Enhance Entity Extraction Using an RDF Store

Jans Aasman,
Franz Inc., 2201 Broadway, Suite 715
Oakland, CA 94612, USA
www.franz.com
ja@franz.com

Rita Joseph
Expert System, 1464 Waggaman Circle
McLean, VA 22101, USA
www.expertsystem.net
rjoseph@expertsystem.net

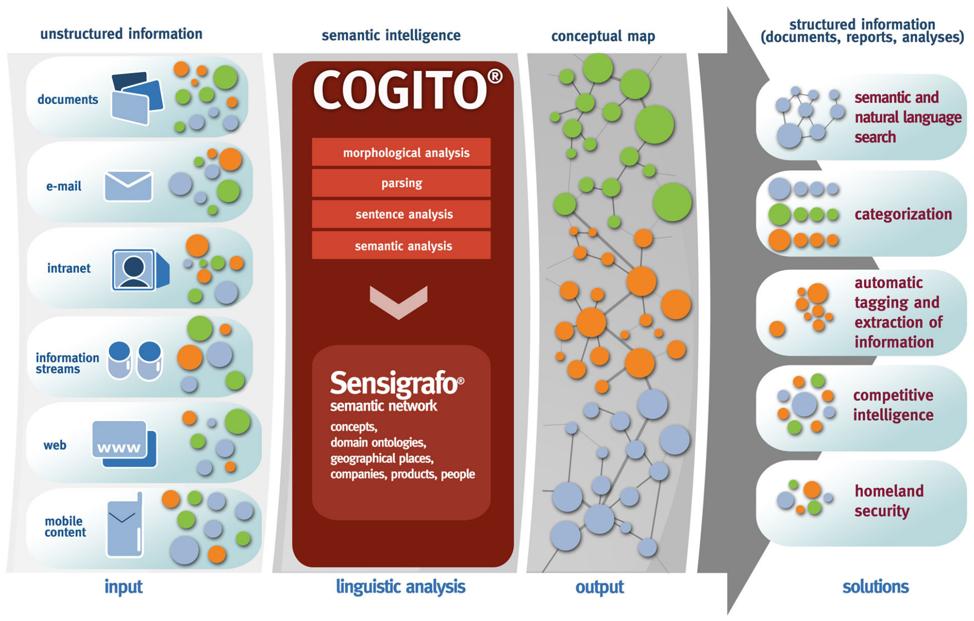
Abstract. Using robust entity extraction technology allows for the automatic comprehension of words, sentences, paragraphs and whole documents. Categorization, extraction, domain establishment, taxonomy and ontology creation can then be built upon a semantic technology infra-structure to exploit the capabilities provided in OWL and RDF. This technology brings value to the intelligence community by allowing users to find, discover and create structured knowledge connections from what were previously unstructured information sets. The presentation “Enhance Entity Extraction Using an RDF Store” will demonstrate how to represent output from Expert System’s Cogito entity extractor in a Franz AllegroGraph RDF database, and perform reasoning along with visually generated SPARQL queries. We will demonstrate a new product called TexTriples that combines professional entity extraction, a scalable triple-store, and intelligent web spidering. We will show how TexTriples collects thousands of articles from newspapers and blogs, and processes them through Cogito to create RDF triples for use in AllegroGraph. The presentation will discuss tips and techniques in dealing with these representations, demonstrate how to relate entities to Linked Data such as DBpedia, Scalability, and finally we will perform a number of queries on the resulting triple store data using some straight forward inferencing.

Keywords: entity extraction, RDF, OWL, taxonomy, ontology, SPARQL, Semantic Web

1 Entity Extraction: Cogito’s Approach

Cogito collects all the structural and lexical text aspects in order to comprehend natural language and understand the meanings of words and sentences. Cogito’s processing results in a cognitive and conceptual map, i.e. a structured representation of qualifying aspects of incoming unstructured data. The output structuring allows the automatic processing of the most relevant elements of the text. (Figure 1)

Figure 1:



1.1 Cogito's Main Features

Cogito is composed of various modules dedicated to specific activities needed to disambiguate texts and process natural language which are essential for automatic comprehension of questions formulated in everyday language.

To automatically understand text, we need:

- a **semantic network**, which is the heart of semantic technology;
- a **parser** to trace each text back to its basic elements;
- a **linguistic engine** to query the semantic network (this links the basic elements of the text with the semantic network of meanings);
- a **"disambiguation" system**.

1.1.1 Semantic Network

We've been developing a set of semantic networks graphically connected to express conceptual representations of language.

A semantic network is a lexical database in which terms are entered and grouped based on their meanings, or the concepts they express. Therefore, they are not ordered alphabetically like in a standard dictionary but according to their meaning (this is why the network is called "semantic") and to the various possible connections among these meanings (and this is why we talk about "semantic relations".)

Each node in the semantic network is linked to the others by semantic connections in a hierarchical and hereditary structure, in the form of a graph.

Our network contains:

- information about connections among objects
- specifications about the lexical domains of each word
- information about the frequency of use

The richness of a semantic network is measured by both the quantity of words/concepts and of the semantic relationships. For example, concepts can be linked to each other in the following ways:

1) Subnomen (hyponymy: relation between a specific concept and a more general one) and supernomen (hypenymy. The supernomen is the more generic term, a word having a general meaning in comparison to others representing specifications of that meaning: ex. "animal" is a supernomen of "cat");

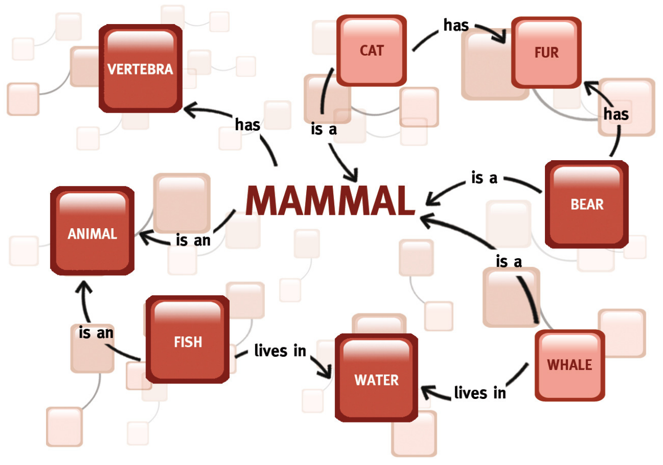
2) Parsnomen (meronymy) and omninomen (holonymy), or the part-whole semantic relationship. A parsnomen is a noun that indicates a part of a whole (which is called the omninomen), such as for example the case of piece-cake (part=portion-whole=object) or plastic-bottle (part = material - whole = object);

3) Relationships among nouns and verbs such as verb-subject or verb-object: given a noun and considering all possible "verb/subject" links, we obtain all the verbs normally (frequently) connected to that noun when it is the subject of the sentence: subject noun "food" verbs "to rot", "to grow", etc. The mechanism is the same when we consider the semantic relation "verb-object": object noun "food" verb "to eat", "to swallow", "to grind", "to chew"…

4) Other kinds of connections, such as geographical links, are based on similar logic

- each geographical element (not only countries, towns, rivers, valleys, etc. but also for example monuments) is connected to other geographical elements. Therefore for example "St. Cloud" is linked to "Minnesota" which in turn is linked to "Midwest" which is linked to "USA". Also, for example "Piccadilly" is linked to "London" which is linked to "England" which is linked to "Great Britain", etc. (Figure 2)

Figure 2:



1.1.2 Parser

The first step to take in order to understand the meaning of a sentence is to determine the grammatical role of each word. For example, in these sentences:

(a) “He **aged** 40 years.”
(b) “The wine has **aged**.”

The word “aged” appears with two different grammar types: in sentence (a) the word is an adjective, while in sentence (b) it's a verb. According to traditional technology the two words are the same, while the semantic technologies assign different meaning to them. Recognizing a word independently of its written form is equally important; nouns and verbs have several forms:

(a) “Marcello Mastroianni was the most popular Italian **actor** abroad.”
(b) “It is dissapointing that most of our popular young **actresses** do not play a protagonist role.”

In the sentences above, two forms (“actor”, “actresses”) expressing the same concept are used. The parser performs a complete morphological, grammatical and syntactical analysis of the sentence, managing more than 3500 rules very quickly. Our parser uses an innovative and ad hoc methodology to query the semantic network, resulting in a significant improvement of the existing parsing. So semantic technology individuates gender - to recognize both words in the sentences above as forms of “acting person” associating all of them to their common meanings correctly, instead of treating different words, individually, as other systems do.

1.1.3 System of Disambiguation

For humans, meaning is something obvious to us because of our capability to refer automatically to cultural elements that help us to understand the meaning of a word. The disambiguator of meanings, included in our semantic technology, thoroughly analyzes sentences or whole documents and distinguishes the right meaning for each element found, eliminating possible ambiguities.

The information of all the possible meanings of words is fundamental in order to process textual content with high precision.

The disambiguation of meaning is one of the most complicated problems of semantics. To obtain a satisfying elaboration speed, the following comoponents are needed:
- a vast knowledge structured like an encyclopedia
- a set of disambiguation algorithms working perfectly

Disambiguating, in fact, is the true problem in the automatic interpretation of text. In order to distinguish between
“The rust eats the tower.”
“The knight eats the tower.”

The research and development of automatic systems for semantic disambiguation must solve a crucial problem: the administration of the number of existing combinations that can be generated when dealing with words and texts. These can be combined together in a very high number of ways, increasing exponentially.

A disambiguation system can work sentence by sentence or can consider whole documents, according to the way it is configured. Distinguishing all the possible meanings of a text is an additional, but extremely critical, step beyond the more common analyses of: logical, grammatical, query of the semantic network, and domain analysis.

There are many examples of interpretations of words that humans can take for granted but a program can not, including expressions meant in a figurative sense.

Some examples of what semantic disambiguator can do include:
Understanding univocally the following sentences:
Example 1:
“*He **has eaten** a chicken.”*
*“The sweater **was eaten** by the moths.”*
*“The rust **ate** the tower.”*
*“The slot machine **ate** his money in just one summer.”*

"Your car ***eats*** *too much oil."*

Example 2:
"We went out for a ***row****."*
"The condemned is in a death ***row****."*
"They've had a big ***row****."*
"My ***row*** *boat is the third in the* ***row****."*

2 AllegroGraph Database

AllegroGraph is an RDF graph database and application framework for building Semantic Web applications. It can store data and meta-data as triples/quads; query these statements through various query APIs like SPARQL and Prolog; and apply reasoning with its built-in RDFS++ reasoner. Additional features of AllegroGraph include Free-Text Indexing, Range Queries, Federation, Social Network Analysis, Geospatial capabilities and Temporal reasoning.

The data represented back in Figure 2, Cogito Semantic Network represents information about mammals and animals. Like much of the data on the web, there are explicit relations like *mammal is an animal* and implicit or common-sense relations such as *petOf is an inverse relation to hasPet* and *Cat is a subClassOf Mammal.*

Though there are many ways to store this information, the W3C has standardized on the Resource Description Framework (RDF). RDF breaks knowledge into *assertions* of *subject predicate object* (like the three sentences above). For obvious reasons, these assertions are called *triples*. If we have many triples from different contexts, we can append an additional slot to each assertion; we call this slot a *named graph*. Even though these assertions are now quads, we'll still call them triples.

AllegroGraph is a high-performance database built to hold this information, query it, and reason with it. One thing to note is that AllegroGraph doesn't restrict the contents of its triples to pure RDF. In fact, we can represent any graph data-structure by treating its nodes as subjects and objects, its edges as predicates and creating a triple for every edge. The named-graph slot can be used to hold additional, application-specific, information. Used this way, AllegroGraph becomes a powerful graph-oriented database.

For many applications, graph databases can be more flexible and faster than RDBMSs because:

- You add new predicates without changing any schema
- One-to-many relations are directly encoded without the indirection of tables
- You never think about what to index because *everything* is indexed

In RDF, an assertion is a statement that contains subject, predicate, and object (in the context of graph). The bulk of an AllegroGraph triple-store is composed of **assertions**. Though called **triples** for historical reasons, each assertion has five fields: subject (s), predicate (p), object (o), graph (g), and triple-id (i)

All of s, p, o, and g are strings of arbitrary size. Of course, it would be very inefficient to store all of the duplicated strings directly so we associate a special number (called a Unique Part Identifier or UPI) with each unique string. The **string dictionary** manages these strings and UPIs and prevents duplication. To speed queries, AllegroGraph creates **indices** which contain the assertions plus additional information. AllegroGraph can also perform freetext searching in the assertions using its **freetext indices**, and keeps track of **deleted triples**. Triple-data generally comes into AllegroGraph as strings either from pure RDF/XML or as the more verbose but simpler N-Triple format. The programmer API also makes it easy to import data from RDBMSs, CSV or any other custom data format.

AllegroGraph has the ability to encode values directly into its triples (thus bypassing the string dictionary completely). This allows for both more efficient data retrieval and extremely efficient range queries. We take advantage of this data representation in the add-on libraries for geospatial reasoning, temporal reasoning and social network analysis.

2.1 Triple-store operations

You can manipulate data in triple-stores via many different interfaces and languages including Java, HTTP, Python, Perl, C#, Ruby and Lisp. Each language provides mechanisms to create and open triple-stores; load them with data in bulk-mode or programmatically; manage indices; enable RDFS++ reasoning; query for triples that match simple or complex constraints; serialize triples in many formats; and understand and manage server performance.

2.1.1 Range Queries
In addition to strings, AllegroGraph can store many datatypes directly in its triples. This lets it perform range queries in a single operation. A range query involves immediate data lookup and comparison and is therefore as fast as a search for an individual triple.

2.1.2 Cursors
When AllegroGraph is given a query pattern, it responds with a cursor that iterates over the triples that match the pattern. Programs can use functions like Java's cursorNext() to move through a cursor or use higher-level constructs like map-cursor.

2.2 Querying Indices

AllegroGraph builds indices so that any query can find its first match in a single I/O operation. We can abbreviate the index flavors using s for subject, p for predicate and so on. What matters with an index is the sort order of the triples. For example, the spogi index first sorts on subject, then predicate, object, graph, and finally, id. If we ignore the triple ID, there are 24 different index flavors running from spogi through gopsi. Out of the box, AllegroGraph builds six index flavors in the background as triples are added. You can customize which indices are built, when they are built and how they are updated. For example, if you never use named-graphs then you can drop the three g indices to save both disk space and processing time.

2.3 Query APIs

2.3.1 SPARQL
SPARQL is the query language of choice for modern triple-stores. AllegroGraph's SPARQL adheres to the W3C standard; includes a query optimizer; and has full support for named-graphs.

2.3.2 RDFS++ Reasoning
Description logic or OWL reasoners are good at handling (complex) ontologies, they are usually complete (give all the possible answers to a query) but have completely unpredictable execution times when the number of individuals increases beyond millions.

AllegroGraph's RDFS++ reasoning supports all the RDFS predicates and some of OWL's. With RDFS++ we exchange the completeness of full OWL for predictable and fast performance

2.3.3 Prolog
Prolog is an alternative query mechanism for AllegroGraph that provides the ability to specify queries declaratively. You send Prolog select queries to the server as a string and you get the bindings back as a list of values.

2.4 Federation - Data Management

AllegroGraph uses the same programming API to connect to local triple-stores (either on-disk or in-memory), remote-triple-stores and federated triple-stores. A federated store collects multiple triple-stores *of any kind* into a single virtual store that can be manipulated as if it were a simple local-store. Since federation provides a natural mechanism to join disparate triple-stores, we can use separate instances of AllegroGraph to load data on multiple CPUs and then combine them at query time.

AllegroGraph's federation mechanism and flexible triple-store architecture combine to make it easy to connect multiple stores together and treat them as one. For example, we can combine the DBpedia, the USGS Geonames database and Cogito information into a single *virtual* store and explore the interconnections between these datasets *without* worrying about where the triples originate. Even better, we can keep different kinds of triples separate and combine them as needed. E.g., we can keep known facts, inferred triples, provenance information, ontologies, metadata and deleted triples in separate, easily manageable stores and combine and re-combine the data as necessary.

Enterprise data volumes are growing without bound making it essential to enable the accumulation and archiving of multi-billions of triples. Federation lets you segment your data into usable *chunks* that can be swapped in and out as needed. Since federated data stores can be built and changed, it is simple to look at historical data whenever necessary.

2.5 Specialized Datatypes

AllegroGraph supports several specialized datatypes for efficient storage, manipulation, and search of Social Network, Geospatial and Temporal information.

2.5.1 Social Network Analysis

By viewing interactions as connections in a graph, we can treat a multitude of different situations using the tools of Social Network Analysis (SNA). SNA lets us answer questions like:

- How closely connected are any two individuals?
- What are the core groups or clusters within the data?
- How important is this person (or company) to the flow of information?
- How likely is it that two people know one another?

AllegroGraph's SNA toolkit includes an array of search methods, tools for measuring centrality and importance, and the building blocks for creating more specialized measures.

2.5.2 Geospatial Primitives

AllegroGraph provides a novel mechanism for efficient storage and retrieval of geospatial data. Support is provided both for Cartesian coordinate systems (i.e., a flat plane) and for spherical coordinate systems (e.g., the surface of the earth or the celestial sphere). Coordinates in two dimensions are encoded into a single UPI. Once data has been encoded this way, AllegroGraph can perform queries in either Cartesian or spherical coordinates very quickly. AllegroGraph's geospatial application also has support for defining polygons and quickly determining whether a point lies inside or outside a given polygon, whether two polygons overlap, and retrieving all triples that lie inside of a given polygon.

2.5.3 Temporal Primitives

AllegroGraph supports efficient storage and retrieval of temporal data including datetimes, time points, and time intervals. Once data has been encoded, applications can perform queries involving a broad range of temporal constraints on data, including relations between: points and datetimes, intervals and datetimes, two points, two intervals, points and intervals.

2.5.4 Freetext Indexing

AllegroGraph can build freetext indexes of the strings of the objects associated with a set of predicates that you specify. Given a freetext index, you can search for text using boolean expressions ("market" AND "housing"), wild cards ("science*" OR "math*"), and phrases ("Semantic Web search").

2.6 Gruff – Graphical Query Generator and RDF Browser

Gruff is a visual query generator for SPARQL and Prolog as well as a triple-store browser that displays visual graphs of a store's resources and their links. Gruff provides an intuitive interface to create queries without prior knowledge of SPARQL or Prolog.

For exploration of your data you can build a visual graph that displays a variety of the relationships in your data. Gruff can also display tables of all properties of selected resources or generate tables with SPARQL queries, and resources in the tables can be added to the visual graph. (Figures 3, 4, 5, and 6)

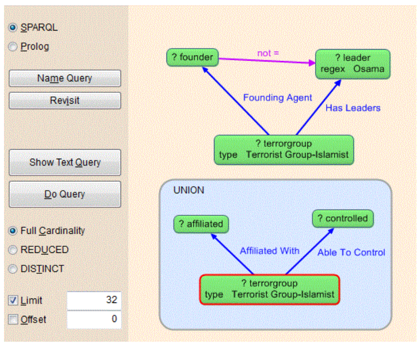


Figure 3:

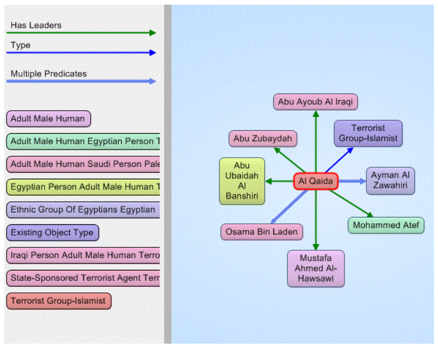
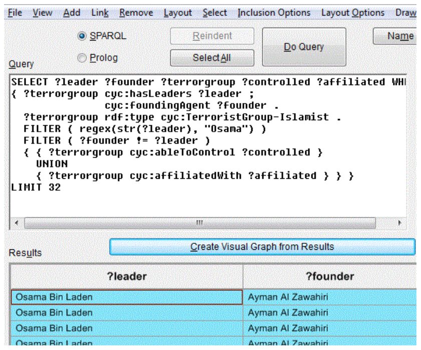


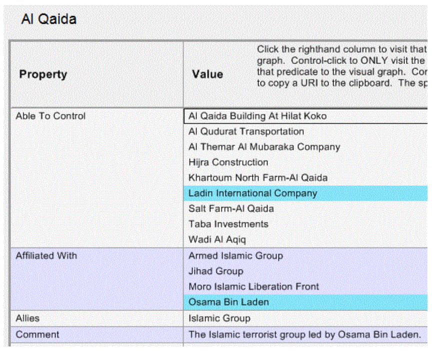
Figure 4:


File View Add Link Remove Layout Select Inclusion Options Layout Options Draw
SPARQL
Prolog
Reindent
Select All
Do Query
Name
Query
Results
Create Visual Graph from Results

```
SELECT ?leader ?founder ?terrorgroup ?controlled ?affiliated WH
{ ?terrorgroup cyc:hasLeaders ?leader ;
               cyc:foundingAgent ?founder .
  ?terrorgroup rdf:type cyc:TerroristGroup-Islamist .
  FILTER ( regex(str(?leader), "Osama") )
  FILTER ( ?founder != ?leader )
  { { ?terrorgroup cyc:ableToControl ?controlled }
    UNION
    { ?terrorgroup cyc:affiliatedWith ?affiliated } } }
LIMIT 32
```

?leader	?founder
Osama Bin Laden	Ayman Al Zawahiri
Osama Bin Laden	Ayman Al Zawahiri
Osama Bin Laden	Ayman Al Zawahiri
Osama Bin Laden	Ayman Al Zawahiri

Figure 5:


Al Qaida

Property	Value Click the righthand column to visit that graph. Control-click to ONLY visit the that predicate to the visual graph. Cor to copy a URI to the clipboard. The sp
Able To Control	Al Qaida Building At Hilat Koko
	Al Qudurat Transportation
	Al Themar Al Mubaraka Company
	Hijra Construction
	Khartoum North Farm-Al Qaida
	Ladin International Company
	Salt Farm-Al Qaida
	Taba Investments
	Wadi Al Aqiq
Affiliated With	Armed Islamic Group
	Jihad Group
	Moro Islamic Liberation Front
	Osama Bin Laden
Allies	Islamic Group
Comment	The Islamic terrorist group led by Osama Bin Laden.

Figure 6:

3 Expert System & AllegroGraph Integration

Franz and Expert System have created a prototype of a new product that integrates a scalable RDF triple store, a commercial grade entity extractor, and a configurable web spider. Overview of TexTriples capabilities:

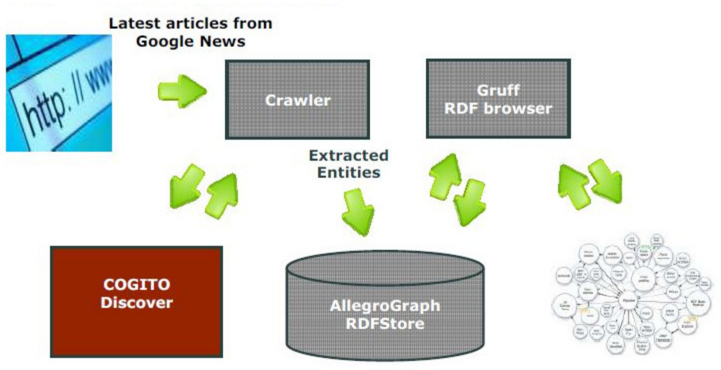


Figure 7

Direct Import. The simplest method of import is pointing TexTriples to a list of files or URLs, apply entity extraction to the text, and then turn those entities into RDF triples.

Spider Import. Slightly more complex is to point TexTriples to a website and specify a number of parameters (stay in website, only use if there is a person name in the main text, etc.). It will spider the entire website according to the parameters and turn text, via entities, into RDF triples.

News related Keyword import. Using a list of search terms TexTriples will spider a large number of news sources and blogs to find the most recent articles about these keywords and turn the text sources into RDF triples. The texts that are found are also annotated with the date of the article and the sentiment analysis score.

Open Link Data Linking (OLDL). If you set OLDL to 'true' TexTriples will:

- Automatically link place names to the Geonames database in order to add latitude and longitude to names.
- Link people names against FreeBase and DBpedia. We will provide some basic disambiguation to make sure that we have a high precision for these people. We also offer to link to user defined databases or external data sources like LinkedIn and Facebook.
- Link organizations to FreeBase and the Hoover's company database. The latter feature does require a subscription to Hoover's.

Graphical Query Generation. All the triples are queryable through our visual AllegroGraph Gruff interface using SPARQL (or Prolog for advanced usage). We provide a visual query builder so that analysts' don't have to go through the trouble of learning complicated query languages.

Advanced Text Mining. There are a number of techniques that we offer. For example you can find all the articles that look like article A based on predicates p1, p2, p3. Another example would be that we find per person the topics that were most popular over time.

Faceted Navigation. Finally, AllegroGraph has full text indexing over all the generated triples and TexTriples can perform faceted navigation of the combination of triples and text.

Scalability. The current corpus is approximately 8 billion triples with additions of approximately 10 million triples each day.

In this demonstration we will provide a "Politics" example where we track all politicians from the executive branch, Congress and the House of Representative for a number of months.