown in section 4, in which we introduce how to use this feature to leverage inference.

In the past, several works have been addressed the comparison between such mathematical collections and collectives. One of the most remarkable study in this direction is [14]. In this work, Guizzardi remarks as collectives are often considered identical to sets while they actually are not. In particular, he analyses how the classical mathematical operations of sets, i.e. the *membership* and the *subset* relations, are not able to describe the relations between an individual and a collection and between a sub-collective and a collective, respectively named as *member-collective* and *subcollective-collective* relations.

Contrary to the set membership, the member-collective relation is intransitive, which means that each member of the collective is *atomic* [15] with regard to the collective itself. Thus, from having a person *p* member of a club *c* and the club *c* member of an association of clubs *a* we cannot infer that *p* is member of *a*. The subcollective-collective relation is actually a transitive relation instead, which holds between plural entities. However, this kind of relations is irreflexive at the type level, which means that two subcollectives part of the same collective must have different characterisations (e.g. collective of the alumni of a school can have part the collective of all the male alumni and the collective of all the female alumni of that school).

The property *co:element* - introduced in Section 3.4 -, which links any collection (either a set, a bag or a list) to its members, is very general and has been defined without particular property constraints. Thus, CO leaves its users to interpret and/or restrict such a property so as to describe either the membership of sets or member-collective relations. In addition, we did not define any property to model either the subset operation or the subcollective-collective relation, thus allowing users to extend CO so as to adopt the semantics they prefer.

3.3. What is new in CO

The version 2.0 of the Collections Ontology we introduce in this article is a meaningful extension of its earliest OWL 1 version. Our main aims were to improve the definition of such an ontology through using a significant portion of the new features introduced by OWL 2.

The work described in this paper was undertaken collaboratively between both authors, PC based in Boston (US) and SP based in Bologna (IT), without face-to-face meetings. Instead we used a combination of Skype discussions, e-mail exchanges, a collaborative wiki page to record issues to be discussed and added to the ontology.

When developing such a new version, we followed all the best practices introduced in [16], which are directly inspired by *OBO Foundry Principles*[6]. In particular, the new version of the ontology:

- should be open for use by all;
- should possess a unique identifier space (namespace);
- should be published in distinct successive versions;
- should have clearly specified and delineated content;
- should be orthogonal to other ontologies;
- should include textual definitions for all terms;
- should use relationships (object and data properties) that are unambiguously defined;
- should be well documented;
- should serve a plurality of independent users;
- should be developed collaboratively.

In addition to the above guidelines, we also had to take into account particular constraints. First, we had to guarantee a backward compatibility of CO version 2.0 with its previous versions, since they are currently used in implemented systems and frameworks, as we introduce in Section 7. In addition, according to both the above constraint and an implementation standpoint, we decided to develop the data structures managing *co:Set* and *co:Bag* differently from the related mathematical entities, as introduced in Section 3.5 and 3.6, respectively.

Although this choice can be seen as odd, inconvenient or even incorrect, we decided to follow this path also to keep the

6 OBO Foundry Principles: http://www.obofoundry.org/crit.shtml.

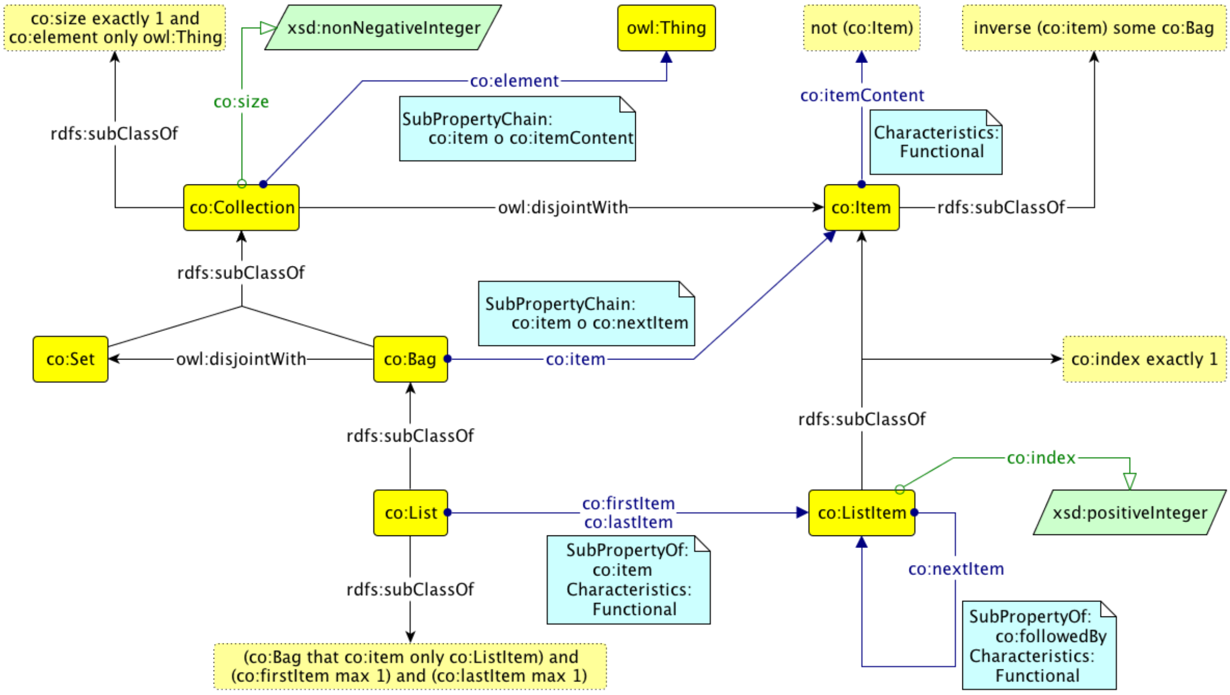

Figure 1. Diagram summarising the main structure of the Collections Ontology.

ontology as simple as possible (and, thus, easier to understand and use by final users). To be totally close to the mathematical definitions of such collections, thus keeping the original mathematical subclass-class organisation, we should define a collection as an entity having items, each referring to the particular individual member of the collection in consideration. In this way, we could say that:

- a bag is a collection having non-ordered items referring to repeatable elements;
- a set is kind of bag having non-ordered items referring to non-repeatable elements;
- a list is a collection having ordered items referring to repeatable elements.

However this organisation, even possible, would have made the new version of CO incompatible with its previous versions and would have increased the complexity when defining sets, adding an item for each of its member – even if that item would not add any particular feature to the set itself, since it is used to guarantee neither repetition nor order in this particular case.

Thus, we decided to organise CO according to a pure structural point of view, thus disjointing bags/lists – which always needs items to enable the repeatability and the order of elements – and sets – which hide such items behind a direct relation with their members through the property *co:element*.

Thus, the main improvements introduced in this version of CO, according to the principles and constraints introduced above, are:

- all the entities are assigned to a new URL base, i.e. “http://purl.org/co/”;
- the existing logical structure of the ontology has been partially re-organised;
- addition of new properties describing inverse relations and indexes of list items;
- use of new OWL 2 DL capabilities to offer a better inference layer;
- introduction of additional logical axioms and SWRL rules for improved consistency checking and integrity constraints;
- addition of natural language labels and comments for improving the human-understanding of CO;
- an accompanying ontology[7] that aligns the current version of CO with the old version developed for SWAN and with other ontologies handling collections;
- a Java API so as to load, manage and store CO collections within a Java application;

In the following subsections, we introduce all the main classes and properties defined in CO, supporting them through exemplar use cases.

3.4. Collection

The class *co:Collection* is the top-level “abstract” class of CO. Any individual of this class can only contain elements as OWL entities (i.e., individuals of the class *owl:Thing*) and must specifies a particular size (property *co:size*). It is the superclass of the “concrete” collections of CO, i.e., *co:Set*, *co:Bag* and *co:List* – we introduce in the following sections. This class and its related properties are defined as follows:

```
Class: co:Collection
        SubClassOf:
                co:element only owl:Thing,
                co:size exactly 1
        DisjointWith: co:Item
ObjectProperty: co:element
```

7 Alignment of CO to other ontologies: http://purl.org/co/alignment.

```
        Domain: co:Collection
        SubPropertyChain: co:item o co:itemContent
        InverseOf: co:elementOf
DataProperty: co:size
        Domain: co:Collection
        Range: xsd:nonNegativeInteger
```

Note that the size of a collection *C* refers to the number of times C refers to its elements. For example, the following collections – composed (property *co:element*) by the same three elements *a*, *b* and *c* – have all different sizes:

- the size of the set *{a,b,c}* is 3;
- the size of the bag *[a,b,b,c,a]* is 5;
- the size of the list *(a,b,c,a,a,c,b)* is 7.

3.5. Set

An individual of the class *co:Set* is a collection that cannot contain duplicate elements. All the elements of the set are directly linked to it through the property *co:element*, as shown in Figure 2. This class is defined as follows[8]:

```
Class: co:Set
        SubClassOf: co:Collection
```

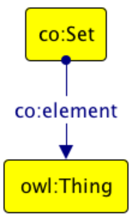

Figure 2. Diagram summarising the class *Set* and the related property *element*.

In OWL, identical elements connected by the same property are, by default, treated as items of a set.

Let us take again into consideration the example introduced in Section 2.1. Using CO sets, it is possible to describe easily that scenario as follows:

```
ex:resolution externs:approvedBy ex:rules-committee .
ex:rules-committee a co:Set
        ; co:element ex:fred , ex:wilma , ex:dino .
```

3.6. Bag

An individual of the class *co:Bag* (that is disjoint with *co:Set*) is a collection that can have multiple copies of each element. As shown in Figure 3, this is performed through the class *co:Item* and the property *co:item*. The class *co:Item* links exactly one resource that effectively is contained in the bag through the relationship *co:itemContent*. The dereferencing mechanism implemented through the properties *co:item* and *co:itemContent* allows, then, to associate a same resource to a collection more than one time. This class and its related properties are defined as follows:

```
Class: co:Bag
        SubClassOf: co:Collection
        DisjointWith: co:Set
ObjectProperty: co:item
        Domain: co:Bag
        Range: co:Item
        InverseOf: co:itemOf
        SubPropertyChain: co:item o co:nextItem
Class: co:Item
        SubClassOf: inverse co:item some co:Bag
        DisjointWith: co:Collection
ObjectProperty: co:itemContent
        Characteristics: Functional
        Domain: co:Item
        Range: not co:Item
        InverseOf: co:itemContentOf
```

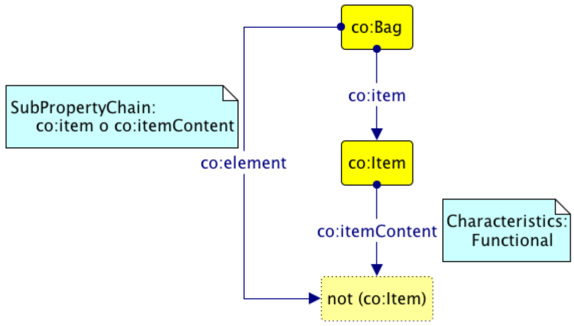

Figure 3. Diagram summarising the class *Bag* and the related class *Item* and properties *item*, *itemContent* and *element*.

Bags can be used in all those scenarios where we do not care about the order and we want to keep track of repeatability of elements. The following example introduces a simple context in which bags can be used for:

The factorisation of the number 20 is "2, 2, 5".

Since the order of the prime factors in the factorisation is not important for mathematical purposes, we can use CO bags to describe the above scenario in OWL:

```
ex:twenty exterm:hasFactorisation ex:twenty-factors .
ex:twenty-factors a co:Bag
        ; co:item ex:i1 , ex:i2 , ex:i3 .
ex:i1 a co:Item ; co:itemContent ex:two .
ex:i2 a co:Item ; co:itemContent ex:two .
ex:i3 a co:Item ; co:itemContent ex:five .
```

Moreover, by means of the OWL 2 feature for defining property chains, it has been possible to infer automatically the membership in a bag, i.e., all the *co:element* relations between a bag instance and all the other objects it effectively contains, that are dereferenced through items and the related properties *co:item* and *co:itemContent* for allowing repetition.

3.7. List

An individual of the class *co:List* (that is subclass of *co:Bag*) is an abstract data structure that implements an ordered collection of elements, where the same element may occur more than once. As shown in Figure 4, the ordering is performed through the property *co:nextItem* that links an individual of the class *co:ListItem* (subclass of *co:Item*) to exactly another one. Moreover, *co:nextItem* is accompanied by its related inverse and transitive properties. As for *co:Item*, the class *co:ListItem* links exactly one resource through the relationship *co:itemContent*.

8 This and all the following excerpts of ontology models are written according to the Manchester Syntax, while all the examples of use of the model are written in Turtle.

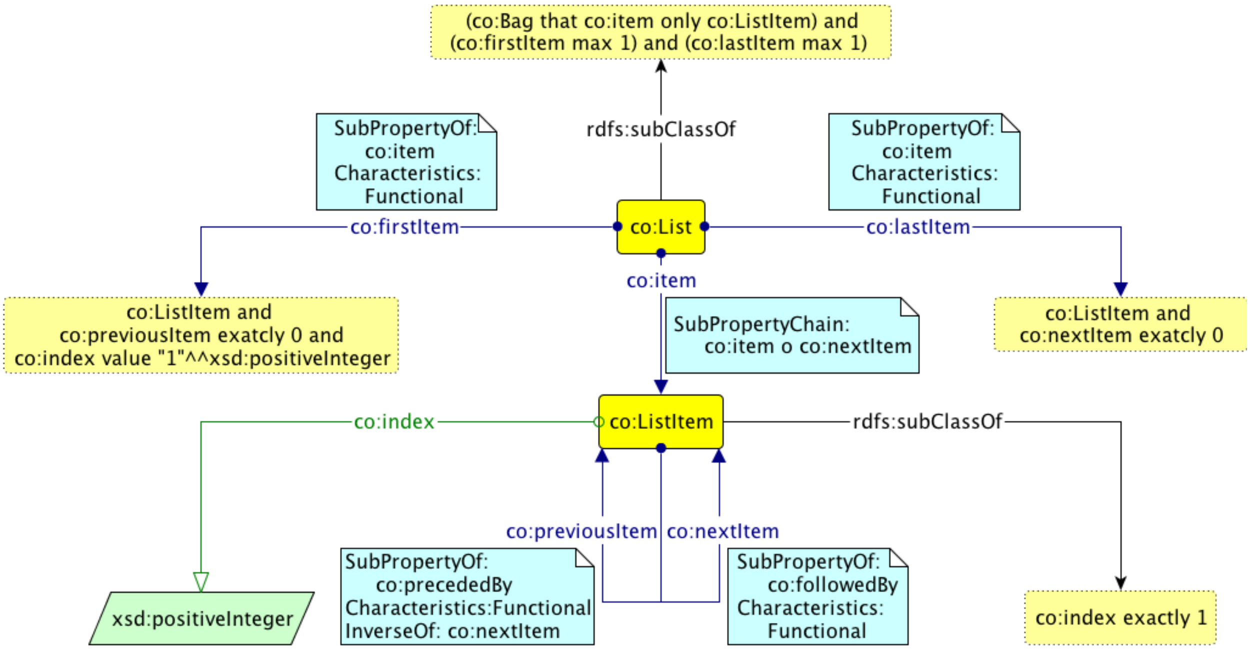

Figure 4. Diagram summarising the class *List* and the related class *ListItem* and properties.

In order to identify which are the first and the last items in a list, two object properties are defined, *co:firstItem* and *co:lastItem*, as sub-property of *co:item*. Of course, list items linked through these two properties cannot be respectively preceded or followed by another list item. This class and its related properties are defined as follows:

```
Class: co:List
        SubClassOf:
                co:firstItem max 1,
                co:lastItem max 1,
                co:Bag that co:item only co:ListItem
ObjectProperty: co:firstItem
        Characteristics: Functional
        SubPropertyOf: co:item
        Domain: co:List
        Range: co:ListItem that
                co:previousItem exactly 0 and
                co:index value 1
        InverseOf: co:firstItemOf
ObjectProperty: co:lastItem
        Characteristics: Functional
        SubPropertyOf: co:item
        Domain: co:List
        Range: co:ListItem that
                co:nextItem exactly 0
        InverseOf: co:lastItemOf
Class: co:ListItem
        SubClassOf: co:Item that co:index exactly 1
ObjectProperty: co:followedBy
        Characteristics: Transitive
        Domain: co:ListItem
        Range: co:ListItem
ObjectProperty: co:precededBy
        Characteristics: Transitive
        InverseOf: co:followedBy
ObjectProperty: co:nextItem
        Characteristics: Functional
        SubPropertyOf: co:followedBy
ObjectProperty: co:previousItem
        Characteristics: Functional
        SubPropertyOf: co:precededBy
        InverseOf: co:nextItem
DataProperty: co:index
        Domain: co:ListItem
        Range: xsd:positiveInteger
```

Let us introduce an example to show how to use CO lists for describing ordered collections. Suppose one wants to describe the paper referenced by [5] specifying its authors (e.g., through the property *dcterms:creator*) in that specific order. It is possible to model this scenario straightforwardly using a CO list as follows[9]:

```
ex:putting-owl-in-order exterm:creator ex:auth-list
        ; exterm:title "Putting OWL in Order:
        Patterns for Sequences in OWL" .
ex:auth-list a co:List
        ; co:size "7"^^xsd:nonNegativeInteger
        ; co:firstItem ex:i1
        ; co:item ex:i2 , ex:i3 , ex:i4
                , ex:i5, ex:i6
        ; co:lastItem ex:i7 .
ex:i1 a co:ListItem
        ; co:index "1"^^xsd:positiveInteger
        ; co:itemContent ex:drummond
        ; co:nextItem ex:i2 .
ex:i2 a co:ListItem
        ; co:index "2"^^xsd:positiveInteger
        ; co:itemContent ex:rector
        ; co:nextItem co:i3 . …
ex:i6 a co:ListItem
        ; co:index "7"^^xsd:positiveInteger
        ; co:itemContent ex:seidenberg .
ex:drummond a externs:Person
        ; exterm:name "Nick Drummond" .
ex:rector a exterm:Person
        ; exterm:name "Alan Rector" . …
```

9 The prefixes *xsd* and *dcterms* in the following examples refer to the XML Schema (http://www.w3.org/2001/XMLSchema#) and the DCTerms (http://purl.org/dc/terms/) vocabularies respectively.

Following this methodology, it is possible to keep separate the elements involved in a list (i.e., the authors of the paper in the previous example) and the position that those elements occupy in a particular list. This feature is particularly important when the same element can be part (at different indexes) of more than one list (e.g., a person can be first author of a paper and third author of another).

3.7.1 Leave it to the inference layer

In CO, the lists are defined in a way that is possible to consider some data as implicit, leaving to a reasoner or an inference system the job of inferring them.

For example, it is not needed to explicitly specify all the items that are involved in a list. In fact, through the following property chain axiom defined for the property *co:item*:

```
co:item o co:nextItem
```

it is possible not to specify all the items of a list, but just the first (property *co:firstItem*) and the last (property *co:lastItem*) ones. In this way, the reasoner will be able to infer all the remaining *co:item* assertions simply following the chain of *co:nextItem* defined by the list items.

Moreover, the combination of the above property chain can be very useful when combined with the following SWRL rules [12]:

```
co:itemOf(?i,?l) , co:index(?i,1)
          -> co:firstItem(?l,?i)
co:lastItem(?l,?i) , co:size(?l,?value)
          -> co:index(?i,?value)
co:itemOf(?i,?l) , co:index(?i,?value) ,
co:size(?l,?value)
          -> co:lastItem(?l,?i)
co:lastItem(?l,?i) , co:index(?i,?value)
          -> co:size(?l,?value)
co:nextItem(?i1,?i2) , co:index(?i1,?value1) ,
add(?value2,?value1,1)
          -> co:index(?i2,?value2)
co:itemOf(?i1,?l) , co:itemOf(?i2,?l) ,
co:index(?i1,?value1) , co:index(?i2,?value2) ,
add(?value2,?value1,1)
          -> co:nextItem(?i1,?i2)
```

Through this inference layer, it is then possible to complete lists even when they present partial information, in particular identifying:

- the first item of a list starting from its index;
- the last item of a list starting from its index and the related list size (and vice versa);
- the size of the list from its last item;
- indexes of items starting from their *co:nextItem* assertions (and vice versa).

3.7.2 Integrity constraints

The *transitive* properties *co:followedBy* and *co:precededBy* (super-properties of *co:nextItem* and *co:previousItem* respectively) are used to indicate all the items that follow/precede a particular item. In CO, no cycles are permitted, i.e., an item cannot either follow or precede itself. OWL 2 allows one to set this behaviour for object properties specifying them as *irreflexive*. However, it is not possible to set those two properties as irreflexive since it would violate one of the constraints needed keep the ontology in a DL framework[10].

Since the constraint on *co:followedBy* and *co:precededBy* is fundamental to keep the ontology consistent, we chose to specify integrity constraints by means of a particular model: the *Error Ontology*[11]. This ontology is a unit test that allows producing an inconsistent ontology if a particular (and incorrect) situation happens. It works by means of a data property, *error:hasError*, that denies its usage for any resource, as shown as follows:

```
DataProperty: error:hasError
        Domain: error:hasError exactly 0
        Range: xsd:string
```

In fact, by defining its domain as "all those resources that do not have any *error:hasError* assertion", a resource that asserts having an error makes automatically the ontology inconsistent[12].

By means of the Error Ontology, we can mandate the properties *co:followedBy* and *co:precededBy* to be, implicitly, irreflexive. This behaviour is implemented through the following SWRL rules[13]:

```
co:followedBy(?i,?i)
        -> error:hasError(?i, "A list item cannot be
        followed by itself")
co:precededBy(?i,?i)
        -> error:hasError(?i, "A list item cannot be
        preceded by itself")
```

4. Leveraging inference

The *Open Reuse and Exchange specification* (*ORE specification*) [17] is a standard defined by the Open Archives Initiative for describing and exchanging aggregations of Web resources.

The main concept of this specification is the *Aggregation*, i.e., a particular resource that aggregates, either logically or physically, other resources. It is also possible to use particular kinds of resources called proxies, so as to refer to a specific aggregated resource in a context of a particular aggregation. Moreover, by using proxies, we can specify an order (with an external vocabulary) for aggregated resources of an aggregation, if needed.

Let us briefly introduce the use of ORE for a real-world scenario. For instance our personal scientific library, composed by a large number of works, can be seen as an aggregation of different papers. We can use ORE to describe this scenario[14]:

```
ex:my-own-library a ore:Aggregation
        ; ore:aggregates
                ex:putting-owl-in-order
```

10 In this particular case, it is not possible to specify an object property as transitive and irreflexive at the same time.

11 Available at: http://www.essepuntato.it/2009/10/error. The prefix *error* refers to entities defined in it.

12 Of course, the Collection Ontology could be forced to be inconsistent in a simpler way that doesn't require the use of the property *error:hasError* – e.g specifying a rule such as *followedBy(?i, ?i) -> owl:Nothing(?i)*. However, we prefer to specify an error message, which can be very useful when used with automated debugging tools.

13 It is important to notice that all these rules do not work at the Tbox level and, thus, you need an Abox to be correctly applied. In addition, they also do not work with anonymous individuals since the DL safe rules constraint must hold to use SWRL rules within OWL ontologies. We are aware of this constraint and, even though all the examples in the previous sections make use of several black nodes (i.e. anonymous individuals) thus making these SWRL rules unusable, we decided to use such black nodes for the sake of clarity.

14 The prefixes *ore* and *w3* refer respectively to http://www.openarchives.org/ore/items/ and http://www.w3.org/TR/

```
                , w3:rdf-concepts
                , w3:rdf-sparql-query .
```

Another exemplar aggregation in the same context can be the bibliographic reference list of a particular article. When we are writing a scientific paper, we use to refer to bibliographic references, each of them referencing a precise paper, for explicitly citing other works in our paper. Of course, two bibliographic references, even when defined in two different papers and referring to the same work, can have associated particular (and contextual) metadata that change reference by reference. This scenario can be described in OWL through ORE as follows:

```
:paper-one-ref-list a ore:Aggregation .
:proxy1 a ore:Proxy
        ; ore:proxyIn :paper-one-ref-list
        ; ore:proxyFor ex:putting-owl-in-order
        ; dcterms:bibliographicCitation "Rector, B.
        et al. (2006). Putting OWL in Order:
        Patterns for Sequences in OWL." .
:proxy2 a ore:Proxy
        ; ore:proxyIn :paper-one-ref-list
        ; ore:proxyFor w3:rdf-concepts
        ; dcterms:bibliographicCitation "Klyne, G.
        et al. (2004). Resource Description
        Framework (RDF): Concepts and Abstract
        Syntax" .
:paper-two-ref-list a ore:Aggregation .
:proxy3 a ore:Proxy
        ; ore:proxyIn :paper-two-ref-list
        ; ore:proxyFor w3:owl2-syntax
        ; dcterms:bibliographicCitation "OWL 2 Web
        Ontology Language Structural, W3C
        Recommendation 27 October 2009" .
:proxy4 a ore:Proxy
        ; ore:proxyIn :paper-two-ref-list
        ; ore:proxyFor w3:rdf-sparql-query
        ; dcterms:bibliographicCitation "SPARQL
        Query Language for RDF, W3C Recommendation
        15 January 2008" .
```

ORE does not require to use a specific vocabulary for describing the order between proxies. Since the order in a reference list is usually important to handle, we can use CO with ORE in order to describe proxies sorting, adding the following statements:

```
:paper-one-ref-list a co:List
        ; co:firstItem :proxy1
        ; co:lastItem :proxy2 .
:proxy1 a co:Item
        ; co:itemContent ex:putting-owl-in-order.
:proxy2 a co:Item
        ; co:itemContent w3:rdf-concepts . …
```

Of course, *ore:Aggregation* and *co:List* are used in a very redundant way in the above excerpts. Adding an additional layer of ontological alignment between the two ontologies can help in obtaining the same set of data writing just some of them. For instance, we can add the following (Manchester Syntax) axioms to ORE with the explicit goal of leveraging inference:

```
Class: ore:Aggregation
        EquivalentTo: co:Set or
                (co:Bag that
                        co:item only ore:Proxy)
ObjectProperty: ore:aggregates
        EquivalentTo: co:element
ObjectProperty: ore:proxyIn
        EquivalentTo: co:itemOf
ObjectProperty: ore:proxyFor
        EquivalentTo: co:itemContent
```

In this way, it becomes possible to re-write a less verbose definition of the first reference list of the above examples as follows:

```
:paper-one-ref-list a ore:Aggregation
        ; co:firstItem [
                dcterms:bibliographicCitation
                "Rector, B. et al. (2006). Putting
                OWL in Order: Patterns for Sequences
                in OWL."
        ; co:nextItem [
                dcterms:bibliographicCitation
                "Klyne, G. et al. (2004). Resource
                Description Framework (RDF):
                Concepts and Abstract Syntax" ] ] .
```

5. Querying CO datasets

CO allows one to make very sophisticated SPARQL queries [18] to datasets containing information structured as CO collections. In this section we introduce just few query samples, of incremental complexity, in order to highlight how CO is able to treat even complicated scenarios. In the next examples, we take into consideration the data described in Section 3.7.

Query: “Give me all the author collections containing persons named ‘Alan Rector’”.

```
SELECT DISTINCT ?collection
WHERE {  ?paper exterm:creator ?collection .
        ?collection co:element [ a exterm:Person
        ; exterm:name “Alan Rector” ] }
```

Query: “Give me all the papers written by persons named ‘Alan Rector’ and not ‘Nick Drummond’”[15].

```
SELECT DISTINCT ?paper
WHERE {  ?paper exterm:creator ?collection .
        ?collection co:element [ a exterm:Person
                ; exterm:name “Alan Rector” ]
        FILTER NOT EXIST { ?collection co:element [
                a exterm:Person
                ; exterm:name “Nick Drummond” } }
```

Query: “Tell me how many author lists contain persons named ‘Alan Rector’”.

```
SELECT (COUNT(DISTINCT ?item) AS ?number)
WHERE {  ?paper exterm:creator [
                ?item a co:ListItem
                ; co:itemContent [ a exterm:Person
                ; exterm:name “Alan Rector” ] ] }
```

15 In the following SPARQL query we use the construct “FILTER NOT EXISTS” to get out the correct answer. This approach only works because the SPARQL processor evaluates the query according to a *close-world* point of view, contrarily to what is prescribed by OWL ontologies in general, that strictly follow the *open-world* assumption. Thus, it is important to clarify there is nothing in the Collection Ontology that allows a reasoner to prove a list does not contain a particular.

Query: "Give me all the author lists where persons are named 'Alan Rector' are either first or second author".

```
SELECT DISTINCT ?list
WHERE {  ?paper exterm:creator ?list .
        ?author a externs:Person
                externs:name "Alan Rector" .
        ?list co:firstItem ?first .
        { ?first co:itemContent ?author }
        UNION
        { ?first co:nextItem [
                co:itemContent ?author ] } }
```

Query: "Give me all the papers and their respective authors ordered by their positions."

```
SELECT DISTINCT ?paper ?person
WHERE {  ?paper exterm:title ?title
                ; exterm:creator [ co:item [
                        co:index ?position
                        co:itemContent ?author ] ]
} ORDER BY ?title ?position
```

6. A Java API

Even when an ontology is well-developed and useful to describe a particular domain, it still remains just a theoretical model if it is not accompanied by an API that allows one to use the model inside software applications. To this end, we developed a complete and extensible Java API for CO[16]. It allows one to create/modify and load/store CO entities directly from a Java code. It is composed by a base package (i.e., "org.purl.co") that implements the core classes for handling CO collections in Java. Moreover, it includes general interfaces for loading/storing an environment of CO collections from/into files or input/output streams.

Our API is a general-purpose library that is easy to be integrated with any other RDF/OWL APIs such as Jena [19] and OWLAPI [20]. This is possible by implementing the interfaces *COReader* and *COWriter* so as to have mechanism to handle RDF resources through the favourite Java library.

In the following excerpts, we introduce the use of our own Jena extension to the CO API. The first thing to do is to create a new CO environment (interface *COEnvironment*) in which we can handle collection of Jena resources (interface *Resource*):

```
COEnvironment<Resource> env =
        new StandardCOEnvironment<Resource>();
```

Each CO environment makes available all the methods for creating new CO collections, i.e. sets (method *createCOSet*, that returns a *COSet* object), bags (method *createCOBag*, that returns a *COBag* object) and lists (method *createCOList*, that returns a *COList* object). Through these interfaces and methods, the creation of the list introduced in Section 3.3 becomes straightforward:

```
Model m = ModelFactory.createDefaultModel();
String ex = "http://www.example.com/ex/";
COList<Resource> auth-list =
        env.createCOList(
                URI.create(ex+"auth-list"));
auth-list.add(m.createResource(ex+"drummond"));
auth-list.add(m.createResource(ex+"rector")); …
auth-list.add(m.createResource(ex+"seidenberg"));
```

16 Available at: http://code.google.com/p/collections-ontology/downloads

Finally, it is possible to load/store an environment from/into files or other input/output streams through implementation of the interfaces *COReader* and *COWriter*. For instance, to store the previous defined list in a particular file using the Turtle format we need to create a new writer, specifying the destination format, and then store the environment in a file:

```
COWriter<Resource> writer =
        new JenaRDFWriter(Format.Turtle);
File destination = new File("mylist.ttl");
writer.store(env,destination);
```

Beside these basic operations, the API implements Java methods and classes for all the OWL properties and classes defined in CO. Moreover, it includes mechanisms to guarantee the correctness and consistency of all the collections one creates.

7. Who is using CO

The Collections Ontology has been already adopted by the Semantic Web applications and projects introduced in this section.

7.1. SWAN

The SWAN project[17] (Semantic Web Applications in Neuromedicine) aims to develop a practical, common, semantically structured framework for biomedical discourse initially applied, but not limited, to significant problems in Alzheimer Disease (AD) research. AlzSWAN[18] an AD knowledge base created in collaboration with the Alzheimer Research Forum[19] represents the most popular instance of the SWAN platform. It consists in a network of about 2400 research statements linked to about 2700 publications.

The SWAN biomedical discourse ontology [6] represents the backbone of the project. The purpose of SWAN is to function as the schema of a distributed knowledgebase in AD, and to link information in that knowledgebase with other information in biomedicine. Back in 2007, the SWAN ontology has been architected as a set of orthogonal modules that combines into the SWAN ontology ecosystem.

One such module was the first version of Collections Ontology as collections are necessary to manage several aspects of the scientific discourse modeling. For example, a scientific argument can be represented by a sequence of research statements such as hypothesis, claims and questions. And their order is crucial as a way to convey the hypothesis properly.

The SWAN platform features have been incrementally embedded in the new Domeo Annotation Toolkit[20] [21] an extensible web application enabling users to visually and efficiently create and share ontology-based stand-off annotation on HTML or XML document targets. Domeo

17 Available at: http://swan.mindinformatics.org

18 Available at: http://hypothesis.alzforum.org

19 Available at: http://alzforum.org

20 Available at: http://annotationframework.org

supports manual, fully automated, and semi-automated annotation with complete provenance records, as well as personal or community annotation with access authorization and control. Domeo uses the SWAN ontology and Collections Ontology for representing scientific discourse.

7.2. EARMARK

The *Extremely Annotational RDF Markup* (*EARMARK*) [22-23] is a new markup meta-language defined by means of Semantic Web technologies. The basic idea is to model EARMARK documents as collections of addressable text fragments, and to associate such text content with OWL assertions that describe structural features as well as semantic properties of (parts of) that content. As a result EARMARK allows not only documents with single hierarchies (as with XML) but also multiple overlapping hierarchies where the textual content within the markup items belongs to some hierarchies but not to others. Moreover, EAMARK makes it possible to add semantic annotations to the content though assertions that may overlap with existing ones.

EARMARK is defined by an OWL ontology[21] that models all the classes and properties for describing typical markup structures, such as elements, attributes, text nodes, parent-child relations and the like. From an ontological perspective, EARMARK documents are just ABox of cited ontology.

One of the most important features that must be supported in document markup languages is the possibility of specifying a particular order between items (e.g., elements and attributes). The EARMARK ontology implements this feature importing the (old version) of CO. This makes it possible to handle markup items as collections of other ordered or unordered, repeatable or non-repeatable items.

A new version of EARMARK (both the ontology and its Java API[22]) is now in-development with the aim of adopting the current version of CO, so as to take advantage from all its new features and inferential power.

7.3. SPAR

The *Semantic Publishing and Referencing Ontologies* (*SPAR*)[23] is a suite of orthogonal and complementary OWL 2 DL ontology modules. They together permit the creation of comprehensive machine-readable RDF metadata for all aspects of semantic publishing and referencing: documents description, types of citations and their related contexts, bibliographic references, document parts and status, agents' roles and workflow processes, etc.

Some of the SPAR ontologies, such as the *FRBR-aligned Bibliographic Ontology* (*FaBiO*) [24], suggest explicitly to use CO for handling scenarios in which specifying an order among entities is mandatory (e.g., the list of the authors of a paper). Others, such as the *Bibliographic Reference Ontology* (*BiRO*), import directly CO for handling particular purposes, such as describing reference lists in research articles.

8. Conclusions

One of the most important and used features of existing RDF data is the possibility of defining collections and containers to group resources as one entity. This characteristic has not been included in OWL since its beginning, even in its latest OWL 2 DL specification. Alternative proposals has been done in past for addressing this issue, but it seems they do not come to develop a shared standard for defining collections within OWL DL frameworks.

In this paper we introduced the *Collections Ontology* (*CO*) version 2.0, our OWL 2 DL ontology developed specifically for addressing the issue of defining collection in OWL frameworks. In particular, we introduced the graphical and formal description of the ontology and we provided examples of usage in terms of Abox modelling, inferences and SPARQL queries. In addition to what we illustrated here, more information and examples are also available on the project website[24].

One of the most immediate future developments for our work is the extension of the Java API so as to release libraries to be used with other Java OWL environments, such as OWLAPI, as well as the porting of the current API in different program languages.

9. Acknowledgements

We thank the editor, the reviewers and Jonathan Rees for their valuable feedback, Tim Clark for his support and finally to some anonymous experts who gave us several suggestions in terms of terminology, modelling and presentation.

References

[1] Carroll, J., Klyne, G. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004. World Wide Web Consortium. http://www.w3.org/TR/rdf-concepts/ (last visited December 27, 2012).

[2] Brickley, D., Guha, R.V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004. World Wide Web Consortium. http://www.w3.org/TR/rdf-schema/ (last visited December 27, 2012).

[3] Motik, B., Patel-Schneider, P. F., Parsia, B. (2009). OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation, 27 October 2009. World Wide Web Consortium. http://www.w3.org/TR/owl2-syntax/ (last visited December 27, 2012).

[4] D'Arcus, B., Giasson, F. (2009). Bibliographic Ontology Specification. Specification Document, 4 November 2009. http://bibliontology.com/specification (last visited December 27, 2012).

[5] Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H. H., Seidenberg, J. (2006). Putting OWL in Order: Patterns for Sequences in OWL. In Grau, B. C., Hitzler, P, Shankey, C., Wallace, E. (Eds.), Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2006). Aachen, Germany: Sun SITE Central Europe. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-216/submission_9.pdf (last visited December 27, 2012).

[6] Ciccarese, P., Wu, E., Kinoshita, J., Wong, G., Ocana, M., Ruttenberg, A., Clark, T. (2008). The SWAN Biomedical Discourse Ontology. Journal of Biomedical

21 Available at: http://www.essepuntato.it/2008/12/earmark

22 Available at: http://palindrom.es/phd/research/earmark

23 Available at: http://purl.org/spar

24 Collections Ontology (CO) Google Code project available at: http://code.google.com/p/collections-ontology

Informatics, 41 (5), 739-751. DOI: 10.1016/j.jbi.2008.04.010.

[7] Chaudhri, V. K., Jarrold, B., Pacheco, J. (2006). Exporting Knowledge Bases into OWL. In Grau, B. C., Hitzler, P, Shankey, C., Wallace, E. (Eds.), Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2006). Aachen, Germany: Sun SITE Central Europe. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-216/submission_34.pdf (last visited December 27, 2012).

[8] Hayes, P., Welty, C. (2006). Defining N-ary Relations on the Semantic Web. W3C Working Group Note, 12 April 2006. World Wide Web Consortium. http://www.w3.org/TR/swbp-n-aryRelations/ (last visited December 27, 2012).

[9] Presutti, V., Gangemi, A. (2008). Content Ontology Design Patterns as practical building blocks for web ontologies. In Li, Q., Spaccapietra, S., Yu, E. S. K., Olivé, A. (Eds.), Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008). Heidelberg, Germany: Springer.

[10] Gangemi, A. (2010). Submission: Sequence. http://ontologydesignpatterns.org/wiki/Submissions:Sequence (last visited December 27, 2012).

[11] Raimond, Y., Giasson, F. (2010). Music Ontology Specification. Specification Document, 28 November 2010. http://musicontology.com/ (last visited December 27, 2012).

[12] Horrocks, I., Patel-Schneider, P. F., Boley, H. Tabet, S., Grosof, B., Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. World Wide Web Consortium. http://www.w3.org/Submission/SWRL/ (last visited December 27, 2012).

[13] Bottazzi, E., Catenacci, C., Gangemi, A., Lehmann, J. (2006). From collective intentionality to intentional collectives: An ontological perspective. In Cognitive Systems Research, 7 (2&3): 192-208. DOI: 10.1016/j.cogsys.2005.11.009.

[14] Guizzardi, G. (2011). Ontological Foundations for Conceptual Part-Whole Relations: The Case of Collectives and Their Parts. In H. Mouratidis, C. Rolland (Eds.), Proceedings of the 23rd International Conference (CAiSE 2011): 138–153. Berlin, Germany: Springer. DOI: 10.1007/978-3-642-21640-4_12

[15] VIeu, L., Aurnague, M. (2007). Part-of relations, functionality and dependence. In M. Aurnague, M. Hickmann, L. Vieu (Eds.), The categorization of spatial entities in language and cognition: 307–336. Amsterdam, Netherlands: John Benjamins.

[16] Ciccarese, P., Shotton, D., Peroni, S., Clark, T. (2013). CiTO + SWAN: The Web Semantics of Bibliographic Records, Citations, Evidence and Discourse Relationships. To appear in Semantic Web – Interoperability, Usability, Applicability. DOI: 10.3233/SW-130098

[17] Lagoze, C., Van de Sompel, H., Johnston, P., Nelson, M., Sanderson, R., Warner, S. (2008). Abstract Data Model. Object Reuse and Exchange Specification. Open Archives Initiative. http://www.openarchives.org/ore/1.0/datamodel (last visited December 27, 2012).

[18] Harris, S., Seaborne, A. (2011). SPARQL 1.1 Query Language. W3C Proposed Recommendation, 08 November 2012. World Wide Web Consortium. http://www.w3.org/TR/sparql11-query/ (last visited December 27, 2012).

[19] Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K. (2004). Jena: implementing the semantic web recommendations. In Feldman, S. I., Uretsky, M., Najork, M., Wills, C. E. (Eds.), Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters (WWW 2004). New York, New York, USA: ACM.

[20] Horridge, M., Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. In Semantic Web – Interoperability, Usability, Applicability, 2 (1): 11-21. DOI: 10.3233/SW-2011-0025.

[21] Ciccarese, P., Ocana, M., Clark, T. (2011). DOMEO: a web-based tool for semantic annotation of online documents. Paper at Bio-Ontologies 2011, Vienna, Austria.

[22] Di Iorio, A., Peroni, S., Vitali, F. (2011). A Semantic Web Approach To Everyday Overlapping Markup. In Journal of the American Society for Information Science and Technology, 62 (9): 1696-1716. DOI: 10.1002/asi.21591.

[23] Di Iorio, A., Peroni, S., Vitali, F. (2011). Using Semantic Web technologies for analysis and validation of structural markup. In International Journal of Web Engineering and Technologies, 6 (4): 375-398. DOI: 10.1504/IJWET.2011.043439

[24] Peroni, S., Shotton, D. (2012). FaBiO and CiTO: ontologies for describing bibliographic resources and citations. In Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 17 (December 2012): 33-43. DOI: 10.1016/j.websem.2012.08.001