Extracting and Querying a Comprehensive Web Database

Michael J. Cafarella
University of Washington
Seattle, WA 98107, USA
mjc@cs.washington.edu

ABSTRACT

Recent research in domain-independent information extraction holds the promise of an automatically-constructed structured database derived from the Web. A query system based on this database would offer the same breadth as a Web search engine, but with much more sophisticated query tools than are common today. Unfortunately, these *domain-independent* Web extractors are usually not *model-independent*; *e.g.*, an extractor that only finds binary relations from text will be blind to relational data found in tables. Because a topic area often has a data model that is a natural fit (*e.g.*, population statistics are usually in tables, while biographical facts about Einstein are embedded in text), even a high-quality domain-independent extractor will miss a substantial amount of data.

Our OMNIVORE system attempts to build a comprehensive Web database by running multiple domain-independent extractors in parallel over a Web crawl, then combining their outputs into a single large entity-relationship database. Each item in the database describes a single real-world entity, and can contain information drawn from a number of popular Web data models. The user can correct flaws in the database, and can query it using either a structured query language or a search-like interface. Due to the Web's sheer size, users cannot be expected to know the result set's metadata *a priori*, so OMNIVORE automatically chooses an output model and schema when it renders results. In this paper we outline the OMNIVORE architecture and provide specific details about our current prototype.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Online Information Services; H.2 [**Database Management**]: Miscellaneous

1. INTRODUCTION

Domain-independent information extraction has been an active research area in the last few years, often using a large set of crawled Web pages as input [1, 8, 10, 17, 19, 20]. There is a huge amount of structured data on the Web, on almost every topic imaginable. The combination of a large Web crawl and a domain-independent extractor should enable the automatic construction of a live structured Web database, combining the breadth of a search engine with a powerful structured query language (instead of just textual relevance queries). Recently several researchers have proposed structured-Web database systems with varying levels of coverage and automation, though always with a strong information extraction component [3, 7, 11, 20].

A Web topic can strongly determine the model in which its data is expressed. For example, it would be surprising (as well as clumsy and error-prone) to publish economic statistics as a long series of natural-language sentences (*e.g.*, "France has GDP of *X*. Spain has GDP of *Y*...") rather than in tabular form. Similarly, some fact triples (such as biographical data) are expressed only or mainly in natural language. Unfortunately, while a domain-independent extractor is not tied explicitly to a topic, it is often tied to a data model. For example, the TEXTRUNNER system extracts fact triples from natural-language text: the extraction contains a binary relation and a tuple for that relation (*e.g.*, {`Einstein, was_born_in, Germany`} [4]. Meanwhile the WEBTABLES system extracts relational-style table

```
democrat.instances
UNION
republican.instances
```

```
SELECT country FROM ALL.instances
WHERE country.gdp > 2T
AND country.debt < 1T
```

```
SELECT a.location FROM company.instances
WHERE a.profit > 1000
AND a.num_employees > 100
```

Table 1: Sample queries expressed over a simple entity-relation data model for Web-derived information. Each entity in the database represents an extracted "real world" object. An entity is identified by a unique string (*e.g.*, `CIDR` or `Seattle`) and has a set of attributes/value pairs as well as a set of types. A query returns a set of zero or more entities. The first query uses type information to compute a new set of entities. The second query finds a subset of entities by filtering on attribute/value data. The third query combines filtering and type information.

CIDR '09 Monterey, California

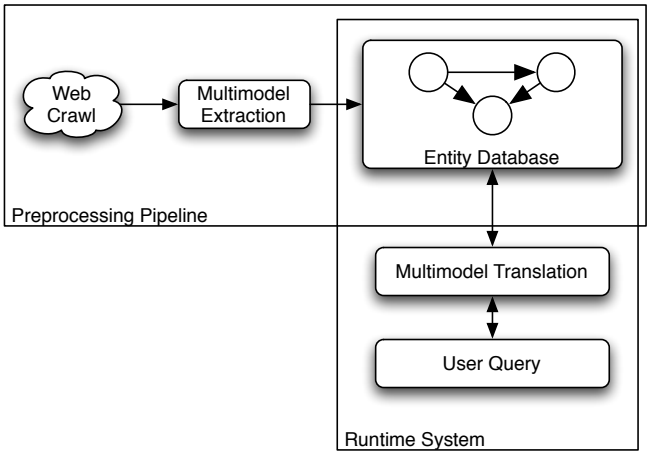

Figure 1: The OMNIVORE architecture. The preprocessing pipeline is a single large batch job that consumes a Web crawl and generates a large structured database of extracted entities. The runtime system applies read-only queries against the resulting database. Users can also correct errors in the query output, but these are not applied through the standard query mechanism.

data from HTML tables (*e.g.*, a table on population statistics) [10]. The combination of these two facts means that even a high-quality domain-independent extractor comes with a strong tie to certain domains and will miss out on a large amount of useful data.

Indeed, the author's experience with previous Web extraction systems (KNOWITALL, TEXTRUNNER, WEBTABLES) has been that while an extractor can achieve very high performance on its "signature" data model, it is fairly easy to formulate a query that cannot be answered with a given extractor's data [4, 10, 17]. (In contrast, a traditional search engine is relatively difficult to "stump.") The problem is exacerbated with queries that involve multiple extracted domains (*e.g.*, when performing a join or multi-predicate selection): a single extractor will fail to find data that satisfies such queries unless all the extractor is suited to all of the domains. For example, a query that seeks the GDP of all countries where Einstein visited would involve data best derived from multiple extractors/data models.

We encounter a second problem when generating query results against a comprehensive Web-derived database. It should be possible for a user to query the database without knowing how the query results (if any) should be structured. For example, suppose the user queries for all entities that are located in Seattle. The query:

```
SELECT x FROM ALL WHERE x.location=Seattle
```

would return companies, people, buildings, *etc.* The best possible output format for these results is probably a series of domain-sensitive relational tables, one for each entity type. The output schema(s) should differ with another result set. The scale of the Web and the general low quality of data has a heavy impact on query-processing for the Web database: not only is there no single schema for the user to query, the user cannot even reasonably request a target schema without knowing the query results ahead of time.

Data model decisions are deeply tied to the data's domain, both at database-creation time and at query time. In a traditional (generally single-domain) database setting, a single data model and schema limit the actions of data-creators and query-writers. But in a Web setting, centralizing control in this way is not feasible. Each Web data creator simply chooses the model and schema that is most-appropriate to her own small part of the overall Web database. The Web database query-writer is handicapped even further due to the sheer volume of topics and data; she may not be able to describe a target model and schema without knowing the actual query result.

In response, we propose the OMNIVORE system. OMNIVORE offers two main architectural contributions, each of which motivates an algorithmic database problem.

First, it attempts to recover a comprehensive set of Web topics by running multiple domain-independent extractors in parallel, then combining their outputs into a single entity-relationship database. Each item in the database represents an extracted "real-world" entity, with associated types and attribute/value pairs. (Table 1 shows a few example queries against the Web database.) Simply combining multiple extractor outputs can improve topic coverage. In addition, we can sometimes boost the quality of individual extractors by allowing them to share raw extraction information. For example, imagine that a simple association-extractor finds that `Einstein` and `patent-clerk` are strongly associated with each other, though it cannot extract the relationship between them. Meanwhile, a fact-triple extractor finds weak information that {`Einstein, had-job, patent-clerk`}. By translating the extractions into a common data model, we can find stronger evidence for the {`Einstein, had-job, patent-clerk`} triple. Combining extractions from multiple sources into a single entity database is thus similar to a reference reconciliation problem, but with a dataset that can easily run into the hundreds of millions of data items, far beyond any published reconciliation results that we are aware of (many of which focus on paper citation databases of just a few thousand items) [13, 16, 18].

The second contribution of OMNIVORE is to render query results using an automatically-chosen "presentation-time" data schema and model. For example, if the above `Seattle` query were to return mainly corporations, OMNIVORE could present its data with a relational table that has attributes `CEO`, `earnings`, *etc.* In this way, the query-writer does not have to indicate a schema ahead of time, but can enjoy the benefits of seeing result data in a relevant schema (without first examining all the results by hand). This task is very similar to the *ModelGen* operator from the model management literature: the system must map some source data to an appropriate target schema, given only the data and the target data model [2, 5]. (Indeed, we can also frame the extractor-to-Web-database mapping task as instance of *ModelGen*, in which each source model is determined by an extractor, a source schema is determined by the extractor's output, and the target model is the Web database's entity-relation model.) However, unlike previous implementations of *ModelGen*, ours must operate without any human supervision or revision, and must incorporate some user-modeling aspects to determine a candidate schema's quality.

Note that OMNIVORE's focus is on surfaced, and general

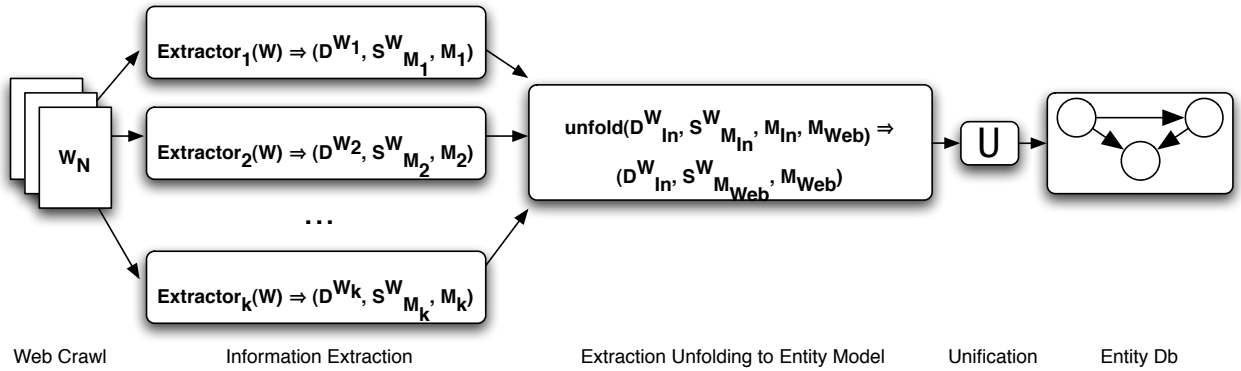

Figure 2: The OMNIVORE preprocessing pipeline. It runs k extractors in parallel, transforms these extractions into the entity-relation data model using model-specific *unfold* operators, then performs the *unification* step, which entails reference reconciliation and extraction thresholding. The result is a single entity-relation Web database.

structured Web data. Thus, we do not spend any effort on extracting data from the deep-web *per se*, though we may indirectly obtain the surfaced part. Also, we focus on the broad class of data that we believe can be expressed using a handful of popular domain-independent data models; while the OMNIVORE architecture supports site-specific extractors, we believe they are in general so expensive that there will be too few to cover most of the Web. The objection that domain-independent extractors will never recover the structured Web as well as domain- or site-specific extractors is beside the point if the latter can only be obtained at a prohibitive cost. Thus, to a large extent, the success of OMNIVORE as a usable system depends on whether most of the Web's data is expressed using a fairly-small number (say, a dozen) of simple extractable models.

As Figure 1 shows, our proposed architecture for OMNIVORE closely resembles a traditional search engine: a major crawl/batch stage followed by query processing. The extraction pipeline is described in Section 2, and the resulting Web database in Section 3. Users can then pose structured read-only queries (and database corrections) against the runtime system, which we describe in Section 4. Finally, we briefly discuss the status of our current prototype in Section 5, in which we make specific decisions about things like how to compile the Web crawl, which extractors to deploy, *etc.* We then conclude with a brief discussion of related work in Sections 6 and 7.

2. EXTRACTION AND DATA INTEGRATION

The goal of the OMNIVORE preprocessing pipeline is to produce the entity-relation Web database for later query processing. As Figure 2 shows, it works by taking a general Web crawl and running it through three processing steps.

Step 1. Information Extraction. The system runs k extractors in parallel over the input pages. Each extractor E_i has a fixed associated data model M_i. (As we describe in Section 5, our current implementation has four extractors with four corresponding data models, including relational tables (from HTML tables) and typed entities (from natural-language text), and others). When an extractor is run on a given page W, it emits a piece of data D^W that is described using extractor-dependent model M_i in a schema that depends on both extractor and Web page: $S^W_{M_i}$.

Step 2. Extraction Unfolding. The next step is transforming the extracted data into the entity-relation Web data model, which we *unfolding*. For each extractor output $(D^W, S^W_{M_i}, M_i)$, the *unfold* operator translates the data into the target entity model M_{Web}. Each extraction that is sent to *unfold* results in output $(D^W_{In}, S^W_{M_{Web}}, M_{Web})$.

The entity-relation model is a common "metamodel" used to describe schemas in other data models. Our implementation for *unfold* can in some cases be extremely basic. For *types*, *ModelGen* creates an entity for both instance and type, filling out the `instances` and `type` relation fields appropriately. For a *fact triple* (A, R, B), we create an entity for A/B. We then add attribute R to A, with value B. The directionality of R is determined by the source language string that links the two entities.

Transforming data from a *relational table* is somewhat more interesting. If the leftmost column of the table is non-numeric and unique, we assume it acts like a primary key for the table and is thus probably a good guess to label an entity. There are $|MN|$ examples of $(key, attr, val)$, where M is the number of columns, and N is the number of rows in the table. We add entities *key* and *val* to the database, with relationship *attr* between them. This algorithm essentially assumes a functional dependency between the primary key and every column in the table; this is not a good assumption in general, and in the future it would be better to attempt to detect when these dependencies are present. We also create some synthetic entities to represent each tuple and the overall relational set, as we discuss below in Section 3.

Note that *unfold* can also be described as a very straightforward implementation of the *ModelGen* operator.

Step 3. Unification. We now have a huge amount of data that is framed in a single data model, but it still does not make up a single usable database. To assemble a coherent database, OMNIVORE has to *reconcile* all the entity references made across a huge number of extractions. For example, references to `George_Bush` should be unified to (at most) two individual objects. Also, OMNIVORE must *threshold* bad extractions where possible so they do not enter the database. Sifting good from bad extractions is a task usually embedded inside the extractor, but we move this step into the unifica-

tion stage so we can take advantage of reconciled references and data from multiple extractors.

There is a substantial body of work on algorithms for reference reconciliation (such as [13, 16, 18]) and it is a difficult challenge for even small datasets (regarding both output quality and computational difficulty). The OMNIVORE task is much harder than most previous instances of the problem, as the set of extractions entering the unification step can easily reach hundreds of millions of items, scaling with the product of the number of extractors and the number of Web pages in the input crawl. Further, it must reconcile references without any domain-specific schemas or rules.

However, unlike many reference reconciliation datasets, OMNIVORE enjoys a very dense set of constraints on the set of possible reconciliations. A single data item that reaches the reference reconciliation step has four associated pieces of extraction-lineage: the entity reference itself, the source extractor, the source data object (*e.g.*, a region of a Web page), and the source Web domain. We can eliminate a vast number of potential reconciliations by making a few broad assumptions about Web authorship (*e.g.*, references on a single page always consistently refer to the same entity; an entity label is not plausible unless it appears via several extractors).

Beyond the algorithmic step, OMNIVORE follows a few rules that make reference reconciliation a more handleable problem. Because most entity labels do not refer to multiple real-world objects, string matches imply entity matches by default. Thus, we would start with a single `George_Bush` item (barring any action by a pipeline-time reconciliation algorithm). The alternative is to create a new object for every new extracted entity label, even if the label has been seen before; such an approach is tailored to the less-frequent case and entails integrating even just multiple item mentions on a single page. We believe the better strategy is an "optimistic" one that assumes identical strings refer to the same entity, while also running tests to detect the comparatively few cases when the assumption is a bad one. For example, when the system detects that the `George_Bush`, `George_W_Bush`, and `George_H_W_Bush` objects overlap and are obviously very close in string similarity, OMNIVORE can ask the user whether `George_Bush` should be split into multiple objects.

Thresholding simply means deciding whether an extraction is "good enough" to be admitted to the database. Extractors have their own techniques for performing thresholding, often based on probabilistic models or extraction frequency. OMNIVORE creates a thresholding classifier via a machine learning technique called cotraining. Cotraining uses multiple distinct classifiers (one for each extractor, in our case) to generate additional training data for each other, thus *bootstrapping* each classifier into higher performance [6]. Data from distinct extractors form distinct "views" of an entity and thus are a good cotraining candidate.

Finally, the unification step can also include explicit user "corrections" of previous versions of the database. The stream of user updates to the Web database is simply treated as a special "extraction" stream that is always assumed to be correct. Combining user and machine-driven updates into a single collaborative database is a relatively-new but very exciting area of research [7, 14, 22].

3. THE ENTITY WEB DATABASE

The heart of OMNIVORE is an entity-relation database that describes every entity we can extract from the Web. There is a unique data object for each entity (with a unique identifier), and each entity object has a label that is either unique or equal to "anonymous." An entity has an associated set of attribute/value pairs. It can have zero or more types (*e.g.*, the `Seattle` item has type `city`, `haven`, and several others). A data value can be one of the standard primitive types (*i.e.*, `string`, `integer`, *etc.*), a reference to another item in the database, or a set of those values.

The `type` and `instances` attributes are set-valued and implement type relationships (*e.g.*, the `Albert_Einstein` object has `scientist` in its `type` field). The special `ALL` object has an `instances` attribute that points to every object in the database. A query returns a set of zero or more objects. A query can refer to arbitrary attribute names; a predicate test that involves an object's non-existent attribute will simply evaluate to false. Figure 2 shows some sample data from the extracted `Seattle` object.

The Web data model as described so far is fairly straightforward and can represent data from a number of different models. We also make a few adjustments so it can accommodate extracted data more easily.

First, attributes can be anonymous. It is often true that it is much easier to relate an entity to a data value than it is to precisely describe that relationship. For example, `Albert_Einstein` and `scientist` should be quite strongly related, but it would be quite difficult to extract the `jobtitle` relationship.

Second, our notion of *type* is very weak and implies nothing about inheritance between super and subentity; a type is simply a way of describing a 1:n relationship between items. We implement types by filling out two special attributes in each entity: `type` and `instances`, which each hold a set of references to other database objects. We do not want to enforce strict inheritance-style rules because even a mainly-correct extracted object can lack attributes and values (whether because of flawed extractors or incomplete source data). Instead, types are used mainly as a shorthand for the query-writer to refer to a well-known set of items. Note that because a OMNIVORE type is simply a set and a label, a "type" is functionally the same as a "set."

All of our examples so far have involved explicitly-labeled entities, but this is not required. The extractor may find an item that appears to be a real-world entity, but for which there is no good unique key. (Consider that it would be very useful to distinguish between multiple instances of `Toyota_Camry` extracted from a collection of classified ads. The classified ads probably do not provide a unique label for each.) Thus, OMNIVORE will create some anonymous objects; for example, assuming that each tuple in a relational table represents some useful entity, it may create a new synthetically-named object for each.

Similarly, human-understandable types/sets such as `city` are easy to label, though others are not. For example, every relational table carries a set-membership assertion about each tuple (or at least primary key) that the table contains. OMNIVORE thus can create unlabeled types/sets just like it can with entities.

Objects do not require labels, but no two objects can share a single label. Not only would such "aliased" labels be confusing for the user, there is some evidence that they are unnecessary. Existing named-entity collections such as

attribute	value	extractor	lineage
name	`Seattle`	*various*	...
type	`city, startup_hub`	KnowItAll	...
instances	*none*		
ismayorof	`Greg_Nickles`	TextRunner	...
population	592,800	WebTables	...
?	`Northwest`	WeakAssoc	...
?	`grunge`	WeakAssoc	...

Table 2: Selected data associated with the `Seattle` entity. Each extraction is also logged with its extractor and associated lineage information (source page(s), actions taken by the unification and reference reconciliation steps, *etc.*).

Wikipedia have shown that even a fairly large collection of items can be distinguished by label. (*E.g.*, there is no "George Bush" entity in Wikipedia, but rather one for "George W. Bush" and one for "George H.W. Bush.") When two real world entities genuinely do share a label, it is the responsibility of the extraction and integration pipeline (from Section 2 above) to either generate or ask the user for additional distinct object-appropriate labels.

There are two methods of querying the system. The structured query interface takes as input a structured query and a set of entities; it also returns a set of entities. The query language offers set operators, and selection predicates on entity attributes. A set may be specified by one of:

- A labeled set of objects, such as `city.instances`
- The `ALL` keyword, meaning the set of all items in the database.
- A subquery

Figure 1 has a few examples of the structured query interface.

The unstructured query interface takes as input a text search string; it returns a ranked list of entities relevant to the input (we might rank the `CIDR` entity highly with the query `database conference California`). As part of its ranking task, the ranker may use information external to the extracted database itself, such as a Web corpus. For example, OMNIVORE may use the query string to find a set of highly-relevant documents, and then rank all entities that appear within the given docs.

4. QUERY PROCESSING

The OMNIVORE query interface, depicted in Figure 3, is responsible for two main tasks.

The first task is accepting queries (both structured and search-style) and returning results to the user. The Web database itself returns a simple set of entities (this level of query-processing is fairly straightforward so we do not discuss it any depth in this paper).

However, note that items from the Web database share no common schema for the objects, and objects may have very different types. Recall the query for `Seattle`-located items from Section 1 above. Without a schema, the naïve method of presenting query results is a simple list of objects. This would be a terrible experience for the end-user, who has probably not queried for a completely-arbitrary set of

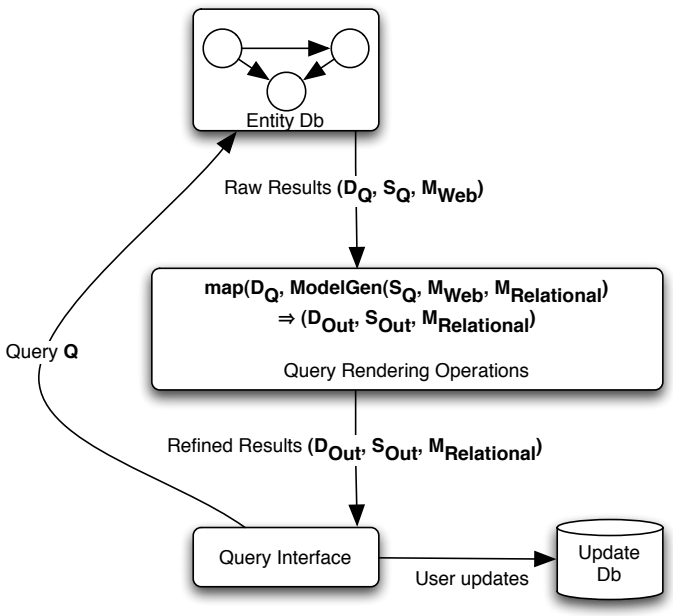

Figure 3: The OMNIVORE query processing system.

entities. It is more likely that the returned objects (or subsets of the returned objects) share some relevant attributes in common. So, rather than presenting raw objects to the user, OMNIVORE transforms the results into an on-the-fly relational table with its own "presentation schema." This schema has a row for each returned object and a relevant attribute in each column. Computing a mapping between the query result data and an appropriate schema in the relational model is similar to mapping the query result data D_Q to a relational schema produced by the *ModelGen* operator. *ModelGen* in this case finds a relational schema that is appropriate given the Web entity-relation data's schema.

Choosing a high-quality output schema mapping for (inevitably) incomplete extracted objects can be difficult, involving user-interface decisions, competing optimization criteria and fast runtime requirements. For example, two subsets of the query results may have data that only partially overlaps, forcing the system to either place the subsets in different tables or emit a table with many NULL values. The TGen system previously attempted to create an optimal relational schema for a set of extracted objects, though with execution times firmly above a minute for large data sets [12]. For both efficiency and result quality, OMNIVORE constrains the *ModelGen* task substantially by simply allowing only one relational table per `type` in the query result set.

The second main task is accepting user feedback to correct inaccurate data in query results. A user can add, delete, or update any object in the returned data, possibly examining the lineage data that accompanies each result. This data is logged and stored until the database-construction pipeline is run again. Currently, we simply use the feedback to correct the end-product, but eventually it should be possible to "push the feedback down" and thus boost or lower confidence in the source extractor, data object, or site so that user-given corrections have impact beyond the fact immedi-

ately at hand.

5. PROTOTYPE SYSTEM

We are in the midst of constructing an OMNIVORE implementation, on a general Web crawl of about 60M Web pages. So far we are using four different extractors:

1. The KNOWITALL extractor, for type information derived from natural language.
2. The TEXTRUNNER extractor, for relation-independent fact-triple information derived from natural language.
3. A simplified version of the WEBTABLES extractor, for relational tables derived from the HTML `table` tag. (We do not have access to the original WEBTABLES implementation as described in [9] and [10].)
4. A novel "extractor of last resort" called WEAKASSOC that operates over any text, including nonstandard (and unparseable) natural language such as classified posts. It emits anonymous-attribute/value pairs for an entity label. This output is less structured compared to other extractors', but because WEAKASSOC assumes very little about the input Web page structure, it can find relationships that might otherwise be difficult or impossible to find.

   For example, `Michael_Cafarella` is fairly strongly associated with `Alon_Halevy` on the Web, but recovering the `coauthor` relationship label would be very difficult. WEAKASSOC output is not very useful by itself, but is helpful for bolstering detailed-but-weakly-supported data from other extractors.

To support fast query processing, our prototype distributes the resulting object database over a cluster of 35 nodes. We are in the process of adding index support to quickly map from attribute labels and values to the containing object.

6. RELATED WORK

A few projects have focused on query processing, or on integrating extracted data from various sources. IBM's AVATAR project combines the results of many extractor-like "annotators," which consume text and emit typed (*e.g.*, *person*) [21]. While AVATAR itself is not domain-specific, it would require a huge number of domain-specific extractors in order to process the entire Web.

Our EXDB system used a probabilistic database to combine information from a few different extractors, with several notable differences from OMNIVORE: all extractions were derived from natural language text, the EXDB data model was limited to triples and *is-a* hierarchies, and information from multiple extractors was never combined at the raw-count level [11]. The DBLife project is a useful *data portal* of information about database science and researchers, but this one domain requires nontrivial human intervention, an approach we do not believe can scale to all topics on the Web [15].

Metaweb's Freebase system is a "wiki"-style collaborative structured database, with a broad range of topics, similar to Wikipedia [7]. The system is designed primarily for human updates, but is seeded with structured data extracted from Wikipedia and other more structured sources. It has a similar entity-centric data model to OMNIVORE, though with a much stronger type system. The DBPedia project has similar ambitions to be a "general-purpose" structured database, though it relies almost exclusively on RDF triples derived from Wikipedia [3].

7. CONCLUSIONS

Simply compiling a large amount of data using a domain-independent extraction technique has proven insufficient for creating a truly comprehensive database of Web information. OMNIVORE attempts to solve many of these problems (*e.g.*, poor extraction coverage, poor user knowledge of any relevant schemas) by placing the data model management-related tasks at the center of the extraction system architecture. We believe that as lower-level pieces of extractor technology become more established, data model issues like those discussed in this paper will become critical hurdles that all usable Web-scale extraction systems must overcome.

8. REFERENCES

[1] E. Agichtein, L. Gravano, V. Sokolovna, and A. Voskoboynik. Snowball: A prototype system for extracting relations from large text collections. In *SIGMOD Conference*, 2001.

[2] P. Atzeni, P. Cappellari, and P. A. Bernstein. Modelgen: Model independent schema translation. In *ICDE*, pages 1111–1112, 2005.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[4] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.

[5] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[6] A. Blum and T. M. Mitchell. Combining labeled and unlabeled sata with co-training. In *COLT*, pages 92–100, 1998.

[7] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[8] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[9] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[10] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Webtables: Exploring the power of tables on the web. In *VLDB*, 2008.

[11] M. J. Cafarella, C. Re, D. Suciu, and O. Etzioni. Structured querying of web text data: A technical challenge. In *CIDR*, pages 225–234, 2007.

[12] M. J. Cafarella, D. Suciu, and O. Etzioni. Navigating extracted data with schema discovery. In *WebDB*, 2007.

[13] A. Culotta and A. McCallum. Joint deduplication of multiple record types in relational data. In *CIKM*, pages 257–258, 2005.

[14] P. DeRose, X. Chai, B. J. Gao, W. Shen, A. Doan, P. Bohannon, and X. Zhu. Building community wikipedias: A machine-human partnership approach. In *ICDE*, pages 646–655, 2008.

[15] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, pages 399–410, 2007.

[16] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[17] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale

information extraction in knowitall (preliminary results). In *Thirteenth International World Wide Web Conference*, 2004.

[18] P. Singla and P. Domingos. Entity resolution with markov logic. In *ICDM*, pages 572–582, 2006.

[19] F. M. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *KDD*, pages 712–717, 2006.

[20] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[21] T.S.Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull*, 29(1):40–48, 2006.

[22] D. S. Weld, F. Wu, E. Adar, S. Amershi, J. Fogarty, R. Hoffmann, K. Patel, and M. Skinner. Intelligence in wikipedia. In *AAAI*, pages 1609–1614, 2008.