Demonstration

OAPT: A Tool for Ontology Analysis and Partitioning

Alsayed Algergawy[1] *, Samira Babalou[2], Friederike Klan[1], Birgitta König-Ries[1]

[1]Institute of Computer Science
Friedrich Schiller University of Jena, Germany
firstname.lastname@uni-jena.de

[2] Department of Computer Engineering
University of Science and Culture, Iran
s.Babaloo@son.ir

ABSTRACT

Ontologies are the backbone of the Semantic Web and facilitate sharing, integration, and discovery of data. However, the number of existing ontologies is vastly growing, which makes it is problematic for software developers to decide which ontology is suitable for their application. Furthermore, often, only a small part of the ontology will be relevant for a certain application. In other cases, ontologies are so large, that they have to be split up in more manageable chunks to work with them. To this end, in this demo, we present *OAPT*, an ontology analysis and partitioning tool. First, before a candidate input ontology is partitioned, *OAPT* analyzes it to determine, if this ontology is worth to be considered using a predefined set of criteria that quantify the semantic richness of the ontology. Once the ontology is investigated, we apply a seeding-based partitioning algorithm to partition it into a set of modules. Through the demonstration of *OAPT* we introduce the tool's capabilities and highlight its effectiveness and usability.

Categories and Subject Descriptors

H.4 [**Information Systems**]: WWW, web applications; H.4 [**Information Systems Applications**]: Data mining

Keywords

Semantic Web, ontology, modularization, analysis

1. INTRODUCTION

Ontologies are the backbone of the Semantic Web. By making information understandable for machines [7] they enable integrating, searching, and sharing of information on the Web. The growing value of ontologies has resulted in the development of a large number of these. According to [3], at least 7000 ontologies exist on the Semantic Web, providing an unprecedented set of resources for developers of semantic applications. On the other hand, this large number of available ontologies makes it hard for software engineers to decide which ontology(ies) is (are) suitable for their needs. Even, if a developer settled on an ontology (or a set of ontologies), she is most often interested in a subset of concepts of the entire ontology, only. For example, the CHEBI ontology [1], contains 46,477 fully annotated concepts describing chemical entities of which not all will be relevant to a specific application. Also, it might be necessary to split up large ontologies like CHEBI in more manageable chunks before feeding them to ontology matching tools or other applications.

To cope with these challenges, in this demo paper, we present *OAPT*, a tool for analyzing and partitioning ontologies. The tool allows the user to interactively investigate a candidate input ontology based on a predefined set of quality criteria. This will help to build trust for sharing and reusing ontologies. Once an ontology has been analyzed, the partitioning algorithm can be applied to partition the ontology into a set of disjoint modules. Our method to examine the ontology quality is based on the consistency and richness of the input ontology. First, a suitable reasoner is applied to the ontology to validate its consistency. It is clear that the way an ontology is engineered is largely based on the domain for which it is designed and modeled. Therefore, a measure for the semantic richness of an ontology should consider different aspects and its potential for knowledge representation [9]. To this end, we then consider a set of structural, semantic, and syntactic metrics. The structural and syntactic criteria can be used to quantify the ontology design and its potential for knowledge representation, while the semantic-based criterion can be used to evaluate how instances are placed within the ontology.

To partition the analyzed ontology into a set of disjoint modules, we introduce a seeding-based clustering approach, called *SeeCOnt*. In particular, input ontologies are parsed and represented as concept graphs. A *Ranker* function is then used to rank ontology concepts exploiting the concept graph features. The highest ranked concepts are finally selected as cluster seeds (cluster heads). Each of these constitutes the initial concept of a resulting module. To assign the remaining concepts to their proper modules, we introduce a membership function. This reduces the complexity of the comparisons by comparing concepts with only seeds instead of all other concepts. Please note that this partitioning method is independent of the concrete application or a concrete subset of concepts a user is interested in. Rather, it

*Department of Computer Engineering, Tanta University, Egypt

[1]https://www.ebi.ac.uk/chebi/

relies on intrinsic ontology characteristics only. This allows, e.g., precomputation of the modules.

The rest of the paper is organized as follows. In Section 2 we present an overview of the proposed system, while in Section 3 we describe our demonstration scenario. Due to space restrictions, in this paper we provide only a glimpse of the techniques employed by *OAPT*. However, we refer to our full research paper [1] for algorithmic details and for more elements of related work.

2. THE TOOL OVERVIEW

First, input ontologies are parsed using the Apache Jena[2] framework. Then, a concept graph is extracted. We define the *concept graph* $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{L})$ as a labeled directed graph. $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ is a finite set of nodes representing the concepts of the ontology. $\mathcal{R} = \{r_1, r_2, ..., r_m\}$ denotes a finite set of directed edges representing various relationships between concepts in an ontology $\mathcal{O}$, such that $r_k \in \mathcal{R}$ represents a directed relation between two adjacent concepts $c_i, c_j \in \mathcal{C}$. $\mathcal{L}$ is a finite set of labels of graph nodes and edges defining the properties of each entity, such as the names of concepts. $n(= |\mathcal{C}|)$ and m are the number of nodes (concepts) and edges (relationships) in $\mathcal{G}$, respectively.

After an input ontology has been parsed and represented as a concept graph, the *OAPT* tool analyzes the ontology based on a predefined set of criteria. If the user is satisfied with the result of the analysis, the ontology is then partitioned into a set of modules, as shown in Fig. 1.

2.1 Ontology analysis

In order to build trust for an ontology as a prerequisite for reuse and sharing, we evaluate and analyze the quality of ontologies. We start the analysis process by applying an OWL reasoner to make sure that the input ontology is consistent. As it is known that the way an ontology is engineered is specific to the domain for which it has been designed and modeled, the ontology design and its potential to represent knowledge should be examined. To this end, we then use the ontology richness as a metric for its quality. The richness of the ontology considers different aspects of ontologies and their potential for knowledge representation [9]. We categorize measures for the richness of an ontology into three categories: *structural*, *semantic*, and *syntactic*. We exemplary present some implemented metrics for each type in detail:

- **Structural richness.** This dimension describes the topology of the concept hierarchy of an ontology. It includes several criteria, such as relationship, attribute, depth, and connection richness. The **Relationship richness, RR,** reflects the variability in the types of relations and the placement of these relations within the ontology. An ontology that contains numerous relation types other than class-subclass relations is richer than a taxonomy with just class-subclass relations. The relation richness (RR) of an ontology $\mathcal{O}$ can be defined as: $RR(\mathcal{O}) = \frac{|R \setminus SC|}{|R|}$, where $|R|$ is the number of relationships in the ontology, and $|SC|$ is the number of sub-class relations. The value of the relation richness criterion is normalized between 0 and 1, where the value of 0 means that the ontology contains only sub-class relationships. Another criterion that can be used to evaluate the structural richness of an ontology is the **connection richness, ConnR**. It indicates the number of connected components of the concept graph, i.e., the number of subgraphs linked to its root element. So, for calculating $ConnR$ we determine the number of root classes, i.e., the children of the root node. $ConnR(\mathcal{O}) = \frac{1}{No\_root\_classes}$. This metric can help to avoid "islands" forming in a knowledge base as a result of extracting data from separate sources that do not have common knowledge.

- **Semantic & Syntactic richness.** These two dimensions describe the semantics and the descriptive information of the ontology. In this context, we make use of several metrics, such as class richness and readability [9]. The **Class Richness, CR,** is an instance-based criterion used to reflect how instances in an ontology are distributed across classes. The class richness (CR) criterion can be defined as follows: $CR(\mathcal{O}) = \frac{|C^I|}{|C|}$ where $|C^I|$ is the number of classes that have instances. Another criterion that is important during the evaluation of the semantic richness of an ontology is the **descriptivity richness, DR**. This measure indicates availability of human-readable knowledge provided by an ontology. The descriptivity of an ontology can be defined as the number of concepts that have comments and/or labels: $DR(\mathcal{O}) = \frac{|C'|}{|C|}$ where $|C'|$ is the number of concepts having comments and/or labels.

We combine these three dimensions to compute the total richness of an ontology using a simple weighted-sum approach. Therefore, the ontology richness (OR) criterion is defined as follows:

$$OR(\mathcal{O}) = w_1 \times StrR(\mathcal{O}) + w_2 \times SemR(\mathcal{O}) + w_3 \times SynR(\mathcal{O}) \quad (1)$$

where $StrR(\mathcal{O})$, $SemR(\mathcal{O})$, and $SynR(\mathcal{O})$ are the total structural, semantic, and syntactic richness of the ontology $(\mathcal{O})$, respectively. w_1, w_2, w_3 are weights that reflect the importance of each of the richness metrics, such that $\sum w_i = 1$. The normalized score is then listed for the user to decide whether to partition the ontology or to look for another one.

2.2 Ontology partitioning

Once the input ontology is inspected, the next step is to partition the concepts $\mathcal{C}$ of the concept graph $\mathcal{G}$ into a set of separate (disjoint) modules $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k$ such that the *cohesion* of concepts within modules is high, while the *coupling* across modules is low. To this end, we develop a seeding-based clustering algorithm. The steps of the algorithms are described in the following:

2.2.1 Determining the proper number of modules

Typically, the number of modules a given ontology should be split up in is determined by trial and error without using objective criteria [6]. In contrast, the *OAPT* tool provides two options to determine the number of modules. First, if the user has enough experience with the ontology to be partitioned, she can directly input the number of modules. Otherwise, the user asks the tool to suggest an "optimal" number of modules.

2.2.2 Ranking the concepts

[2]https://jena.apache.org/

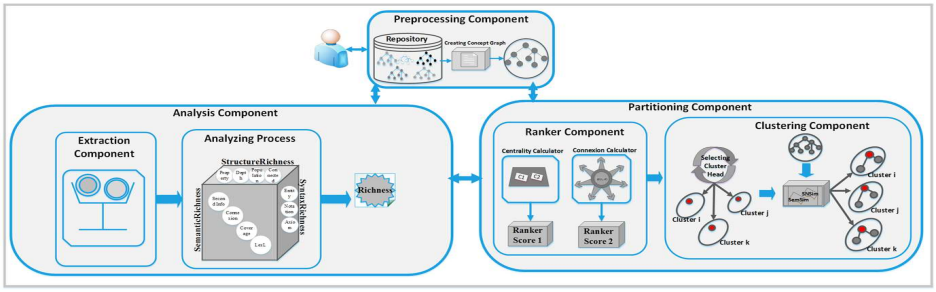


Figure 1: Tool overview

The seeding-based algorithm starts by selecting a set of nodes distinguished as *important nodes*. These nodes are then selected to be cluster heads, $\mathcal{CH}$. To quantify the role a node has within the concept graph, we introduce a new function, called *Ranker*. This function should be as simple as possible but effective. I.e. computing the *Ranker* function should not consume much time, however, correctly rank the concepts inside an ontology.

Ranker Function.

The importance of a node in a concept graph is determined by the properties of the node itself and its surroundings [5, 10]. This leads us to use graph-theoretic measures based on graph connections in the *Ranker* function. In the current software, we included two different implementations of the *Ranker* function. The first is based on the *centrality* measure of a concept, while the second depends on the context of the concept. To consider the effect of the concept itself through its edges, we use a set of centrality measures [4]. During the employment of the first *Ranker* function, we observed that it is an effective measure but requires a lot of time to rank concepts. This makes it unsuitable for partitioning large ontologies. Therefore, we propose another ranking function, which is based on the *connexion set* concept. The *connexion set* $\Psi(c_i, d)$ of a concept $c_i \in \mathcal{C}$ is defined as: $\Psi(c_i, d) = \{SubClass(c_i, d) \cup SuperClass(c_i, d)\}$, where $\Psi(c_i, d)$ is the set of all concepts within d levels that effect on c_i. $SubClass(c_i, d)$ is the set of children of c_i within d hierarchical levels, and $SuperClass(c_i, d)$ is the set of parents of c_i within d hierarchical levels. It is evident that the importance of a concept increases as it has a larger number of surrounding nodes.

2.2.3 *Determining cluster heads.*

Once having computed the importance of the concepts of a concept graph, the next step is to select which concepts represent cluster heads, $\mathcal{CH}$. If simply the nodes with the highest score are selected as the cluster heads, the distribution of cluster heads across the concept graph would be disregarded. To avoid this problem, the distance between two cluster heads is measured, and among the highest scored nodes, those with a minimum distance $\mathcal{D}$ from each other are selected as the cluster heads.

2.2.4 *Finalizing Clustering*

The seeding-based algorithm creates one module per cluster head. Then, it places direct children in the corresponding cluster and finally, for the remaining nodes, a membership function is used to determine the cluster each node shall be assigned to. The direct placement of children reduces the time complexity, since it reduces the number of comparisons by avoiding to compute the membership function for all concepts.

Membership Function.

Once having determined the cluster heads, ($\mathcal{CH}$), and having assigned direct children to their proper heads, the next step is to place the remaining concepts into the fitting cluster. To this end, we developed a membership function, *MemFun*. First, each concept is associated with a flag, $\mathcal{F}$, such that if $\mathcal{F}$ of the concept c is false, it means c is not assigned to any cluster yet and thus, the membership function has to be called for the concept c. In addition, the $\mathcal{F}$ flag can be set only once, i.e. each node can be placed in only one cluster so that there is no overlap between clusters. The membership function determines for each concept $c_i \in \mathcal{C}$ to which module $\mathcal{M}_i, i < \mathcal{K}$ it shall be assigned. For this, the similarity of c_i with all $\mathcal{CH}_i$ is calculated and then c_i is placed in a cluster with the maximum similarity value. Using the proposed membership function, each concept is compared with the Cluster Heads only, instead of comparing it to all concepts as usually done (see e.g. [2, 8]). This reduces the complexity of the algorithm.

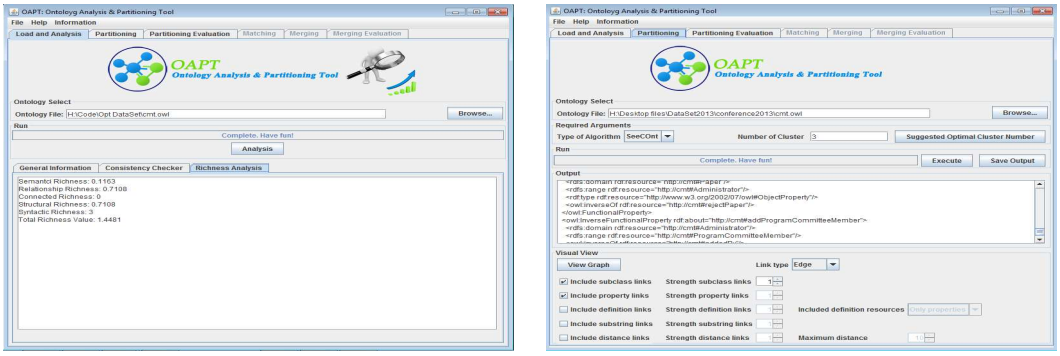
In order to compute the membership of a concept to a cluster head, a linear combination of structural and semantic similarity measures is calculated as follows:

$$MemFun(c_i, \mathcal{CH}_k) = \alpha \times SNSim(c_i, \mathcal{CH}_k) + (1 - \alpha) \times SemSim(c_i, \mathcal{CH}_k) \quad (2)$$

where α is a constant between 0 and 1 that reflects the importance of each similarity measure, $ShareNeighbors(SNSim)$ and semantic similarity $SemSim$ are two similarity measures that quantify the structural properties of the concept c_i, respectively. The $SNSim$ measure considers the number of shared neighbors of c_i and $\mathcal{CH}_k$. The neighbors of a concept are the concept's direct children, the concept's parents, the concept's siblings and the concept itself. While the $SemSim$ measure considers the semantic connection between a concept and a cluster head, which is based on the concept hierarchy.

2.2.5 *Partitioning analysis*

After having partitioned an ontology into a set of modules, the $OAPT$ tool analyzes the quality of the partitioning output. This step aims at monitoring the quality of the partitioning process. We implemented a set of quality metrics such as size, cohesion, coupling, and connectivity. The module size is used to check the number of classes and number



(a) Ontology analysis (b) Ontology partitioning

Figure 2: OAPT Screenshots

of relations within the module to validate if it is adequate.

3. DEMONSTRATION SCENARIO

In this demonstration, we will start by presenting the different features of the *OAPT* tool[3] such as the process of analyzing an ontology (Figure 2a), partitioning the ontology, or sharing the partitioning quality (Figure 2b). The demonstration will consist of two main parts. First, we would like the user to appreciate the importance of the analysis phase. Second, we present the core of the *OAPT* tool.

Ontology analysis.

In this part, we aim at demonstrating the importance of the ontology analysis phase and how it largely affects the following steps. The user can select an ontology from the given repository and then start to study the effect of different evaluation criteria. The user starts with applying a single evaluation criterion and studies its effect on the semantic richness of an ontology. Then, the user attempts to combine different sets of the evaluation criteria to see why this set of metrics is needed.

Ontology partitioning.

In this part, we demonstrate the various steps of the ontology partitioning component. First, we allow the user to validate the importance of determining an optimal value for the number of modules. She starts with guessing a value for the number of modules an ontology should be partitioned into, and then she asks the tool to suggest such a number. Once the number of modules that an ontology should be partitioned into has been determined, the user can apply the *SeeCOnt* algorithm to get these modules. The set of the output modules as well as a set of evaluation metrics will be shown to the user to validate the quality of the partitioning algorithm.

4. CONCLUSIONS

In this demo, we show how *OAPT* can be used to investigate ontologies in a way that enables knowledge engineers to determine the quality of an ontology. Once an ontology is investigated, the tool can partition it into a set of disjoint modules. We developed and implemented a new seeding-based clustering approach. The tool has been evaluated and validated with ontologies from different domains which demonstrates the effectiveness and the usability of *OAPT*. This will be demonstrated in the sample workload that we prepare for the demo. In our future work, we willo complete the tool in order to support the developers in selecting which module(s) fulfill(s) his requirements. Furthermore, we plan to improve the ontology analysis phase by considering more measures and criteria and to improve also the partitioning phase by taking into account other partitioning techniques. Furthermore, we plan to visualize all these processes and steps to be more user-interactive.

[3]http://fusion.cs.uni-jena.de/fusion/activity/oapt/

5. ACKNOWLEDGMENTS

This work is partly funded by DFG in the INFRA1 project of CRC 1067 AquaDiva.

6. REFERENCES

[1] A. Algergawy, S. Babalou, M. J. Kargar, and S. H. Davarpanah. SeeCOnt: A new seeding-based clustering approach for ontology matching. In *ADBIS*, pages 245–258, 2015.

[2] A. Algergawy, S. Massmann, and E. Rahm. A clustering-based approach for large-scale ontology matching. In *ADBIS*, pages 415–428. 2011.

[3] M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing knowledge on the semantic web with watson. In *5th International Workshop on Evaluation of Ontologies and Ontology-based Tools*, pages 1–10, 2007.

[4] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 1997.

[5] A. Graves, S. Adali, and J. Hendler. A method to rank nodes in an rdf graph. In *ISWC*, 2008.

[6] G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, pages 281–288, 2003.

[7] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems Journal*, 16:30–37, 2001.

[8] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, 67:140–160, 2008.

[9] S. Tartir and I. B. Arpinar. Ontology evaluation and ranking using OntoQA. In *ICSC*, pages 185–192, 2007.

[10] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis. RDF digest: Efficient summarization of RDF/S kbs. In *ESWC*, pages 119–134, 2015.