OntoNotes: A Unified Relational Semantic Representation

Sameer S. Pradhan
BBN Technologies
Cambridge, MA 02138

Eduard Hovy
ISI/USC
Marina Del Rey, CA 90292

Mitch Marcus
University of Pennsylvania
Philadelphia, PA 19104

Martha Palmer
University of Colorado
Boulder, CO 80309

Lance Ramshaw
BBN Technologies
Cambridge, MA 02138

Ralph Weischedel
BBN Technologies
Cambridge, MA 02138

Abstract

The OntoNotes project is creating a corpus of large-scale, accurate, and integrated annotation of multiple levels of the shallow semantic structure in text. Such rich, integrated annotation covering many levels will allow for richer, cross-level models enabling significantly better automatic semantic analysis. At the same time, it demands a robust, efficient, scalable mechanism for storing and accessing these complex inter-dependent annotations. We describe a relational database representation that captures both the inter- and intra-layer dependencies and provide details of an object-oriented API for efficient, multi-tiered access to this data.

1 Introduction

The OntoNotes project is addressing the challenge of large-scale, accurate, and integrated annotation of multiple levels of the shallow semantic structure in text. Experience has shown that when individual levels like syntactic parse structure, propositional structure, and semantic role labels can be annotated with high consistency (inter-tagger agreement, or ITA), then machine learning models can also be implemented to predict those structures with only somewhat lower accuracy. The premise of OntoNotes is that integrated annotation covering many levels will allow for richer, cross-level models enabling significantly better automatic semantic analysis.

Given that goal, the mechanism used to store and access the annotation becomes a key part of the research, and not just a tool support issue. The representation has to correctly capture the dependencies between the different annotation layers, so that consistency can be maintained across layers. The representation also has to allow for unified access, so that cross-layer features can be used in the integrated predictive models that will make use of these annotations. This paper describes how we have addressed the research challenge of modeling such multi-layer annotations, with complex, cross-layer dependencies, while providing efficient, convenient, integrated access to the data.

2 OntoNotes

The OntoNotes [6] project is multi-year, collaborative effort between BBN Technologies, the University of Colorado, the University of Pennsylvania, and the University of Southern California's Information Sciences Institute.

2.1 Data

The Year 1 release consists of newswire data from two languages – English and Chinese. The English portion is a 300k word, 597 documents collection from the non-financial news portion of the Wall Street Journal (WSJ), and the Chinese portion is a 250k word collection comprising 325 documents from the Xinhua newswire and 78 documents from the Sinorama magazine. These documents are annotated with the following layers of information:

1. **Syntax** – A syntactic layer representing a revised Penn Treebank [10, 1].
2. **Propositions** – The proposition structure of both verbs and nouns in the form of a revised PropBank [13, 1]
3. **Word Senses** – Coarse grained word senses are tagged for the most frequent polysemous verbs and nouns, in order to maximize coverage. The word sense granularity is tailored to achieve 90% inter-annotator agreement as demonstrated by Palmer et al. [12]. The related senses of different words will eventually be connected to concepts in the Omega ontology [14].
4. **Names** – The corpus was tagged with a set of 18 proper name entity types that were well-defined and well-tested for inter-annotator agreement by Weischedel and Burnstein [18].

5. **Coreference** – General anaphoric coreference that spans a rich set of entities and events—not restricted to a few types, as has been characteristic of most coreference data available until now—has been tagged with a high degree of consistency. Attributive coreference [6] is tagged separately from the more common identity coreference.

For a more detailed description of the different layers, the reader is referred to [6].

2.2 The Challenge

To the best of our knowledge, this is the first time that an attempt has been made to integrate so many different, rich, layers of syntax and semantics. Although, in combination, we had decades of experience in annotating the individual layers (excluding coarse grained word sense, and general anaphoric coreference), we had no mechanism for integrating them. This posed a significant challenge as it is more or less an open problem, and there exist no off-the-shelf resources that can be used to meet this end.

The questions that this level of integration pose are:

1. How do we ensure that all the components are consistent with each other? The types of inconsistencies that we encounter in this process are two fold: i) Ones that challenge the underlying assumptions and mechanics of the annotation differences and require a careful revision of the inter-connecting components, and ii) Ones that are more technical (engineering/formatting) in nature. Although there is no silver-bullet for solving either one, a solution to the latter can significantly alleviate problems encountered during representational manipulations.

2. How do we distribute all these different layers of data? Should we distribute them as independent pieces and leave the task of assembling them to the end user? or, do we provide an integrated representation that greatly simplifies the dissemination of this rich information? If the latter, what would be the best way to accomplish this?

3. What type of representation would best facilitate the use of this information as training data for systems that will be incorporated into applications with their own knowledge sources. Can this representation also support leveraging these additional knowledge sources during the training process?

2.3 Properties of an Ideal Solution

Since these layers represent related linguistic information, there is a high degree or interconnection between them. To begin with, the PropBank annotations are defined over nodes in the Treebank. Most[1] of the coreference links have also been defined, by design, over the nodes in the Treebank. Vital information relating to the interpretation of each word sense, including its relation to the WordNet senses, their argument structure, as well as the constraints imposed on arguments by the particular semantic frame that a predicate invokes, are captured in the sense inventory and frame files. In short, the information required to interpret the semantics of a sentence is spread over several different files, and without some mechanism to control this information, it can easily become asynchronous.

An ideal solution would be one that combines this information in an integral whole which allows an end-user to both easily interpret all the vital connections as well as to easily manipulate the information. The representation of such layers, we believe, should provide a bare-bones structure independent of the individual semantics with the following properties:

1. **Efficiency** – It should efficiently capture both intra- and inter- layer semantics

2. **Independence** – It should maintain the independence of each annotation layer. Any component should be replaceable with a parallel representation. For example, it should allow replacing PropBank with FrameNet [3]. That is, as long as the parallel representation exhibits the same core properties, it should be easily incorporable into the whole.

3. **Flexibility** – It should provide mechanisms for flexible integration. For example, it should accommodate a change in representation of, say, propositions over spans of text instead of over nodes of a given syntactic representation.

4. **Granularity** – It should be integrated at a level of granularity so as to allow relatively easy integration of more components.

5. **Robustness** – It should capture first-order, or primary connections between the components in such a way that secondary connections do not become inconsistent upon a superficial change in representations.

6. **Queriability** – It should facilitate cross-layer queries.

7. **Versioning** – It should not be limited to storing the annotations themselves, but should manage different versions and hypotheses generated by automatic systems, allowing them all to coexist.

[1] Some entities/events constitute subparts of the relatively flat NP structure in the Treebank and have to be defined over word spans instead of corresponding to nodes

3 A Solution

We have developed a representation which we believe possesses all the above qualities to a more or less satisfactory degree: a relational database representation that is used to define and store the required semantics underlying the data, and an object layer which allows for intuitive manipulation of this data.

3.1 Relational Layer

The main reason for using a relational database representation is that it allows for flexibility of design and provides an objective language to efficiently specify all the interconnections in the form of its schema. Furthermore, there exist time-tested mechanisms to deal with the problem of maintaining consistent states of each layer, and at the same time allow for concurrent modifications to individual layers. In short, this is done by defining dependencies through the use of Primary and Foreign keys over database tables. The reader can find this information discussed in depth in any standard text book on the subject of relational databases.

The entity relationship diagram for the database is shown in Figure 1. The tables are shown divided into six logical blocks depending on the type of linguistic annotation that they represent: i) The corpus itself, ii) Treebank, iii) PropBank, iv) Word Sense, v) Names and vi) Coreference. Each of the annotation schemes adds meta-information to the corpus. The lowest common granularity of annotation is represented by the Treebank tokens[2], rather than the more prevalent character-based indexing of several existing corpora, significantly simplifying the visualization and manipulability of the data. If required, the token-based indexing scheme allows for dereferencing to lower levels of atomicity without affecting the interface between the corpus and other layers of annotation. This could be achieved by the addition of a table that maps to the lower level, thus providing a comfortable degree of encapsulation. This tokenization is particularly well-suited for the existing layers of annotation as they take the Treebank tokenization as a basis of dereference. There are a few exceptions; for example, in the case of coreference for about 2% of the cases an entity is (most likely) a subpart of one of the flat-NPs that the Treebank realizes. This requires only the addition of a derivable layer of token-based offsets. In the case of names, although the original annotation is based on token-offsets, about 93% of the names actually align with the existing nodes in Treebank. In such an instance, we provide a derived layer of nodes corresponding to the named entities. In short, the database captures the primary level of granularity and allows for the flexibility of adding derived layers for better interpretation when necessary.

[2] words tagged with part of speech in the Treebank

In reviewing the design, note that all the tables with the suffix `_type` indicate tables that only store type information. The others represent instance tables where instances, instead of just the type information, are stored. Various cardinalities such as *one-to-one* or *one-to-many* relationships are shown using the *crow's feet* style of representation. The `ontonotes` table contains the id for the OntoNotes corpus. This is associated with the many subcorpora that represent it – identified by the `subcorpus` table. The subcorpus contains many files captured in the `file` table, which in turn contain one or more documents (the `document` table). The document then contains one or more sentences (the `sentence` table). This now puts a structure on the raw text which we are embellishing with layers of linguistic information. The second logical block is the Treebank. Here at the center lies the `tree` table which represents a general case of trees, nodes and leaves. The root tree has a `NULL` parent whereas the leaves have `NULL` children. Hierarchical information is captured through table recursion. The meta-information on the tree nodes is captured in the `compound_function_tag` and `function_tag_type` tables. There can be more than one function tag associated with a node in the tree, and many nodes associated with the same function tag type – known as a *many-to-many* relationship in database terms, and the design principles dictate the generation of a link table which we call the *compound*_function_tag table and which contains a composite primary key which is made up of the two foreign keys: one being the primary key of the `function_tag` table; and the other being the primary key of the `tree` table. For the sake of generality, each `tree` has an associated `token` and other syntactic information such as the part of speech type (`pos_type` table), the phrase type (`phrase_type` table), the trace chain (`syntactic_link` table). The reason for giving the latter table a more general name is that it is also used to capture links from the propositions as indicated by the reference to the *argument_id*.

The PropBank logical block captures the propositional annotation. Here, at the center lies the `proposition` table which has associated with it a `predicate` and one or more `arguments`. Both the predicate as well as the arguments exhibit a *many-to-many* relationship with the nodes in the Treebank. Therefore, we create two more link tables `predicate_node` and `argument_node`. `predicate_type` and `argument_type` tables capture the respective type information. Each predicate in the PropBank invokes a semantic *frame* and that determines which of it's core arguments such as ARG0, ARG1, etc. can be legally associated with that predicate as well as the semantics of those arguments. This frame type is represented in the `pb_sense_type` table, and the `pb_sense_type_argument_type` serves as the link table. This in turn is connected across the logical boundary to the

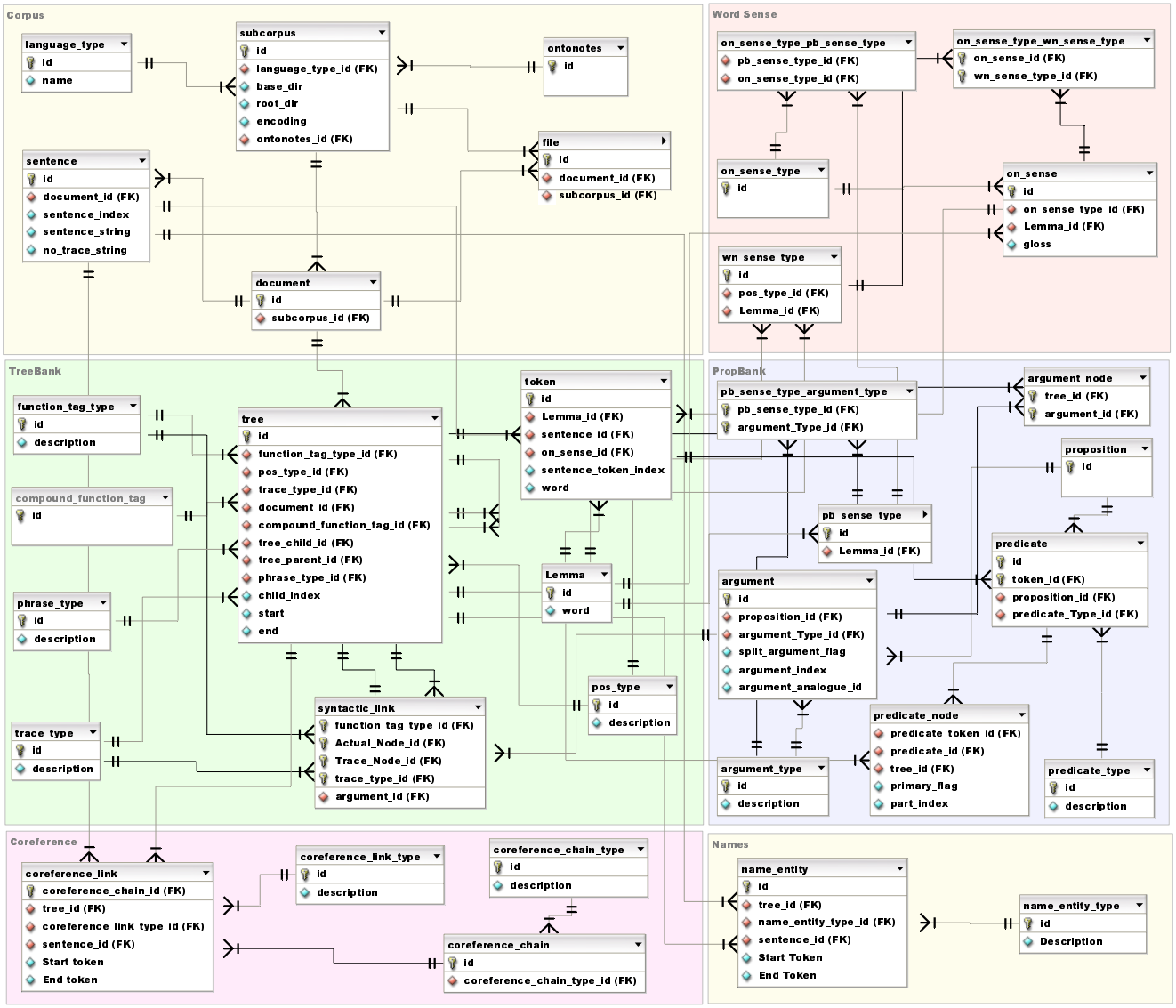

Figure 1. Entity-Relationship diagram of the OntoNotes database

`on_sense_type_pb_sense_type` table in the Word Sense portion of the database. What this means is that the OntoNotes sense type has a *many-to-many* mapping to the PropBank frame sense. A similar relationship to the WordNet senses is captured in the `wn_sense_type` and `on_sense_type_wn_sense_type` and the `wn_sense_type` tables.

Finally, names are represented with the `name_entity` and `name_entity_type` tables in the logical block called Names, and the tables `coreference_chain` and `coreference_link` capture the coreference chains and links within each document in the corpus. The `coreference_chain_type` and `coreference_link_type` store the respective type information. As mentioned earlier, most of the coreference links and names correspond to a node in the tree, and that information is stored as `tree_id` in the respective table where applicable, or NULL otherwise. We used the MySQL database to realize the database design.

3.2 Object Layer

A well-defined relational layer provides a clear foundation for designing the object layer. To facilitate a better design of the object layer, we took the following decisions:

1. We decided to trade-off database normalization for database design elegance and integration with the object world. Therefore, we did not normalize the tables beyond the first normal form.
2. We decided to create a composite primary key for

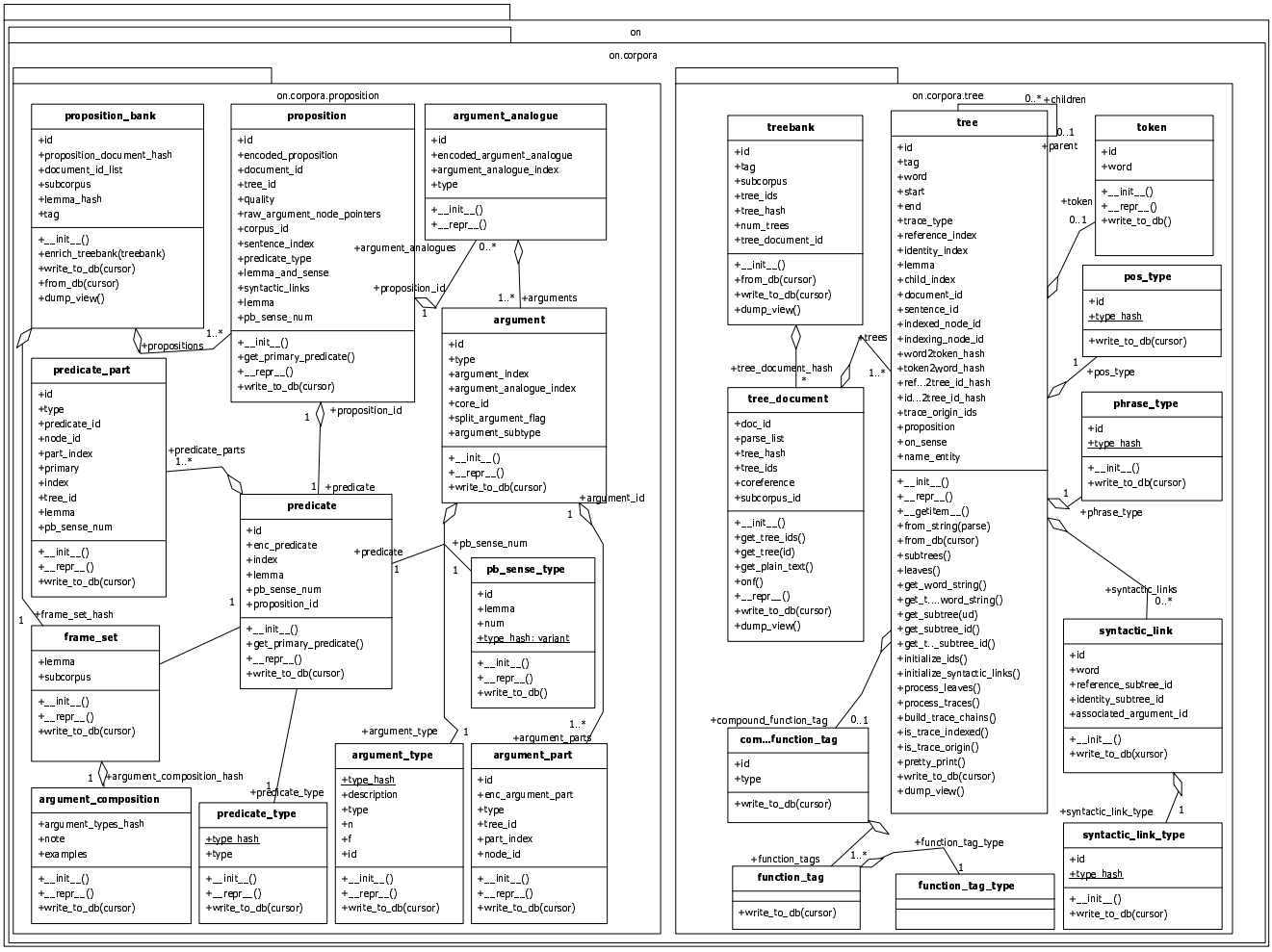

Figure 2. UML representation of the modules on.corpora.tree and on.corpora.proposition which represents the Treebank and PropBank respectively.

each table rather than using auto-increment, or using a database generated composite primary key. The goal here was seamless integration between the relational and object world. This meant that the database primary keys can also be used to index the objects in the respective containers. Another advantage of this is that the keys by themselves have a lot of semantic value and a person looking at them can, to some extent, decipher what it represents.

In the object design, almost every database table corresponds to an object with the database columns representing part of the attributes. There is almost always more derived information required for logical manipulation that is part of an object which is not in the database table itself. This resonates with the original plan of having only the bare-minimum of information needed to represent the data in the database. If required, more complex logic can be easily built in the object layer. Also, most often it is the case that the objects inside container objects are skeletal representations of those objects thus maintaining component independence.

Depending on the interface requirements of a particular application, the level of detail in each object can be altered to meet various needs. As an example, the word sense of a lemma can be tagged to be “1” but that “1” actually represents the sense “1” which has various attributes associated with it, such as its definition, the associated PropBank frame groups, the associated WordNet sense groups, etc. This is achieved by just accessing the “sense_inventory” class which centralizes information for each lemma and their senses, mappings etc. This has an alternative benefit of avoiding possible redundancy and therefore inconsistency. Figure 2 shows the UML representation of part of the package “on.corpora” which comprises, among others, the modules “on.corpora.tree” and “on.corpora.proposition” The `argument_part` class represents the information in the “ar-

gument_node" table. The `argument_analogue` does not really have a exact parallel in the database world, but it captures the fact that an argument can be represented by multiple nodes in the tree – a trace node, which points to possibly another trace node which points to the actual node that identifies the string that the argument is represented with. All three, in the PropBank world, are considered to be equivalents of the same argument. Therefore the name "argument_analogue". The attribute "argument_analogue_index" is in order to maintain the directional association between these equivalent arguments. Further, each layer is provided with application level logic to integrate itself with the other layer that it has an integral connection with. For example, in this particular case, owing to the centrality of the Treebank the `propbank` and `on_sense` classes have been provided with `enrich_treebank` methods which make the necessary connections, thus forming full-fledged objects. Whereas, `name_bank`, and `coreference_bank` have been provided with `enrich_align_treebank` methods that try to align the token spans with nodes in the tree when there exists one.

More details on the design can be found in the "ontonotes-db-tool" API that has been submitted for distribution with the corpora. The Python programming language has been used to implement the object layer.

4 Interaction Lifecycle

We will take a brief look at a typical interaction between the raw data, database, and object layers. A few lines of code allow a pre-defined corpus file-structure to be read and converted into objects. The application logic could identify several errors and inconsistencies in the raw data that are encountered during object creation. These are mostly at the independent object level. Upon successful object creation they are written to the database. Each object uses its `write_to_db(cursor)` method to write itself to the database. The top-level container contains the overall logic whereas the intermediate containers know how to write themselves to the database and these methods are delegated from top down to achieve the desired result. Next the database can identify errors and inconsistent relationships. These are written to a report file. Part of the data that does get read into the database is guaranteed to be consistent with the design. The `ontonotes-db-tool` API can then be used to read in the database, and generate the objects from it. All top level containers know how to re-create the full-fledged objects and pointers to objects as required. This is possible through the `from_db(cursor)` method.

In a typical database initialization lifecycle, the raw data will be read and written to the database iteratively until all the errors and inconsistencies are solved. When the data in the database is stable, most likely a lifecycle would involve reading the database and converting it into objects and performing the required manipulations. This manipulation can be done at three different levels – i) using generated objects only; ii) using SQL queries only, ii) combining both to achieve the simplest data manipulability.

5 Benefits of this Architecture

Lets say the user wishes to find an answer to the following question – *What is the distribution of named entities that are* ARG0*s of the predicate "say"?* The pseudocode outlined in Figure 3 shows how you could accomplish that using a combination of SQL and object-level manipulation. Without the current architecture, this manipulation would have required significant pre-processing by each end-user, requiring possible re-interpretation of the underlying semantics of the data itself, and a possible introduction of error therewith.

While synchronizing and revising the Treebank and PropBank annotations, there were several cases of well-defined changes such as: i) if an NP representing an ARGM argument, is dominated by a PP, which is dominated by a VP that is the parent of the predicate, then the ARGM label is transferred from the NP to the dominating PP; ii) if an argument tagged on an NP is dominated by an SBAR, dominated by a WH-phrase, then a separate semantic link (SLINK) is created that links the ARG label on the WH-pronoun to its referent, the latter of which was previously directly tagged as an argument of the predicate; iii) traces are inserted to address some synchronization issues; iv) argument attachments, and therefore phrase attachments were altered. More such changes are mentioned in [1]. All these changes required some degree of change to the Treebank, and therefore all the pointers in PropBank annotation had to be revised to be consistent with the Treebank. Since we did not have this tool/representation when we performed these merges, it was quite a painful and kludgy experience. There are tools such as `TSurgeon`, which do allow arbitrary manipulations of tree structures, but fail to be very useful in such cases where multiple representations are simultaneously affected, and have to be synchronized. In light of this new representation and API, those changes would be extremely easy.

We used the OntoNotes data in the Lexical Sample WSD task of the SemEval [16] competition. It was extremely easy to format the word sense information from OntoNotes to conform to the predetermined Semeval lexical-sample task format.

The amount of coreference annotation in OntoNotes was richer and more in quantity than any other effort in the past. However, the tool that we used was a generic tool called "Callisto[3]" developed at MITRE. Although it is a very flexible tool, it does not provide mechanisms to add data-level

[3] `http://callisto.mitre.org/`

Procedure: get_name_entity_distribution(a_arg_type, a_predicate_lemma)

```
Load OntoNotes database
for all proposition ∈ on.proposition.proposition_bank.propositions() do
    if proposition.predicate.lemma == a_predicate_lemma then
        (arg_type, arg_id) ← get_arg_info("select * from argument where proposition_id = proposition.id")
        for all arg_type ∈ arg_types do
            if arg_type == a_arg_type then
                node_id ← get_arg_node_id("select * from argument_node where argument_id = argument.id")
                document_id ← proposition.document_id
                sentence_id ← proposition.sentence_id
                document ← on.tree.treebank.get_tree_document(document_id)
                tree ← document.get_tree(sentence_id)
                node ← tree.get_subtree(node_id)
                name_entity_type ← node.name_entity
                name_entity_hash[name_entity_type] ← name_entity_hash[name_entity_type] + 1
            end if
        end for
    end if
end for
return name_entity_hash
```

Figure 3. Pseudocode for performing a cross-layer query on the representation using the API

consistency checks based on the semantics of the individual tasks that you use it to accomplish. One recurring manifestation of this limitation was that annotators could erroneously add a coreference link to multiple coreference chains. To correct this, we added a quality control step which comprised of trying to add the annotated documents to the database, and inserted a routine using the DB-API, that created a report of multiple-link errors, which the annotators could easily read, and edit the coreference chains accordingly to eliminate the inconsistencies. This process was repeated until no errors were identified while loading a document to the database.

Even in the well established, and heavily-used Penn Treebank, we identified some orphan, or duplicate traces that were subsequently corrected. This process identified many more such cases in the Chinese Treebank, which are relatively new and unexplored by the community – Especially since automatic parsers typically tend to ignore this information and not reproduce it.

Last, but not the least, annotation tools can be built using this representation as their backend, and tools could be easily written to visualize this complex data fairly easily.

6 Related Work

Although this is probably the first attempt at combining so many different layers together, the importance of this goal has already been recognized by the community and attempts have been made to reach a consensus; most notably by the formation of the ISO/TC 37/SC 4 standard by the International Standards Organization. This standard identifies principles and methods for creating, processing and managing language resources [8]. A working group within this standard WG1-1 has been trying to put together a Linguistic Annotation Framework (LAF) that can serve as a basis for harmonizing existing language resources. However, the bottom line is that this is a framework, and whoever wants to conform to that has to define the schemas and write translation routines to convert their data into the required representation – which is a form of feature structure graph. We envelope our specific six layers by efficiently capturing their internal semantics at a skeletal level and provide an object-level representation that could be easily translated into any format that might evolve from this standard.

Another feature structure representation – GLARF [11] tries to capture information in various Treebanks, and also tries to superimpose some predicate argument structure. This technique is more representation centric – trying to capture a union of various individual representations, however, as per our understanding, without any means of easy access to the data. It is also not clear how it would extend to accommodating more layers of semantic information.

An additional significant effort is the Unified Linguistic Annotation (ULA) project [15] which is also a collaborative effort aimed at merging several existing semantic annotation efforts: PropBank, NomBank, Coreference, the Penn Discourse Treebank, and TimeBank. Crucially, all individual annotations are being kept separate in order to make it easy to produce alternative annotations of a specific type of semantic information (e.g., word senses, or anaphora) without needing to modify the annotation at the other levels.

One of the main goals of this project is to eliminate any theoretical incompatibilities between the different layers.

In the past, database representations have been used to store individual annotation layers as in case of FrameNet [2] as well as WordNet. However, they are restricted to predicate argument structures and pure ontological representations, respectively. Neither of them have provided native layers of API for easy manipulation. The distribution is a collection of XML documents, so end-users have to write routines to read and manipulate the data themselves. Recently, as part of the Unstructured Information Management Architecture (UIMA) [7] effort, IBM has introduced a mechanism called Common Analysis System (CAS) [4] that allows definition of annotation types and serialization as well as querying capabilities. To our knowledge, it has not yet been widely used, or reported, so a detailed comparison is not possible at this time.

Two other works that comes closest to ours in terms of the types of annotations and corpora itself, are the Prague Dependency Treebank [5] and the Salsa project [9, 17]. However, even these do not include word senses, their connections to the Ontology, and a full range of coreference. Neither do they address additional languages (Chinese and Arabic) and genres (Broadcast News, Talk Shows, etc.) that we address.

7 Conclusions

In conclusion, we have created a corpus with various levels of semantic information integrated in one big database. This process identified several levels of inconsistencies that were resolved, ensuring a clean, consistent final product. The relationships between all the layers and within the layers themselves is efficiently captured in the database schema. We have also provided an object layer on top of the database layer, written in Python, which can flexibly manipulate the data at the level of the database or as objects, to extract information across layers which was not easily possible before. It can also produce the individual layers by themselves as well as a human readable representation. All this is available for distribution through the Linguistic Data Consortium (LDC). This should facilitate defining custom views of the data as well as extracting cross-layer features for use in predictive models, neither of which was easily possible previously.

8 Acknowledgments

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA/IPTO) under the GALE program, DARPA/CMO Contract No. HR0011-06-C-0022.

References

[1] O. Babko-Malaya, A. Bies, A. Taylor, S. Yi, M. Palmer, M. Marcus, S. Kulick, and L. Shen. Issues in synchronizing the English Treebank and PropBank. In *Workshop on Frontiers in Linguistically Annotated Corpora*, July 2006.

[2] C. Baker, C. Fillmore, and B. Cronin. The structure of the framenet database. *International Journal of Lexicography*, 16(3):281–296, 2003.

[3] C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet project. In *Proceedings of COLING/ACL-98*, pages 86–90, Montreal, 1998. ACL.

[4] T. Gotz and O. Suhre. Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43(3), 2004.

[5] J. Hajic, B. Vidova-Hladka, and P. Pajas. The prague dependency treebank: Annotation structure and support. In *IRCS Workshop on Linguistic Databases*, 2001.

[6] E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. OntoNotes: The 90% solution. In *Proceedings of HLT/NAACL*, USA, June 2006.

[7] IBM. Unstructured information management architecture (UIMA) – `http://www.research.ibm.com/uima`. 2005.

[8] N. Ide and L. Romary. International standard for a linguistic annotation framework. *Journal of Natural Language Engineering*, 10(3-4):211–225, 2004.

[9] K. Erk and S. Pado. A powerful and versatile xml format for representing role-semantic annotation. In *Proceedings of LREC*, 2004.

[10] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

[11] A. Meyers, G. R., M. Kosaka, and S. Zhao. Covering treebanks with glarf. In *ACL/EACL Workshop on Sharing Tools and Resources for Research and Education*, 2001.

[12] M. Palmer, O. Babko-Malaya, and H. T. Dang. Different sense granularities for different applications. In R. Porzel, editor, *2nd Workshop on Scalable Natural Language Understanding*, Boston, MA, USA.

[13] M. Palmer, D. Gildea, and P. Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

[14] A. Philpot, E. Hovy, and P. Patrick. The omega ontology. In *Proceedings of the ONTOLEX Workshop at IJCNLP*, Jeju Island, South Korea, October 2005.

[15] J. Pustejovsky, A. Meyers, M. Palmer, and M. Poesio. Merging PropBank, NomBank, TimeBank, Penn Discourse Treebank and coreference. In *Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*,

[16] SemEval 2007. SemEval-2007. In *4th International Workshop on Semantic Evaluations*, 2007.

[17] D. Spohr, A. Burchardt, S. Pado, A. Frank, and U. Heid. Inducing a computational lexicon from a corpus with syntactic and semantic annotation. In *Proceedings of IWCS-7*, 2007.

[18] R. Weischedel and A. Brunstein. BBN pronoun coreference and entity type corpus LDC catalog no.: LDC2005T33. BBN Technologies, 2005.