Ontology Visualization Tools to Assist in Creating and Maintaining Textual Term Definitions

Daniel R. Schlegel*, and Peter L. Elkin

Department of Biomedical Informatics, University at Buffalo, SUNY, Buffalo, NY, USA

ABSTRACT

Textual definitions are often missing or incomplete in ontologies. Visualizing the ontology structure can assist ontology developers in creating or modifying definitions, but existing tools are not sufficient. Tools such as Protégé don't show enough context to be helpful, and the "show context and distort" technique used in existing visualizations systems is insufficient. We propose the use of the CSNePS GUI with its relation-based graph exploration tools to visualize ontologies for the purpose of definition maintenance and creation. We also propose an algorithm to order the undefined ontology terms to allow ontology developers to be most efficient in their definitional duties.

1 INTRODUCTION

Textual definitions in ontologies have often been shown to be incorrect, inconsistent, insufficient, or missing. This is unfortunate since definitions[1] play an important role both to the designers and users of ontologies. For designers, definitions of higher level concepts assist in the creation and definition of lower level concepts. Users of ontologies often rely on textual definitions to understand the meaning of a term.

It may seem to many ontology designers that the logical definitions, specified by the properties of a term and its place in the term hierarchy, are enough of a definition in some cases. It has been shown that there is an important correspondence between textual and logical definitions, and each can provide an accuracy check on the other (Seppälä *et al.*, 2014). It's important therefore that both be present, and that they be consistent with each other.

We are particularly interested in two use cases. In the first, a textual definition is missing, while a logical definition exists. In this case the ontology designer should try to construct a textual definition from an understanding of a term and its surrounding context in the ontology. This seems to (for better or worse) align with the current workflow for creating ontologies. In the second use case, an ontology designer reviews existing textual definitions, comparing them with the logical definitions through, again, exploring the surrounding context in the ontology.

The creation of definitions is labor intensive, and requires navigating the ontology in tools not designed to assist in definition-making. Protégé (Stanford Center for Biomedical Informatics Research, 2015) is the most commonly used ontology development tool. Unfortunately, it is not particularly well suited to definition creation or maintenance. Protégé makes it easy to see the properties of a single term, and to see that term's place within the class hierarchy. In creating definitions, this is not enough: the definition writer needs to see the properties of siblings and parents to understand what makes the term unique. The need to see all of this data at once indicates that a graph representation may be the ideal method to visualize ontology segments.

The only tool that the authors are aware of which has been designed to assist users in definition creation is TermGenie (Dietze *et al.*, 2014). TermGenie is a tool (currently used only for the Gene Ontology) which is meant to make it easy to add new terms to an ontology when needed. The extent of its assistance for definition creation is the use of "textual definition building blocks."

There have been many visualization methods developed to view and interact with ontologies (see (Katifori *et al.*, 2007) and (Katifori *et al.*, 2014) for discussions on the subject). The main issue in visualizing ontologies is that ontologies may be very large, and depending on the task, the user needs different (and different amounts of) context surrounding their current focus. Silva *et al.* (2012) have made clear that one visualization strategy does not fit every use case.

We propose the use of features present in the graphical user interface (GUI) for the CSNePS knowledge representation and reasoning system (Schlegel and Shapiro, 2012) which have been designed to assist ontology developers in the task of writing textual definitions. In the following sections we will first discuss what makes a good term definition, followed by a brief introduction to CSNePS and its GUI. The yardstick by which a system for assisting with definitions will be measured is given in Section 4. In Section 5 we will explore the usefulness of viewing context around a term of interest. We will then introduce two features of the CSNePS GUI designed to make term definition easier: the ability to easily explore ontological terms and their properties within a graph visualization, and to easily pick out undefined terms in an ontology in an order optimal for writing definitions. We close with a discussion of these features and what we imagine their impact to be within the ontology community.

2 CRITERIA FOR GOOD TERM DEFINITIONS

An internet search quickly yields hundreds of articles on how one should write a good definition in English. The easiest and most successful formulation is likely the Aristotelian definition. An Aristotelian definition is made up of two parts: the *genus*, and the *differentia*. The genus is the class of thing being defined. The differentia are what differentiates the item under definition from other members of the genus. For example, a "tabby" is defined as "a cat whose fur is mottled or streaked with dark stripes." In this definition "cat" is the genus (tabbies are cats), and "fur is mottled or streaked with dark stripes" is the differentia, distinguishing tabbies from other types of cats.

A recent book on building ontologies with the Basic Formal Ontology (Arp *et al.*, 2015) expands on the criteria for good definitions through a series of best practices, framed specifically in terms of ontology. The principles they outline are as follows:

1. Provide all nonroot terms with definitions

*To whom correspondence should be addressed: drschleg@buffalo.edu

[1] When we say "definition" in this paper, we'll be referring to textual definitions. If we are referring to logical definitions, we'll make that explicit.

2. Use Aristotelian definitions
3. Use essential features in defining terms
4. Start with the most general terms in your domain
5. Avoid circularity in defining terms
6. To ensure intelligibility of definitions, use simpler terms than the term you are defining
7. Do not create terms for universals through logical combination
8. Definitions should be unpackable

These principles distill down into three categories. First, what terms to define. All terms which are not a root node should have a definition.[2] Item 7 is also related to this, but we will ignore it for the purposes of this paper since we aren't concerned with ontology creation in general. Next, what makes a good definition. As we've discussed, Aristotelian definitions are preferred. These definitions lend themselves nicely to the concept of unpacking. For example, if "A" is defined as "a B that Cs", and if "A" appears in some text, then that text should be equally understandable if "A" is replaced with "a B that Cs." A definition for a term should discuss the most central (essential) features the term – those that make the term what it is. Superfluous detail should be avoided in favor of conciseness. Definitions should never be circular. That is, the definition should not contain the term being defined, or any near synonym of the term. Along a similar vein, the language used in a definition should be simpler than the term you're defining. The last category is how to go about doing the defining. You should start with the most general terms and work down, as more specific terms rely on the general ones.[3]

The tools discussed within this paper are meant to assist ontology developers with items from each of these three categories.

3 CSNEPS AND ITS GUI

The CSNePS user interface has been designed for interacting with the CSNePS knowledge representation and reasoning system (Schlegel, 2015). While the features discussed in this paper do not directly rely on the CSNePS system, some details would seem idiosyncratic without some brief background material.

One of the core knowledge representation ideas which CSNePS espouses is that a knowledge base may be seen as simultaneously a set of logical assertions, a set of frames, and a *propositional graph*. In the tradition of the SNePS family (Shapiro and Rapaport, 1992), a propositional graph is a directed graph in which nodes represent, among other things, propositions and logical formulas, while edges serve to indicate the roles played by components of the propositions and formulas. Every node is labeled with an identifier. Nodes representing individual constants, proposition symbols, function symbols, or relation symbols are labeled with the symbol itself. Nodes which stand for relations are given a label of the form `wft`i`!`, where "`wft`" stands for "well-formed term"[4] and the appended "`!`" indicates that a proposition is asserted in the knowledge base. Edges are drawn from `wft` nodes to each of their arguments, and are labeled with the role the target of the edge plays in the relation. For binary relations, the `wft` node may be "collapsed", and an edge can be drawn from the node for the first argument of the relation to one for the second. This edge can be labeled with the relation name itself.

The GUI for CSNePS is built around the tri-view of knowledge bases used by CSNePS. In this paper we will only concern ourselves with the graph view. While currently the features of the GUI rely on CSNePS, in the future we intend to make many features of the GUI agnostic to the underlying representation system.

4 METHODS

In order to achieve the goal of writing good definitions, it is important for the definition writer to understand the term being defined. Some part of this can be achieved by examining the existing terminological structure of the ontology,[5] and various properties on the term under definition and terms around it. To reduce the cognitive load on the writer, the more relevant context which is visible at once, the better. As was mentioned in the introduction, this is where Protégé falls short, making only a small amount of context visible at once.

While not enough context makes definition writing difficult, too much is perhaps even worse. Ontologies are large, and much of the contents may be irrelevant for any specific term's definition. Making too much of the context available to the user simply crowds out the relevant context.

Given the interconnectedness of ontologies and the many relations that are present, the most obvious method for visualizing them is to use a graph. OWL ontologies, which we focus on here, have many different syntaxes, and some are more amenable to graphing than others. Given the tri-view representation of CSNePS discussed in the previous section, we prefer to use the OWL Functional-style syntax (Bock *et al.*, 2009). This syntax makes available a set of logical relations for ontological data, along with names of the roles each argument of the relation play in the relation itself. This is then easily graphed.

Part of writing good definitions is writing them in an ordered way. It is non-trivial in current tools to list the terms which are not yet defined, and no tools sort this list into something which can be mapped to the principle given in Section 2: work from most general to least general terms. A graph representation with well-defined relations, such as `SubClassOf` in OWL, can make this possible.

5 VISUALIZING CONTEXT

It is a basic tenant of all graph visualization systems that there must be some method for visualizing the context of a node (or nodes) of interest. In ontology visualization tools this entails visualizing some number of relations outward from the term or terms of interest. Unfortunately, if this is not done in an intelligent way, the result is nearly as bad as if such a capability did not exist at all.

In designing tools to assist with definition creation we first took a naïve approach to visualizing context, allowing users to select some degree of relations outward from a term which they were interested in. The software simply shows nodes for all relations which the term is in, and continues doing this for each newly-shown term up to the

[2] The authors recognize that some terms may be indefinable, but textual definitions should be given to the greatest extent possible.

[3] For a much more detailed discussion of definitions, see Arp *et al.* (2015) pages 68–76.

[4] The word "term" in CSNePS refers to the fact that CSNePS uses a term logic, this is different from the sense of "term" used in ontological parlance.

[5] See (Elkin, 2012) for an overview of terminologies.

depth given by the user. We quickly found that context alone was not appropriate.

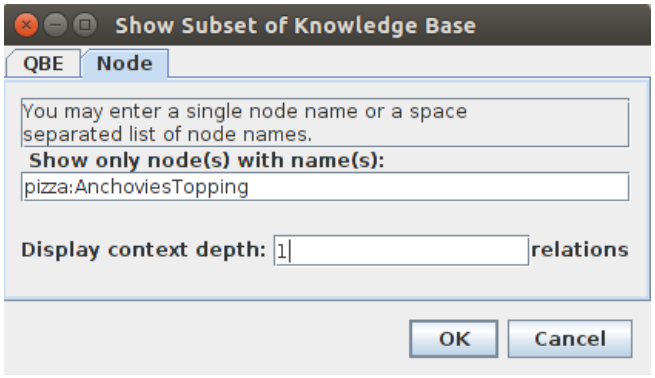

Fig. 1. The user interface for showing degrees of context of a term.

Consider an example from the commonly used pedagogical pizza ontology (Horridge *et al.*, 2009).[6] In Figure 1 we indicate that the node we are interested in is the `pizza:AnchoviesTopping` term, and we would like to see one degree of context. The resulting graph is shown in Figure 2.

The one-degree graph is fairly accessible and easy to understand. The ontology developer can see that `pizza:AnchoviesTopping` is a subclass of `pizza:FishTopping` and that it's disjoint from the classes `pizza:MixedSeafoodTopping` and `pizza:PrawnsTopping`. This would all be accessible using Protégé and selecting a single term in the class hierarchy. The power of context doesn't reveal itself until at least two levels.

Unfortunately, in large ontologies, even two levels of context is too much to view nicely. In Figure 3 you can see that the result is a hairball of unintelligible relations. Graph visualization systems generally, in addition to context, include some method for distorting the graph so that the user can easily access the parts of context they are interested in. The CSNePS GUI does contain methods for distorting the graph with a "lens", but we've found it less than useful. We argue that if it were possible to show only the desired context, there would be no need to hide portions of it through distortion.

6 ONTOLOGY GRAPH EXPLORATION

As discussed, visualizing context without constraint has limited usefulness in our application. In fact, for definitions, what we really want to visualize are any of the logical properties on terms which may help us pick out essential features. We believe combining context visualization with relation-based graph exploration may be the ideal method for viewing these essential features.

Consider again the `pizza:AnchoviesTopping` example from the previous section. Even at a single level of context as seen in Figure 2 there were portions of the graph which weren't helpful for definition building. Instead, we believe it is more appropriate to first consider the `pizza:AnchoviesTopping` node with no context, then expand the relations we believe contain essential features, one or more at a time, and repeating this process until the desired context is visible.

We have designed a mechanism in the CSNePS GUI to make this easy. The user can easily choose to show a selected set of relations attached to a term, as shown in Figure 4. This is done simply by right-clicking on a node and selecting to show edges connected to the node. Each time the user does this, a single level of that relation is expanded. The user may do this as many times as they like. The set of relations given are simply the OWL relations which the term is in (but which aren't visible yet). In this figure, the user begins with the term they wish to view details about, expands some relations of interest, and repeats the process once more. This allows the user to iteratively expand the graph without an huge surprises. An expansion can easily be undone if the user does not like the result.

The final graph in Figure 4 displays several essential features of the `pizza:AnchoviesTopping` term, as well as some information about its place in the class hierarchy. The ontology developer would be well on her way to writing a definition using these graph contents. Using our interface, this result can be arrived at in seconds. We believe this is a much better experience than sifting through the result of heavy-handed context expansion methods.

7 ORDERING DEFINITION CREATION

As we have discussed, all non-root terms in an ontology should be defined. In addition, when approaching an ontology in which some or all of the non-root terms are undefined, there is an appropriate order that the terms should be defined so that the task is as easy as possible. In Aristotelian definitions, a definition relies on having an appropriate genus (generally, the class just above the term being defined). Having a definition for that genus can help building its child term definitions easier. So, it seems that a top-down approach is warranted: start with the most general terms in the domain, and work downward.

We have still found this guideline to be rather unconstrained: simply approaching an ontology and choosing any undefined term from the first level which contains undefined terms is not optimal. We propose two additional conditions which may ease definition creation.

1. If an ontology already has some defined terms, define terms with the most siblings already defined first. An Aristotelian definition requires differentia, so each term must be differentiated from its siblings. If a term has some siblings already defined, this may make defining the term easier.
2. Define a term's siblings before moving on to non-sibling terms. The relationship between an term's definition and its siblings definitions indicates that it could ease the definition process if a term's siblings are defined before continuing to non-siblings.

We have developed a formal algorithm which takes these considerations into account (see Algorithm 1), and implemented it within CSNePS. Executing the algorithm provides the ontology developer with an ordered list of terms. The developer then may begin at the top of the list in writing definitions, and work their way down. This may be accessed in CSNePS by typing

```
(onto-tools/definition-order)
```

[6] We have chosen to use this ontology since the topic widely accessible, and the ontology doesn't make heavy use of IUIs which can make some portions of the ontology more confusing.

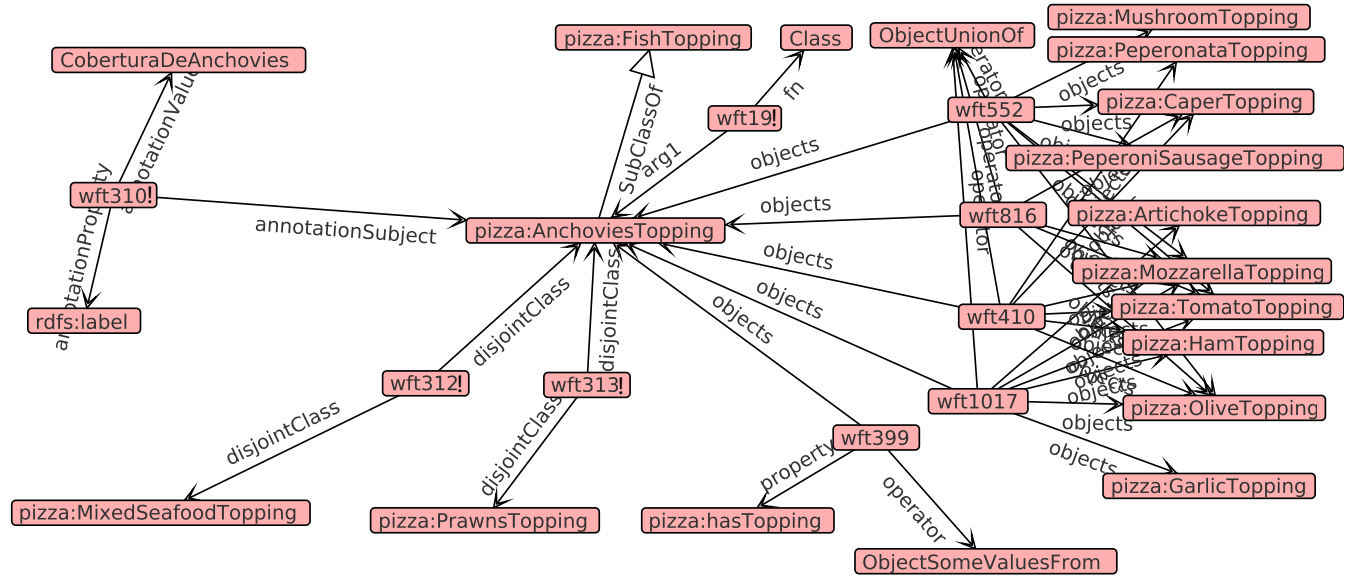

Fig. 2. The node `pizza:AnchoviesTopping` with one degree of context.

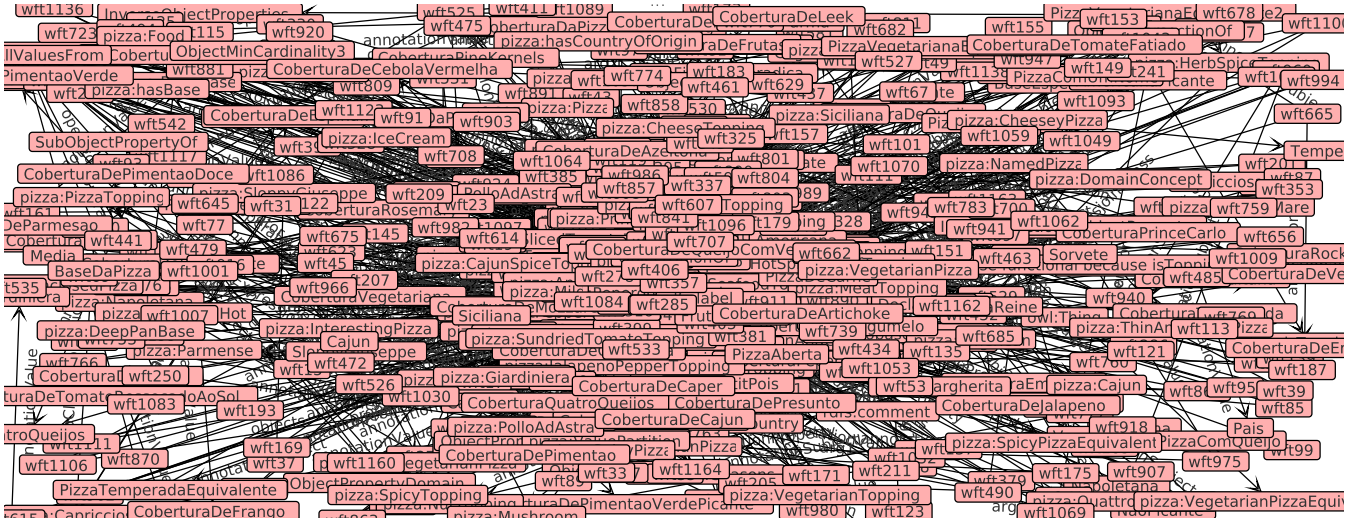

Fig. 3. The node `pizza:AnchoviesTopping` with two degrees of context.

at the console or GUI REPL. By default it looks for a property named `definition` or with the label `definition` on each term to determine if that term is defined. The user may also execute

```
(onto-tools/define-next-term)
```

to clear the graph area and show the term node for the next term to be defined.

8 DISCUSSION

In the methods section we defined some criteria for tools to help write definitions: enough, but not too much, context must be visible, and a method to order definition writing would be helpful. The method for graph expansion allows the user to see as much or as little context as they deem fit. The disadvantage here is that the user must make more actions to get to a populated visualization than context-only methods. But, we have found that the expansion can be done very quickly, and that using just context results in more time wasted sifting through the excess data in the graph. Judging by the number of irrelevant nodes the user is confronted with, the method developed here is a clear winner. The algorithm we have developed for ordering definition writing also match up with the criteria given at the beginning of the paper.

We believe these tools can have a significant impact on the ontology community. Reviewing existing ontologies using these tools can easily show the number of undefined terms, and push developers toward improving the quality of their ontology. In addition these tools can help developers create high quality definitions, by

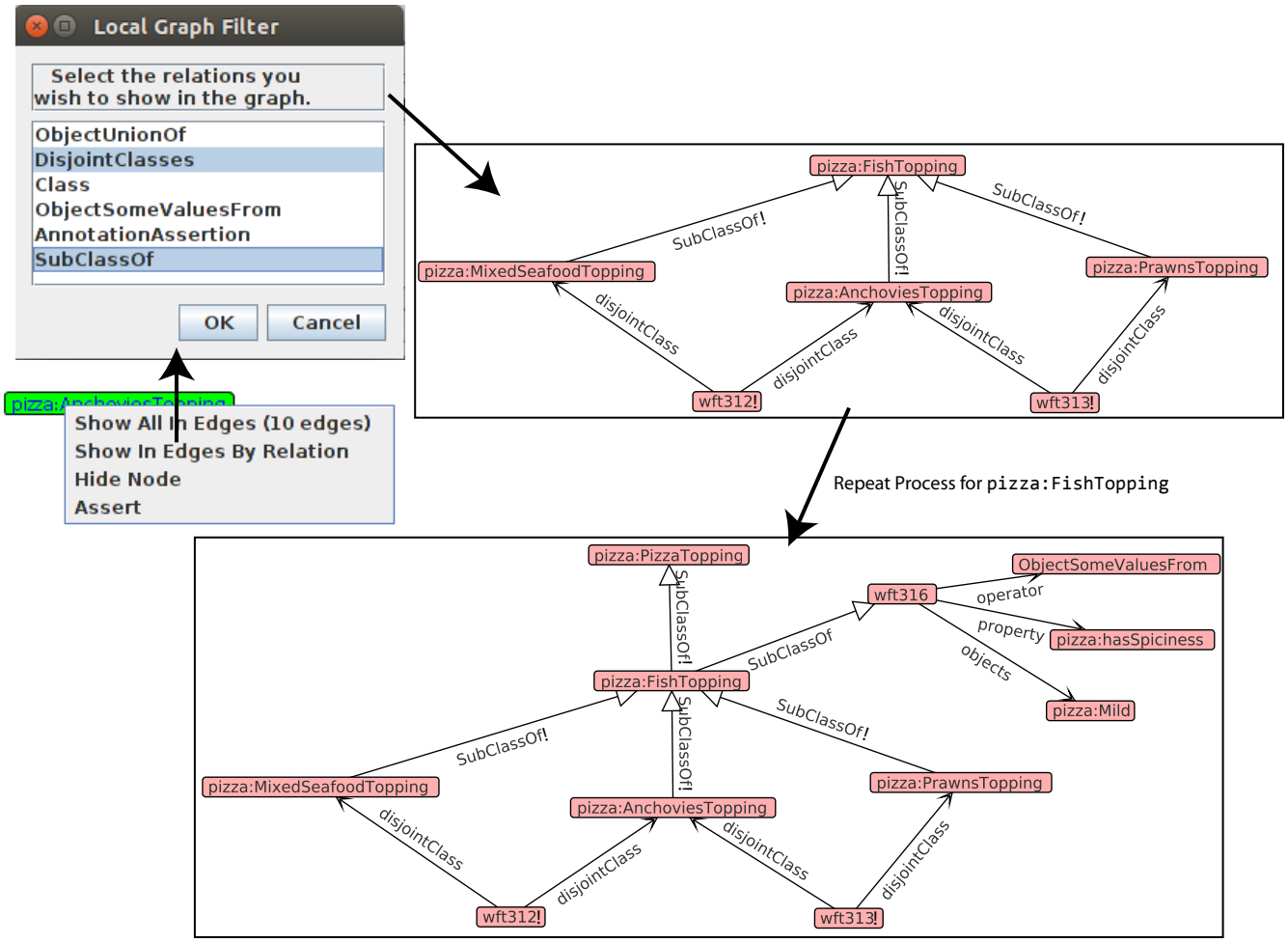

Fig. 4. An example exploration of the graph around the term `pizza:AnchoviesTopping`.

Algorithm 1 Algorithm for ordering terms to be defined.

function DEFINITIONORDER($termSet$, $rootSet$)
  $nonRootTerms \leftarrow$ DIFFERENCE($termSet$, $rootSet$)
  $termDistanceMap \leftarrow$ new Map
  for all term t in $nonRootTerms$ **do**
    if t not defined **then**
      $termDistanceMap[t] =$ DISTTOROOT(t, $rootSet$)
    end if
  end for
  $defnOrder =$ new List
  for i from 1 to MAXDISTANCE($termDistanceMap$) **do**
    $termsD \leftarrow$ TERMSWITHDIST(i, $termDistanceMap$)
    Sort $termsD$ by number of defined siblings, decreasing, while grouping siblings together.
    $defnOrder \leftarrow defnOrder + termsD$
  end for
  return $defnOrder$
end function

encoding some of the best practice guidelines (such as working top-down) and facilitating others (such as finding essential features).

In the future we intend to modify these tools so that they may be used with whatever back-end representation the user wishes, whether it be CSNePS, a SPARQL-based triplestore, or a full-fledged graph database such as Neo4j (Neo Technology, Inc., 2015). In the meantime, we have created an OWL to CSNePS converter using the OWL API (Horridge and Bechhofer, 2011) and an adaptation of the OWL Functional-style syntax.[7] Further enhancement of these tools could include integration with natural language processing techniques to check existing definitions for errors.[8] It could be rather simple to detect circularity in definitions. Using ontology-based understanding techniques (as is done in clinical notes using, for example, the iNLP system (Elkin *et al.*, 2010)) it may also be possible to detect issues such as using different kinds of differentia in sister terms.

[7] CSNePS is available at: `http://github.com/SNePS/CSNePS`, the OWL to CSNePS converter is available at: `http://github.com/digitalneoplasm/owl-csneps-converter`

[8] As one reviewer kindly suggested.

9 CONCLUSION

Writing definitions for terms in an ontology is difficult and labor intensive. Iterative graph expansion based on particular relationships has been shown to result in less cluttered graphs than simple context expansion. We believe this can help designers produce higher quality definitions more quickly. Computer algorithms to indicate to designers which terms should be defined in which order can help make defining terms more efficient. We hope that these tools will be used and developed further by the community, as giving terms good definitions is one step toward making ontologies more useful.

ACKNOWLEDGEMENTS

The authors appreciate Selja Seppälä's help with background research for this paper.

REFERENCES

Arp, R., Smith, B., and Spear, A. D. (2015). *Building Ontologies with Basic Formal Ontology*. Cambridge, MA: MIT Press. Forthcoming.

Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., *et al.* (2009). *OWL 2 web ontology language: Structural specification and functional-style syntax*. W3C Recommendation.

Dietze, H., Berardini, T. Z., Foulger, R. E., Hill, D. P., Lomax, J., Osumi-Sutherland, D., Roncaglia, P., and Mungall, C. J. (2014). Termgenie–a web-application for pattern-based ontology class generation. *Journal of Biomedical Semantics*, **5**(1), 48.

Elkin, P. L. (2012). *Terminology and terminological systems*. Springer Science & Business Media.

Elkin, P. L., Trusko, B. E., Koppel, R., Speroff, T., Mohrer, D., Sakji, S., Gurewitz, I., Tuttle, M., and Brown, S. H. (2010). Secondary use of clinical data. *Stud Health Technol Inform*, **155**, 14–29.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semantic Web*, **2**(1), 11–21.

Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., and Wroe, C. (2009). *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.2*. The University of Manchester.

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., and Giannopoulou, E. (2007). Ontology visualization methodsa survey. *ACM Computing Surveys (CSUR)*, **39**(4), 10.

Katifori, A., Vassilakis, C., Lepouras, G., Torou, E., and Halatsis, C. (2014). Visualization method effectiveness in ontology-based information retrieval tasks involving entity evolution. In *Semantic and Social Media Adaptation and Personalization (SMAP), 2014 9th International Workshop on*, pages 14–19. IEEE.

Neo Technology, Inc. (2015). Neo4j, the world's leading graph database. `http://neo4j.com/`.

Schlegel, D. R. (2015). *Concurrent Inference Graphs*. Ph.D. thesis, State University of New York at Buffalo.

Schlegel, D. R. and Shapiro, S. C. (2012). Visually interacting with a knowledge base using frames, logic, and propositional graphs. In M. Croitoru, S. Rudolph, N. Wilson, J. Howse, and O. Corby, editors, *Graph Structures for Knowledge Representation and Reasoning, Lecture Notes in Artificial Intelligence 7205*, pages 188–207. Springer-Verlag, Berlin.

Seppälä, S., Schreiber, Y., and Ruttenberg, A. (2014). Textual and logical definitions in ontologies. In *Proceedings of the Second Annual International Workshop on Ontology Definitions (IWOOD 2014)*, pages 35–41.

Shapiro, S. C. and Rapaport, W. J. (1992). The SNePS family. *Computers & Mathematics with Applications*, **23**(2–5), 243–275.

Silva, I. d., Santucci, G., and Freitas, C. d. S. (2012). Ontology Visualization: One Size Does Not Fit All. In K. Matkovic and G. Santucci, editors, *EuroVA 2012: International Workshop on Visual Analytics*. The Eurographics Association.

Stanford Center for Biomedical Informatics Research (2015). The Protégé ontology editor and knowledge acquisition system. `http://protege.stanford.edu/`.