AAAI Technical Report SS-12-04
Intelligent Web Services Meet Social Computing

LexOnt: A Semi-Automatic Ontology Creation Tool for Programmable Web

Knarig Arabshian
Bell Labs, Alcatel-Lucent
Murray Hill, NJ

Peter Danielsen
Bell Labs, Alcatel-Lucent
Naperville, IL

Sadia Afroz
Drexel University
Philadelphia, PA

Abstract

We propose LexOnt, a semi-automatic ontology creation tool for a high-level service classification ontology. LexOnt uses the Programmable Web directory as the corpus, although it can evolve to use other corpora as well. The main contribution of LexOnt is its novel algorithm which generates and ranks frequent terms and significant phrases within a PW category by comparing them to external domain knowledge such as Wikipedia, Wordnet and the current state of the ontology. First it matches terms to the Wikipedia page description of the category and ranks them higher, since these indicate domain descriptive words. Synonymous words from Wordnet are then matched and ranked. In a semi-automated process, the user chooses the terms it wants to add to the ontology and indicates the properties to assign these values to and the ontology is automatically generated. In the next iteration, terms within the current state of the ontology are compared to terms in the other categories and automatic property assignments are made for these API instances as well.

Introduction

In recent years, the availability of services on the World Wide Web has surged. In order to make good use of these services, both human and software consumers require knowledge about existing services. Thus, the need for automatic service discovery and composition is critical. A current technological trend is creating web service mashups. The concept of a mashup is to allow users to create their own content from different types of sources such as websites, RSS Feeds[1], or Flickr[2]. A user is able to filter tailored information on a personal page to view and share with others. It is not necessary for a user to know how to create websites, but can do so simply by bringing different components together via a simplified user interface. For example, the Google Maps API[3] is often used in conjunction with location-based web services.

Currently, there are a number of web services in various domains, such as social media or mapping services that offer their APIs to be used in mashup applications. Programmable Web[4] is one such directory that offers a listing of Web service APIs. Programmable Web classifies APIs in a flat categorization where each API is manually classified within a single service category. Search is limited to attributes such as protocol or messaging type and is not related to semantic attributes of the service category. Thus, service discovery and composition within the ProgrammableWeb directory is a challenging process, since it requires considerable manual effort to locate services, understand their capabilities and compose mashup applications. Furthermore, every site has its databases modeled in a specific way, causing semantically equivalent properties to be defined differently, since data is not easily shared across different domains in the Internet.

We enhance the service descriptions by using an ontology to describe each service category. The main benefits in using an ontology to describe services are that it provides structure and semantics to service metadata which allows it to be easily classified and shared. With automated classification, service discovery becomes seamless even as more services are added or the ontology changes. Also, services can be discovered according to specific attributes, which is useful when composing services in mashup applications. Furthermore, service metadata can be shared across different domains. If a service description for a "Travel" or "Mapping" service is stored in a generic ontology repository, a mashup application can easily compose these types of services by finding its description within a repository. Thus, with an ontology description, an API can be automatically classified and queried by its attributes resulting in automated service discovery and composition.

One of the difficulties in using ontologies for describing a service domain is in creating a generic classification that describes the service class and its high-level properties. Currently, most of the semi-automated ontology generation tools either have an established taxonomy or a structured corpus, such as in the fields of biology and medicine which aid domain experts in creating ontologies. Furthermore, the

[1]RSS Feeds: http://www.rssboard.org/rss-specification (accessed Jan 24, 2012)

[2]Flickr: http://www.flickr.com (accessed Jan 24, 2012)

[3]Google Maps API: http://code.google.com/apis/maps/index.html (accessed Jan 24, 2012)

[4]Programmable Web: http://www.programmableweb.com (accessed Jan 24, 2012)

tools that analyze generic text to semi-automatically create an ontology, create taxonomic hierarchies but do not indicate property descriptions of the classes. We discuss this in the related works section. As we have indicated above, there is a need for high-level classification ontologies of different service classes or domains and their high-level properties but the current state of the art does not accomplish this. Thus, in order to make the use of ontologies for web service descriptions prevalent, a tool is required that will allow anyone to create a high-level ontology description of a domain given a corpus of data.

We propose LexOnt, a semi-automatic ontology creation tool that suggests high-level property terms for a given service class which distinguish it from the rest of the categories. Although we have tested this tool using the PW service directory, we anticipate that LexOnt can be applied to all types of unstructured text corpora describing a certain domain. The main contribution of LexOnt is its novel algorithm which takes as its input unstructured text of a given domain and generates and ranks frequent terms and significant phrases within a PW category as candidate object property assignments by comparing them to external domain knowledge within Wikipedia[5], Wordnet[6] and the current state of the ontology. LexOnt can thus be used to semi-automatically create ontologies even if the ontology engineer is not necessarily an expert of a certain domain, since the terms and phrases it suggests to the user are those that are common within the corpus and the domain.

LexOnt accomplishes this by initially generating a list of terms and phrases obtained by well-known NLP algorithms such as TF-IDF (Salton and Buckley 1988) and significant phrase generation. It then matches these terms to the Wikipedia page description of the category and Wordnet synonyms and ranks them higher, since these indicate domain descriptive words. In a semi-automated process, the user chooses the terms it wants to add to the ontology and indicates the properties to assign these values to and LexOnt generates the appropriate ontology entities. In the next iteration, terms within the current state of the ontology are compared to terms in the other categories and automatic property assignments are made for these API instances as well. We describe the details of LexOnt's algorithm, implementation and results in this paper.

Related Work

Most of the related work closest to our work involves semi-automated ontology creation for taxonomic hierarchies or domains that already have some kind of structural description. Machine learning and NLP techniques are used on text corpora alongside an already existing ontological description of the domain to generate an ontology for that specific dataset. There is work that uses clustering for semi-automatic construction of ontologies from parsed text corpora (Bisson, Nedellec, and Canamero 2000), (Reinberger and Spyns 2004), for creating taxonomic hierarchies (P. Cimiano 2004) and for generating topic ontologies (B. Fortuna 2007). The work closest to ours are those that involve finding property relationships between concepts or use external knowledge bases. A few systems which do either have been proposed: TextToOnto (Maedche and S. Staab 2000), OntoLT (Buitelaar and Olejnik 2004), and Ontolearn (Navigli and P. Velardi 2003).

Text2Onto creates an ontology from annotated texts. This system incorporates probabilistic ontology models (POMs). It shows a user different models ranked according to the certainty ranking and does linguistic preprocessing of the data. It also finds properties that distinguish a class from another. The main difference with LexOnt is that Text2Onto uses an annotated corpus for term generation. OntoLT allows a user to define mapping rules which provides a precondition language that annotates the corpus. Preconditions are implemented using XPATH expressions and consist of terms and functions. According to the preconditions that are satisfied, candidate classes and properties are generated. Again, OntoLT uses pre-defined rules to find these relationships. OntoLearn, like LexOnt, uses an unstructured corpus and external knowledge of natural language definitions and synonyms to generate terms. However, the ontology that is generated is a hierarchical classification and does not involve property assertions.

Thus, the main difference between LexOnt and these systems is that they either start off with some kind of annotated description in addition to the text corpora or they produce a pure hierarchical ontological description and exclude high-level property descriptions within the service domain. LexOnt, on the other hand, starts with unstructured text of a given service domain and uses external knowledge from online sources such as Wikipedia and Wordnet to generate terms. It then uses the terms from the constructed ontology to influence the generation of future terms. Also, in our work, we are concentrating mainly on describing service domains. Although our techniques may be applied to all types of domains, it works best for describing high-level service domains where generic properties are shared across the single domain.

Programmable Web Overview

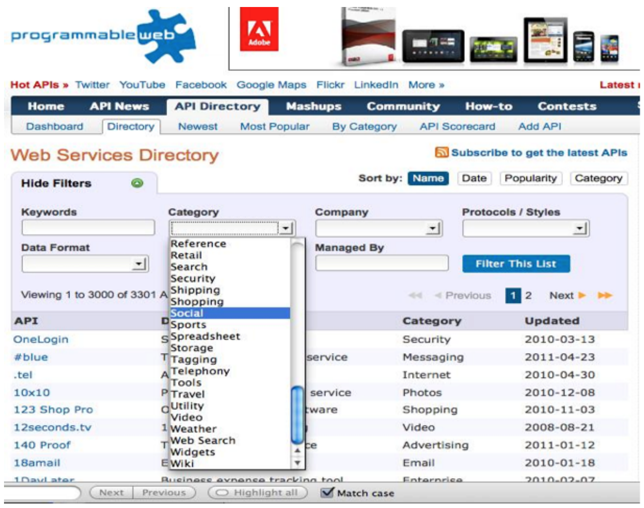
Currently ProgrammableWeb maintains a collection of over 5000 APIs divided into 56 high-level categories. The API categories are service classes such as: Advertising, Answers, Blog Search, Blogging, Bookmarks, Calendar, Chat, Database, Dating, Dictionary, Email, Photo, Real Estate, Social, Travel, Utility, Video, Weather, etc. A PW user registers its service by submitting a registration form. The form contains, among other things, URLs for the provider and sample links.. After reviewing the API, the PW team manually assigns it to a single category.

PW generates two HTML pages, given an API registration: provider.html is the front page of the provider's URL and sampleUrl.html is an HTML page published in the PW website which has specific information for the API itself. Throughout the paper, we denote the textual corpus as the "API description". The API descriptions is natural language text without any fixed structure, since we used the text data stripped from the HTML pages of an API.

[5]Wikipedia: http://www.wikipedia.org (accessed Jan 24, 2012)

[6]Wordnet: http://wordnet.princeton.edu (accessed Jan 24, 2012)

In order to search the directory, a user can either enter tags or search via a form. The form provides a drop-down menu to choose the category and properties such as keywords, data format, company, protocols. These properties, although helpful in somewhat filtering out APIs, do not really help in searching for a service based on its semantic description. Figure 1 gives an overview of the PW search site. The search results are usually a long list of links to possible API matches and a brief description of each API. If the user has a specific service in mind or wants to create a mashup of a few services, he will have to browse through the API site to understand what the API is about before he even begins to read through the documentation to start coding. Thus, merely the discovery process can be quite lengthy.



Viewing 1 to 3000 of 3301 A

API	D	Category	Updated
OneLogin	S	Security	2010-03-13
#blue	T service	Messaging	2011-04-23
.tel	A	Internet	2010-04-30
10x10	P service	Photos	2010-12-08
123 Shop Pro	C tware	Shopping	2010-11-03
12seconds.tv	1	Video	2008-08-21
140 Proof	T ce	Advertising	2011-01-12
18amail	E	Email	2010-01-18
1DayLater	Business expense tracking tool	Enterprise	2010-02-07

Figure 1: Programmable Web Directory Search

LexOnt: A Semi-automatic Ontology Creation Tool for Programmable Web

Motivation

The goal of LexOnt is to find high-level properties of a class that distinguish it from other classes. Specifically, we are using the LexOnt tool to determine generic properties of service classes in the Programmable Web directory such that they can be categorized and searched for automatically using an ontology classification. The service classification can be likened to the profile section of an OWL-S (OWL for Services) (owl) ontology where a service's semantics is described in a high-level ontology. For future work, we will look at describing the full service functionality of the PW services.

To illustrate, take the Travel and Social service categories. A typical web service that offers travel information may have features to book flights, hotels, or cars. Other travel sites may provide tourism advice or aggregate travel data and compare prices. Social networking services typically have user profiles, friend lists and some kind of sharing attribute. There are many social networking services available on the Internet today and these are distinguished by what is being shared: photos, videos, tags, reviews, etc. Thus, the goal of LexOnt is to enable a user to distinguish common terms within a category such that these terms can be applied as distinguishing features of a service domain. Once these terms are chosen, they are added as ”hasFeature” properties to the ontology.

It is unrealistic to assume that an ontology can be created in a purely automated fashion. Ontologies are often subjective descriptions of a given domain which require human evaluation. However, we can semi-automate this process by analyzing a corpus of data within a domain and aiding a user who may not be completely familiar with a domain to choose terms that may describe this domain, and then automating attribute creation. Furthermore, as the ontology is being created for a domain, the information within it can also be used to give higher weights for the terms within the corpus that match the terms that have been instantiated in the ontology.

Algorithms

The novelty in LexOnt's algorithm is twofold: 1) use information from HTML text describing service APIs, Wikipedia articles describing that service domain and Wordnet for synonymous terms as our corpus for generating a top-N list of words and phrases that may be used toward assigning distinguishing service features 2) semi-automate the construction of the ontology by labeling terms that have been assigned manually to the ontology based on the term generation and then creating or asserting property assignments within the ontology for subsequent iterations. Thus, we incorporate the current state of the ontology into the corpus of data itself to assign higher weights or labels to those terms which are already assigned in the ontology and then regenerate a list of terms from the text data given the current ontological terms.

Term Extraction with TF-IDF and Significant Phrases Initially, LexOnt uses well-established NLP techniques to extract terms from the API text descriptions that may be used as distinguishing features of the service. Two lists are generated: a top N list of TF-IDF ranked terms and a list of two-termed significant phrases.

TF-IDF: Text frequency- inverse document frequency (TF-IDF) score of a word in a corpus shows how important the word is in the corpus. Importance of a word in a particular document depends on how frequently the word has been used in the document and how common the word is in all the documents in the corpus.

Significant Phrases: A signficant phrase consists of two or more words. We chose to generate a list of these phrases, in addition to single term TF-IDF ranked words, because we determined in our tests that some of these phrases gave a very good indication of high-level property descriptions. For example, in the Advertising service class, significant terms generated were things like: “mobile advertising,” “facebook advertising” or “campaign advertising.” This immediately gave a synopsis of what the API was about. Significant phrase generation is a two-phase process. First, collocations, terms that appear together, are determined. Then, from this list, unique collocations are filtered out.

The Chi-square test calculates the significance of the collocated words. It measures how often the words in a phrase are seen together and how often they are seen alone. For example, if in a description the word "social" appears 8 times, "stream" appears 8 times and "social stream" appears 8 times, then "social stream" is considered as a significant phrase as there is a high correlation of these words to be appeared together as a phrase. To calculate chi-square probability of an n length phrase, a n-by-n table is constructed. The χ^2 sums the differences between observed and expected values in all squares of the table.

Once the collocations are determined, we filter this list and find the unique phrases to determine the distinct properties. This helps to prune out irrelevant word collocations such as: "sign in," "api usage," "log out" that appear in almost all the API descriptions and are not important for our purpose. We perform the following steps to find distinctive phrases of an API:

1. Create testing sets and training sets. The testing set is generated via the API being processed. The training sets, on the other hand, are generated using all the APIs that do not have the same category as the API being processed. For example, if the API being processed belongs to the Advertising category, choose APIs which are not from the Advertising category.
2. Find frequencies of n-grams in the training set.
3. Find frequencies of n-grams in the testing set.
4. N-grams in the testing set are sorted according to their significant score. The significant score is the z score for binomial distribution[7].

Use of external knowledge-base: Wikipedia, Wordnet and Constructed Ontology Given the categories of an API, external resources, like Wikipedia, Wordnet and the constructed ontology can be used to leverage the underlying concepts and properties of an API. For example, top 20 words from the Advertising Wikipedia page are: *advertising, marketing, brand, television, semiotics, advertisement, billboard, radio, product, bowl, sponsor, consumer, advertise, placement, super, logo, commercial, infomercial, message, promotion*. LexOnt uses these words to rank the top terms. For each of these terms, it then looks for synonymous terms that may be related to it. It uses Wordnet to find synonyms of the top-N words that are being used. Finally, it searches through the ontology to see if there are matching terms and if there are, labels or ranks these terms to indicate that they have already been created. Below, we illustrate the step-by-step algorithm for using the external knowledgebase. Table 1 also shows an example of generated terms for one of the APIs in the Advertising category.

1. Extract Wikipedia page for each category.

[7]z-score for binomial distribution: http://alias-i.com/lingpipe/docs/api/com/aliasi/stats/BinomialDistribution.html (accessed Jan 24, 2012)

Table 1: Example of property selection: 140 Proof

Top N TF-IDF from Wiki	Advertising, marketing, brand, television, semiotics, advertisement, billboard, radio, product, bowl, sponsor, consumer, advertise, placement, super, logo, commercial, infomercial
Top N TF-IDF from Wordnet	ad, advertisement, advertizement, advertising, advertizing, advert, promotion, direct_mail, preview, advertorial, mailer, newspaper_ad, commercial, circular, teaser, top_billing
Top N TF-IDF from PW Category	proof, Persona, Stream, Replies, Authors, Say, hello, Ad, Brands, social, Consumers, Advertisers, Audience, Ads
Top N TF-IDF Ranked based on external kb	Advertisers (wiki), Consumers (wiki), Social (wiki), Brands (wiki), Ads (related), Ad (related), proof, Persona, Stream, Replies, Authors, Say, Hello, Audience
Top N Significant Phrases ranked based on external kb	stream advertising (wiki), social stream (wiki), say hello, author, replies, google groups, ober, michaels, proof, erik michaels, persona targeting

2. Find top words based on TF-IDF ranking.
3. If a word or phrase in the API contains any of the top Wikipedia words, rank it higher than the others.
4. Find synonymous or related terms to the list of generated terms using Wordnet
5. If a word or phrase in the API contains any of the related terms, rank and label these
6. If any of the generated terms lexically match terms in the ontology, label these as well.

Semi-automatic ontology creation process LexOnt is implemented as a Protege plugin to provide a user-friendly interface during the interactive process of creating and editing an ontology. Thus, the user installs the LexOnt tab in the Protege editor and can seamlessly switch to the ontology editor tab within the Protege application.

The diagram in Figure 2 illustrates the process of ontology creation with LexOnt. It starts with a set of PW service categories, generates a top-N list of terms and phrases, given the initial corpus of data and creates a list of ontology classes based on the PW service categories. The user then creates a few high-level object properties. In our case, we have chosen an object property that indicates distinguishing features of a service. Thus, we have created a $Feature$ class and its corresponding $hasFeature$ property, which points to this class.

The PW classes and their APIs are displayed in a list on the side bar. When the user clicks on an API, top-N terms and phrases appear. The user can then choose relevant terms and enter it into the system as property-value assignments. We have built the interface in such a way that when the user right clicks on a term, he can either search for it in the original corpus to see how it is used or assign it as a possible property value in the ontology. For example, if the user sees that the phrase ”mobile advertising” is relevant, it can right click on the term, choose the appropriate property to assign this term to, which is $hasFeature$ and then click on the ”Find” button. This displays all the locations in the API which have this term in it. The user can also do a search for this term in all the other APIs within a category.

Matches within all the APIs are displayed in a table that can be edited by the user. Once the user chooses the terms that best describe the APIs and edits it, it clicks on the ”Create” button which automatically creates instances of APIs within that category and assigns the relevant terms chosen to the $hasFeature$ property.

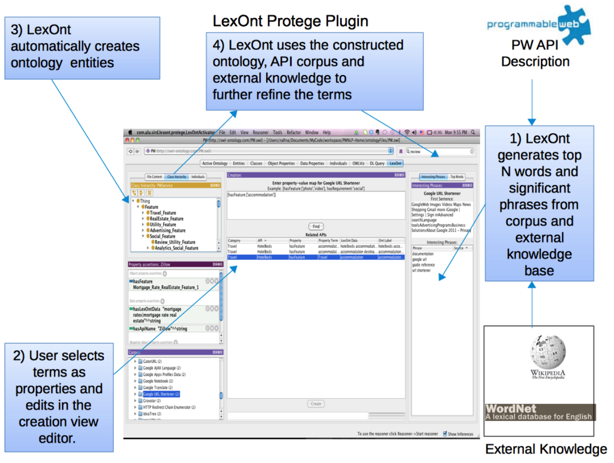

3) LexOnt automatically creates ontology entities
LexOnt Protege Plugin
4) LexOnt uses the constructed ontology, API corpus and external knowledge to further refine the terms
programmableweb
PW API Description
1) LexOnt generates top N words and significant phrases from corpus and external knowledge base
2) User selects terms as properties and edits in the creation view editor.
WordNet
A lexical database for English
Wikipedia
External Knowledge

Figure 2: Ontology Creation Process with LexOnt

Example An example of property assignments for the $140Proof$ API instance is illustrated in the snapshot in Figure 3. The figure shows that the *140Proof* API instance belongs to the *Advertising_Service* class and has features from the *Social_Stream_Advertising_Feature* and *Persona_Targeting_Advertising_Feature* classes which are both subclasses of the *Feature* class.

This example not only shows how the instances are being assigned, but how the *Advertising_Feature* class has been constructed. As terms are chosen from the corpus and assigned as properties to individuals, the *Advertising_Feature* class is constructed automatically as well. Thus, it can be seen that LexOnt has determined that the *Advertising_Service* class can have advertising features such as banner, campaign, cloud, contextual, facebook, etc.

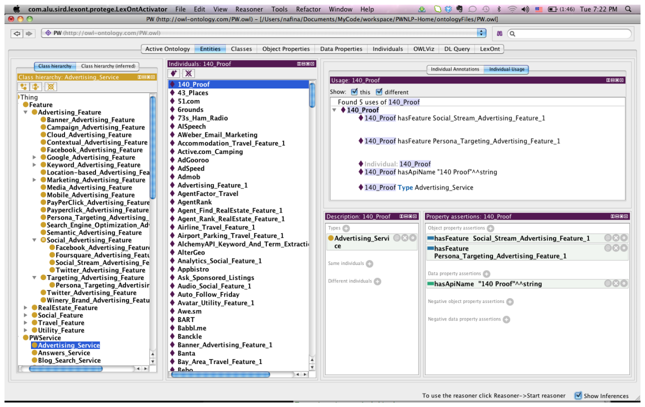

Class hierarchy: Advertising_Service
Individuals: 140_Proof
Usage: 140_Proof
Description: 140_Proof
Property assertions: 140_Proof
140_Proof hasFeature Social_Stream_Advertising_Feature_1
140_Proof hasFeature Persona_Targeting_Advertising_Feature_1
140_Proof hasApiName "140 Proof"^^string
140_Proof Type Advertising_Service

Figure 3: Ontology snapshot for the $140Proof$ API Instance

Implementation and Evaluation

LexOnt was implemented using three different Java APIs. The Lingpipe API[8] was used for the NLP algorithms to generate TF-IDF terms and Significant Phrases. The Protege plugin GUI was implemented using the Protege API (Knublauch et al. 2004). Finally, the ontology generation code was implemented using the latest version of the OWL-API (Bechhofer, Volz, and Lord 2003) which also handles the next generation of OWL 2 (Grau et al. 2008). As a future paper, we will discuss the contributions we have made in creating a user-friendly interface for semi-automatic ontology creation. Currently, only OntoLT has been implemented as a Protege plugin, but is now incompatible with the current version of Protege. Since Protege is a robust ontology editor, having LexOnt as a Protege plugin tab has significantly improved the ontology creation process.

We tested for three things when evaluating LexOnt: the precision/recall of the TF-IDF term and Significant Phrase generation; how helpful the external knowledge base was when choosing terms; and whether or not the terms were used in their exact form, similar form or different forms. The results are seen in tables 2, 3, 4 respectively.

The categories that were used to generate the terms were chosen according to specificity, number of APIs and a priori knowledge of the domain. The Advertising and Real Estate service categories each had a manageable number of APIs, on average around 40 APIs. These categories are also specifically defined domains that had comprehensive Wikipedia pages. Additionally, the co-author who created the ontology for these categories did not have much knowledge about either of these domains. We then chose two categories that the ontology creator was familiar with: Travel and Social. The former had around 75 APIs and the latter had over 200 APIs. Finally, we chose a generic Utility category which had 65 APIs and had a hodgepodge of various services with no corresponding Wikipedia page.

[8]Alias-i. 2008. LingPipe 4.1.0. http://alias-i.com/lingpipe (accessed Jan 24, 2012

Table 2: Precision/Recall Stats

Category	Sig. Phrase	TF-IDF	Recall
Advertising	3.98%	2.77%	43.88%
Real Estate	1.02%	.92%	9.57%
Social	3.21%	2.8%	20.19%
Travel	1.96%	2.4%	30.91%
Utility	9.58%	3.83%	34.91%

Table 3: Percentage of terms used from KB

Category	Sig. Phrase	TF-IDF
Advertising	41.38%	52.73%
Real Estate	100%	100%
Social	31.90%	11.38%
Travel	82.26%	72.73%
Utility	0%	0%

Table 2 showed that on average, the precision of generated terms was around 4% for Significant Phrases and 3% for TF-IDF terms. The recall for both was around 28%. Although these numbers are not very promising in and of themselves, when looking at Tabel 3 to see how these match to the external knowledge base, the results are quite promising. For the categories with external KBs, the percentage of terms matching KB terms were between 30 and 100 percent for Significant Phrases and 11 and 100 percent for TF-IDF terms. We also determined that for the categories with a smaller API set and well-defined Wikipedia pages, these numbers were quite high. Thus, it was quite feasible to quickly assess distinguishing features of a category just by looking at the generated terms and how well they matched the external KB.

For the Social category, the percentage of terms used from a KB was not as high because there was a lot of variance within this cateogry. Social APIs varied widely between what was shared. Aside from the usual friends, blogs, statuses, tags, photos and videos, there were APIs that shared other types of items such as wine lists, books, courses or fashion. The Utility category didn't have a corresponding Wikipedia page and thus we relied heavily on the TF-IDF and Significant phrase generation and finding these terms within the original corpus via the LexOnt interface in order to determine what the API features were.

Finally, we tested to see how these terms were actually assigned within the instances. Thus, we checked to see whether these matches were exact, similar or completely different. For example, if LexOnt produced a term "mobile" but the actual ontology assignment was "mobile advertising," this would count as a similar match. From our results, we saw that the percentage of equal and similar matches for individuals were quite high for all the categories. However, the percentage of different matches varied. Again, we can see that for a category that has a wide variety of data, such as in Social, the percentage of differing terms ranked higher than the others.

Table 4: Term Usage

Category	Equal Terms	Similar Terms	Different Terms
Advertising	85.71%	100%	65.71%
Real Estate	16.67%	91.67%	66.67%
Social	1.73%	86.9%	79.1%
Travel	6.25%	100%	2.3%
Utility	5%	60%	50%

Discussion

LexOnt has shown to be an effective tool for semi-automated ontology creation. From our initial results, we have determined that using an external knowledge base to filter out generated terms and phrases greatly increases the accuracy of the feature selection. It also helps in understanding the common terms within a corpus. We will continue to test LexOnt for the other categories and compare results.

Future Work and Conclusion

For future work, we would like to test LexOnt on other types of corpora besides the PW directory to see how well it works within other domains such as in medicine or biology. We also want improve the seamless integration of LexOnt's automated ontology creation mechanism with the user's interaction. For example, when a user is in the process of creating an ontology, he should be able to save the current state of the LexOnt tool and the tool should be aware of all the instances created, and store information not only about generated terms that were accepted but those that were rejected as well. This way, LexOnt will avoid false positives within the generated terms.

Additionally, once we have a complete ontology, we will build a querying interface to Programmable Web and test to see how service discovery has improved with the new ontology. Eventually, we would like to describe services not just on a high-level but in its functional details. Allowing standard descriptions of a service's inputs, outputs, pre and post conditions will aid in autmomatic service composition. Once we have the high-level discovery phase automated with this ontology, we plan on continuing this work to see how we can generate an ontology description of the service which include its functional details.

In conclusion, we have presented LexOnt, a semi-automatmic ontology generator that aids in the ontology creation of a high-level service ontology. It uses the Programmable Web directory of services, Wikipedia, Wordnet and the current state of the generated ontology to suggest relevant terms that may be incorporated within the ontology. LexOnt builds the ontology iteratively, by interacting with the user, taking in terms that the user has chosen, adding these to the ontology and ranking terms according to the external knowledge base. From our initial findings, we have determined that LexOnt is a useful tool to generate a high-level ontology description of a domain, specifically for users who are not domain experts.

References

B. Fortuna, M. Grobelnik, D. M. 2007. Ontogen: Semi-automatic ontology editor. In *Human Interface, Part II, HCII 2007*.

Bechhofer, S.; Volz, R.; and Lord, P. 2003. Cooking the semantic web with the owl api. *The Semantic Web-ISWC 2003* 659–675.

Bisson, G.; Nedellec, C.; and Canamero, L. 2000. Designing clustering methods for ontology building: The mok workbench. In *Ontology Learning Workshop, The 14th European Conference on Artificial Inteligence (ECAI)*.

Buitelaar, S., and Olejnik, D. 2004. A protege plug-in for ontology extraction from text based on linguistic analysis.

Grau, B.; Horrocks, I.; Motik, B.; Parsia, B.; Patel-Schneider, P.; and Sattler, U. 2008. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web* 6(4):309–322.

Knublauch, H.; Fergerson, R.; Noy, N.; and Musen, M. 2004. The protégé owl plugin: An open development environment for semantic web applications. *The Semantic Web–ISWC 2004* 229–243.

Maedche, A., and S. Staab, S. 2000. Semi-automatic engineering of ontologies from text. In *12th International Conference on Software Engineering and Knowledge Engineering*.

Navigli, R., and P. Velardi, A. G. 2003. Ontology learning and its application to automated terminology translation. In *IEEE Intelligent Systems, vol. 18:1*.

Owl-s (semantic markup for web services).

P. Cimiano, A. Pivk, L. S.-T. S. S. 2004. Learningtaxonomicrelations from heterogeneous evidence. In *Ontology Learning and Population Workshop The 16th European Conference on Artificial Inteligence (ECAI)*.

Reinberger, M., and Spyns, P. 2004. Discovering knowledge in texts for the learning of dogma-inspired ontologies. In *In Proceedings of the Ontology Learning and Population Workshop, The 16th European Conference on Artificial Inteligence (ECAI)*.

Salton, G., and Buckley, C. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24(5):513–523.