# Towards *ECSSE*: live Web of Data search and integration

Richard Cyganiak
Digital Enterprise Research Institute, National University of Ireland, Galway
richard@cyganiak.de

Michele Catasta
Digital Enterprise Research Institute, National University of Ireland, Galway
michele.catasta@deri.org

Giovanni Tummarello
Digital Enterprise Research Institute, National University of Ireland, Galway
giovanni.tummarello@deri.org

## ABSTRACT

We illustrate the works toward implementing an Entity Centric Semantic Search Engine (*ECSSE*). ECSSE leverages the Sindice Semantic Web Index to find and combine together semantically structured data published on the web. With respect to previous Semantic Web Data integrators, ECSSE, uses an holistic approach in which large scale semantic web indexing, logic reasoning, data aggregation heuristics, ad hoc ontology consolidation, external services and user interaction all play together to create rich entity descriptions and live, embeddable data mash ups.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.3.3 [**Information Search and Retrieval**]: Information filtering; H.3.3 [**Information Search and Retrieval**]: Selection process

## General Terms

Entity Search, Data Smushing

## Keywords

Sindice, ECSSE, OKKAM

## 1. INTRODUCTION

In the last two years, the amount of structured data available on the Web in interoperable format (Web of Data) has grown by several orders of magnitude. The phenomena mainly responsible for this are Semantic Web Linked Data, Microformats and the emergence of RDFa.

The Linking Open Data effort[1] has made available online hundreds of millions of RDF based entity descriptions in datasets like DBpedia, Uniprot, Geonames etc. While most of the efforts are still due to a relatively small number of institutions and individuals, the amount of information made available is stunning, with billions of semantic statements estimated online [5] and growing rapidly.

Microformats[2], on the other hands, have enjoyed a good following especially in the Web 2.0 community, with large web sites embedding them to provide at least some element of structured information in certain pages (Among these one can cite, e.g., LinkedIn, YouTube, Digg, Last.FM, Facebook and many others). While we are not aware of updated public statistics for the use of microformats, it is safe to say that the number of RDF triple equivalent is also reaching billions.

RDFa, W3C recommendation for serializing RDF inside HTML [1], is also proving more and more popular as it enjoys the simplicity of Microformats while allowing easy extensions.

While the amount web data and its growth have been notable, few applications still make it clear why producing this data is valuable to the end user and to the data producers themselves.

So far, probably the most notable has been Yahoo Search Monkey[3]. Search Monkey is a functionality integrated in the Yahoo search engine that provided enhanced visual representation of results from pages containing structured data in microformats or RDF. While this certainly provides some incentives for web data producers, Search Monkey limits itself to visual enhancements on its single result listing.

In this paper we present our works toward an Entity Centric Semantic Search Engine (ECSSE), an search application that leverages the Sindice Semantic Web Index [10] to automatically integrate and make use of information coming from multiple web sources.

Possibly more interesting, ECSSE doubles as interactive tool for on the fly creation of semantic data mash ups displaying live web of data information. The user is given the ability to refine the mash ups, adding and removing sources as needed, and the final, live, mash up can be embedded in any Web page using JavaScript tags.

## 2. ON NAVIGATING THE WEB OF DATA

[1] `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

[2] `http://microformats.org`

[3] `http://developer.yahoo.com/searchmonkey/`

Navigating the web of data, has been long time desiderata for the Semantic Web community. While single datasets, e.g. DBpedia, can be interactively navigated, the goal of having environments where information would automatically or semi automatically be displayed from multiple sources has been elusive.

In one such environments, information would flow to the user simply becouse it is relevant to the entity currently visualized or the user task at end. As a result, this would demonstrate "reuse of information outside the context in which it has been created", possibly one of the most appealing goals in semantic web research.

So far two notable approaches have been demonstrated. In 2006, the SWSE Semantic Search engine demonstrated large scale aggregation of Semantic Web data [4]. Possibly for the first time, thanks to the use of a scalable cluster infrastructure, SWSE could collect and contain a significant part of the data available on the Semantic Web so to be able to aggregate information pages with elements coming from multiple sources talking about the same entity. To perform such entity information consolidation, SWSE strictly adhered to the theoretical rules of the Semantic Web:consolidation via reuse of the same identifier across different data sources and several forms of lightweight reasoning such as explicit SameAs statements, OWL Inverse Functional Properties etc [6].

As a result, the engine displayed two peculiar side effects, as discussed in [4]. On the one hand, for each textual query indicating an entity, multiple Semantic Web Entities would be displayed, mostly due to the very scarce reuse of URIs across different sources [4]. On the other hand, when often information would be wrongly aggregated due to errors or different interpretations of semantic properties across different datasets. For example, if a property, e.g foaf:homepage, is defined as Inverse Functional Property, then all the entities that had this value set to null would share the same, erroneous aggregated set of statements.

A completely different approach, is that of the Tim Berners Lee initiated Tabulator project[7].

In Tabulator, the idea is to leverage the linked data principle: data published on the Web in RDF should have dereferenciable identifiers (URIs). If this is the case, then the identifier doubles as a network location, so it is possible to fetch the description of the entity by resolving, e.g. with an HTTP lookup, the identifier itself.

In Tabulator, once the user enters a starting resolvable URI, the entity description is fetched and the contained statements are displayed, typically in form of a statement tree. The interesting part comes when the user decides to investigate on one of the leaf of said tree. If the leaf is itself a resolvable URI, then the description of the URI is also fetched and the new statements coming from the new location are therefore added to the old ones. This, in practice, creates a live data mash ups driven by how the user decides to explore the graph.

This approach, while fascinating in theory, suffers for certain shortcomings. In theory, for tabulator to consistently return information, the resources should always contain backlinks, that is, all the statements that are known to be in common with other dereferenciable resources. This is clearly a daunting maintenance task which is much against the nature of the web and the way people create their dataspaces. In ECSSE, we will see that this role is fulfilled instead by the Sindice index. A further shortcoming is the dependence on identifier reuse. If a dataset doesn't mention explicitly the URI for the same conceptual resource in another dataset, no browsing and data mash up can be possible.

[4]e.g. http://swse.deri.org/list?keyword=giovanni+tummarello returns 44 RDF nodes each with different informations attached

# 3. ECSSE: PROCESSING WORKFLOW

ECSSE revolves around the creation of *Entity Profiles*. An entity profile is a summary of an entity that is presented to the user in a visual interface. Entity profile usually include information that is aggregated from more than one source. The basic structure of an entity profile is a set of key-value pairs that describe the entity. Entity profiles often refer to other entities, for example the profile of a person might refer to their publications.

A *source*, in our terminology, is a Web document that contains structured data. Examples include RDF/XML documents, HTML pages with embedded RDFa markup, or HTML pages with embedded microformats.

There are several ways how a user can instruct ECSSE to display an entity profile:

1. Based on a keyword query
2. By following a hyperlink from one entity profile to another
3. By accessing a *permalink* to an entity profile (possibly created by another user) or simply by visualizing a web page where a Javascript tag embeds an entity profile via a permalink.

The process of creating an entity profile involves the following main steps, which are described in detail later in this section:

**Initial source acquisition:** A set of initial sources is identified based on keyword search, explicitly named URIs.

**Fetching and parsing:** The content of the sources is retrieved from a web cache, or directly from the web for a limited number of cache misses. Structured data is extracted from each source.

**Resource ranking:** The structured data extracted from each source is broken down into chunks that each describe distinct entities. These chunks, called *resource descriptions*, are ranked based on a number of criteria, most importantly proximity to the given keyword query. The top descriptions are selected for inclusion in the entity profile.

**Source set expansion:** The set of sources is expanded based on `owl:sameAs` links and inverse functional properties found in the descriptions. Similarly new sources can potentially found levering the OKKAM entity naming service (see later for a description). All these newly discovered sources are again fetched, parsed and ranked.

**Entity profile creation:** The descriptions are merged into a single entity profile.

**Property consolidation and ranking:** Often different properties (keys in the key-value pairs that describe the entity) express the same thing. Properties are consolidated based on simple linguistic heuristics and a manually-created set of consolidation rules. Properties are ranked for display purposes.

**Value labeling:** For key-value pairs where the value is not a literal value but a reference to another resource (usually by URI), a best-effort attempt is made to retrieve a good label for the resource. This might cause additional retrievals from the web cache or the web.

**Value consolidation:** If a property has several values with identical or very similar labels, they are collapsed into one value for visual presentation. Values are ordered.

**Source list refinement:** After the entity profile is presented to the user, they can refine the list of sources by removing poor sources (that talk about a different entity or introduce noise), by requesting more sources (which will cause lower-ranked sources to be retrieved and merged into the profile), or by adding further keywords (which will cause a new set of initial sources to be acquired and merged into the profile).

## 3.1 Data acquisition

ECSSE operates on data collected as part of the Sindice[10] project.

The Sindice infrastructure uses the RDF data model as a lingua franca that hides the syntactic differences between source formats. A set of parsers, which we are currently publishing as open source under the name *any23*[5], is used to extract RDF data from those different formats. Different RDF serialization formats (RDF/XML, Notation 3, Turtle, N-Triples, RDFa) are supported, as well as several microformats (hCard, hEvent, hListing, hResume and others). Conceptually, any format for which a converter or extractor into RDF is available, can provide input for ECSSE and Sindice.

Sindice collects data from the web using a number of techniques: web crawling, RDF dump indexing based on Semantic Sitemaps[2], and receiving update notifications (*pings*) from sources such as PingTheSemanticWeb.com and our own ping interface.

After documents have been fetched from the Web and their structured data parts have been extracted, the structured part is stored in a highly specialized index that facilitates keyword queries as well as more advanced query forms based on triple patterns. The index, based on information retrieval technology, as well as part of the infrastructure is described in [10][8].

The structured data extracted from web documents is also stored in the HBase based *page repository*, which allows subsequent fast access to the documents' contents without incurring the cost of web retrieval. It is simply a large distributed hash map that contains the content fetched from each URL that Sindice knows about, as well as additional metadata, most notably an inference closure computed over the document's RDF graph using the quarantined reasoning technique described in [3].

## 3.2 Requesting an entity profile

The process of creating an entity profile takes three inputs, each of which is optional:

1. A keyword search phrase
2. A number of source URLs
3. A number of resource identifiers (URIs)

The difference between the last two items is that a source URL names a document, which is accessible on the web, and might contain descriptions of any number of entities. A resource identifier names a specific entity, but may or may not be resolvable to a web document.

The initial user interface shown to an ECSSE user presents an input box that allows entry of either a search phrase, or a single resource identifier. Other combinations of inputs are accessed through hyperlinks either from within ECSSE or from a permalink.

## 3.3 Initial source acquisition

The first challenge is to identify a set of initial sources that describe the entity sought for by the user. This is performed via textual or URI search on the Sindice index and yields to a set of of source URLs that are added to the input source URL set.

Next, a search for each resource identifier is performed in the Sindice index. The Sindice index does not only allow search for keywords, but also for URIs mentioned in documents. This allows us to find documents that mention a certain identifier, and thus are likely to contribute useful structured information to the description of the entity named by the identifier.

Now we have a list of sources that potentially describe the entity signified by the user query. The list is naturally ranked: Sources directly specified in the input come first, and the other sources are ranked as returned by the Sindice index.

If the source list is long, it is trimmed. The desired length is still subject to experimentation, but 25 sources seems to be a good compromise of response time, data variety, and it is still manageable in the user interface. If there is a large number of sources from a single domain, then these are dropped with preference. This ensures a larger data variety and produces

[5]http://code.google.com/p/any23/

what appears to be a more interesting default search result. The user interface has then a control for requesting more resources, which repeats the process with a higher source cutoff limit.

## 3.4 Fetching and parsing

In the next step, we retrieve the RDF content extracted from each source. This uses a three-tier cache system: First, a memcached server is consulted. Second, the page repository is queried. Third, an HTTP request to the source URL is performed to retrieve its contents. In the third case, the Sindice parsers and extractors are invoked to get the structured content from the source. Whatever the result, it is stored in memcached to speed up subsequent requests.

If a successful request to the web was performed, then the source URL is sent to the ping manager module of Sindice, which ensures it will be scheduled for later fetching by the main Sindice infrastructure. This will result in adding it to the index and the page repository, including triples inferred during reasoning. Web requests especially if users browse from one entity profile to another referred entity. It is a nice way of discovering new sources that are relevant to user interests, and a low-cost method (compared to undirected web crawling) of increasing the coverage of the Sindice index and the page repository.

## 3.5 Resource ranking

The structured RDF graph extracted from each source is broken down into chunks (called *resource descriptions*) that each describe distinct entities. This is made easy by the use of the RDF data model. A resource description contains the outgoing and incoming RDF triples of a specific resource.

In some cases it would be desirable to include more information into a resource description. An example are geographic locations, which are often attached to a resource via a property such as `foaf:based_near`, which points to another resource, often an RDF blank node, which in turn has properties `geo:lat` and `geo:long` that give the geographical coordinates. Obviously it would be good to have the coordinates included in the resource description, even though they are only indirectly attached to the resource in question. This could be solved either by manually identifying commonly occurring cases such as the one given here, or by using generic heuristics based on graph shape, e.g. include linked blank nodes that have less than a certain number of outgoing triples.

We discard trivial resource descriptions (e.g. those that contain just one triple).

As an example of a decomposition into resource descriptions, consider the case of a typical FOAF[6] file that describes a person. It will be decomposed into one resource description for the file's owner, one (small) description for each of their friends listed in the profile, and possibly one description for the FOAF document itself (containing statements about its `foaf:maker` and `foaf:primaryTopic`.

Resource descriptions are now ranked. If the resource has one of the resource identifiers from the source acquisition step, then it will immediately receive a large boost, as there is almost total certainty that it described the entity in question.

Each description will be matched and scored against the keyword phrase, considering both RDF literals and (with a lower score) words in URIs. This helps to pick out the correct resource in cases such as FOAF files, which talk about multiple people, but it is easy to select the right one given a name.

Very small entities are slightly reduced in score, because experimental results show they are unlikely to contain interesting information, while cluttering up the source list in the user interface.

Resource descriptions below a certain threshold are removed from consideration. We now have a ranked list of descriptions that are hoped to describe the same entity. Of course, since fuzzy keyword matching is used in several places in the process, the result will often contain false positives.

A first cut of our algorithm used only the highest-ranking resource description from each source, discarding all others. This has proven to be problematic, as our ranking sometimes would score the document resource description higher than the description of the person or other entity described in the document, because both might have the same, highly salient, label. By including both, we leave the problem to the user, instead of risking the wrong pick.

## 3.6 Source set expansion

If the number of highly-scoring resource descriptions is low at this point, then an attempt is made to discover additional sources, based on the RDF data we have already retrieved and established to likely describe the target entity. We obtain new resource identifiers for the target entity using four methods:

1. If the selected resource descriptions are centered on a URI (not a blank node), then this URI is considered.
2. If the resource descriptions include any `owl:sameAs` links, then the target URIs are considered.
3. If the resource descriptions include any OWL inverse functional properties (IFPs) from a hardcoded list (e.g. `foaf:mbox` and `foaf:homepage`), then a Sindice index search for other resources having the same IFP value is performed. (Resources having the same value for an IFP are considered equivalent under OWL inference rules.)
4. By means of a query to the OKKAM service. OKKAM is an experimental service which assigns names to entities on the web [9]. OKKAM returns resource identifiers along with a confidence value. Any resource identifiers whose confidence value exceed a certain threshold are added to the set of input resource identifiers. We observe that currently the number of entities that are reliably recognized by the OKKAM service is still low, as not many OKKAM ids can be found out in the

[6] http://www.foaf-project.org/

web, so this step will often not produce a result. In the case where it returns results however, it is a high-quality identifier that is likely to contribute relevant results to the next steps.

Any resource identifiers discovered using these methods will be fed into the process at the *Fetching and Parsing* step.

## 3.7 Entity profile creation

All selected resource descriptions are merged into a single entity profile. This simply means that all key-value pairs from all resource descriptions are combined into a single description. A reference to the original source is kept for each value.

## 3.8 Property consolidation and ranking

Often different properties (keys in the key-value pairs that describe the entity) express the same thing. The next step is to consolidate the potentially large and chaotic list of properties into a simpler list that is more meaningful to the user. In RDF, properties are named with URIs; we consider only the last segment ("local part") of the URI. By convention, this local part is usually a good name for the property, written in CamelCase or with underscores or dashes, which are converted back into a more readable string consisting of space-separated words. In the future, we should also check the definition of the property (obtainable by dereferencing its URI, and often already in the page repository) for an `rdfs:label`.

The next step is to treat both incoming triples (of the shape "other-entity - relationship - our-entity") and outgoing triples (of the form "our-entity - relationship - value" or "our-entity - relationship - other-entity") as outgoing triples. This is done simply by flipping the incoming triples around, and adding an *inverse flag* to the relationship. For example, *A creator B* becomes *A is creator of B*.

Next, we apply some simple english-language heuristics on the property names. This is based on observing properties typically used in the wild. The heuristics are:

- remove initial "has" (e.g. in "has title")
- remove initial "holds" (e.g. in "holds role")
- remove initial "gives" (e.g. in "gives presentation")
- remove final "of" and flip property (e.g. in "member of")
- remove surrounding "is ... of" and flip property (e.g. in "is topic of")

Next, we apply a manually-compiled list of approximately 50 preferred terms. For example, we replace all of the following property names with the preferred term "web page": work info homepage, workplace homepage, page, school homepage, weblog, website, public home page, url, web. Special attention has been given to terms that can be used in customized ways in the user interface: labels, depictions (images), short descriptions, web links.

Next, we drop a number of properties that are of little value in an end-user interface, e.g. `foaf:mbox_sha1sum` or `rdfs:seeAlso`.

The ad-hoc list of rules is easily changeable. So far, it has been fine-tuned mostly for producing good results with profiles of people, but we observe that it takes very little effort to extend it into new areas, which seems to be a good investment for any area where there exists a significant amount of data that uses a common set of properties.

After consolidation, properties are ranked. We use a simple ranking metric: the number of sources that have values for the property. This will push generic properties such as "label" and "type" to the top. The number of distinct values for the property is also factored in: Properties where many sources agree on one or a few values (as observable e.g. with a person's name or homepage) receive a boost.

## 3.9 Value labeling

For key-value pairs where the value is not a literal value (such as a name or a date), but a reference to another resource (usually by URI), a best-effort attempt is made to retrieve a good label for the resource:

1. The original source RDF graph in which the resource was found is examined for typical label property, such as `foaf:name` or `dc:title` or `rdfs:label`.
2. If nothing is found, and it is a URI, it will be resolved against the page repository or the web, as described above in *Fetching and Parsing*.
3. If nothing is found, and it is a URI, then the last part of the URI will be used in a manner similar as described above for property names.

A typical entity profile can refer to dozens or hundreds of other entities, so this is an expensive process. Yet it is very important for a decent user experience. We consider showing a quality label much more desirable than showing a URI or some fragment of a URI, and it is practically a requirement to allow a user to make sense of the entity profile. The labels also feed into further ECSSE requests: When a user wants to follow a link to another entity, then the underlying resource identifier(s) as well as the label are used to submit a new ECSSE request in order to produce the linked entity's profile.

To achieve responsiveness despite the large required number of page repository or web requests, the initial profile displayed to the user will only contain labels produced by method 1 and 3 above. The additional labels are retrieved while the user already sees the profile and will be displayed incrementally using AJAX requests.

Another obvious method of reducing processing requirements that we have not yet implemented would be to show only the first N values for each property. If a person has 200 contacts in their FOAF profile, only the first 10 could be shown by default until the user takes action to show the rest.

### 3.10 Value consolidation

If a property has several values with identical or very similar labels, then they are collapsed into one value to improve the visual presentation. For example, several sources that describe a scientist can state that they have authored a certain paper, using different identifiers for the paper. Without label-based consolidation, the paper would appear several times because the identifiers are not the same. After label consolidation, it appears only once. Both identifiers are retained internally. A click on the paper will cause a new ECSSE search that has the label and both of the URIs as input.

Since labels are retrieved and displayed incrementally, the value consolidation has to be performed in the same fashion.

### 3.11 Source list refinement

After the entity profile is presented to the user, they can refine the result by adding or removing sources.

Almost any entity profile initially includes some poor sources that add noise to the results. Mixed into the desired entity profile are other entities that have the same or a similar name, or that for other reason ranked highly in the text search portions. The user interface allows quick removal of these. Widgets for source removal exist in the list of sources, and next to each value that is displayed in the profile. If the profile shows a poor label or unrelated depiction for the entity, a quick click will remove the offending source, and the next-best label or depiction will automatically take its place if present.

Since the profile is based on a fixed number of resources, it will often show only a subset of what is known about the entity. There is a button for including more sources in the source list. It will retrieve more sources from Sindice, run the usual processing, and mix the results into the profile.

We plan to include widgets that facilitate retrieval of more information of a specific kind. For example, if a person's entity profile shows several academic publications that come from sources on a certain domain, then it is likely that fetching more sources from that domain will yield more publications.

Another situation is when the search yields not much useful information about the desired entity, or the useful information is drowned out by unrelated noise. This occurs if the chosen search phrase does not reliably yield information about the desired entity. There is an analogous case in traditional web search: If the results are poor, an experienced search engine user will try slightly different search terms to narrow or broaden or skew the results. We want to enable similar behavior to allow a user to interactively refine the results. At the moment, the user can enter a new search phrase, the entire process will be repeated, and the profile from the new search will be mixed in with the previous results. This is not optimal; it would be better if the new results were skewed towards the set of sources that the user retained from the previous search. This is subject to ongoing experimentation.

### 3.12 Implementation

Figure 1 summarizes the implementation architecture of the ECSSE system. ECSSE is built on top of the Sindice infrastructure and uses the Sindice Index, the Sindice Page Repository, the Sindice Fetcher, and the any23 parser suite. ECSSE submits ping notifications to the publicly available Sindice ping submission API.

The ECSSE processing workflow is implemented partially in a set of Java servlets hosted in a Tomcat application server, and partially client-side in Javascript. Moving parts of the processing to the client side has improved responsiveness of the user interface. The static parts of the user interface are implemented as a Ruby on Rails application.

ECSSE also invokes the public OKKAM service.

## 4. ECSSE IN ACTION

ECSSE, at the time of the writing, is still in a preliminary alpha release: it can be tested at `http://sindice.com/ecsse` but it has not yet been publicly announced.

In this section we will illustrate how ECSSE presents itself to the end user and comment on some interesting specific results. Commenting on specific results is by no mean intended to be an evaluation, which we will perform qualitatively and quantitatively in future works , but gives an idea of the potentiality of the system.

*ECSSE for Giovanni Tummarello*

In case of researcher "Giovanni Tummarello", 14 sources are presented. Interestingly, some of these sources are not RDF native. For example, we can spot among the results Tummarello's public Facebook profile, which provides a microformat adding one more picture to the mash up, along with that provided by other sources such as, for example, the one coming from the DERI institute team page[7]. From the same source come other useful bit of information such as his work phone number, some of the publications and related projects etc.

A number of results presents the OKKAM logo next to them as they have been obtained by querying the OKKAM service for Tummarello's identifier and then using Sindice to locate sources mentioning it. While the result of this aggregation is, partly, shown in Figure **??**.

*ECSSE for Eyal Oren*

In case of researcher "Eyal Oren", ECSSE performs well but includes wrong source (Eyal Podell, whose daughter is called Oren). The user is then presented with the right picture of Eyal Oren, a list of publications coming from different sources as in the previous case but an "'abstract"' description in which the name is different and the person is indicated to be an actor. It is truly straightforward to spot this mistake and, by hovering the mouse on top of the wrong description, to spot the highlighted source where this data comes from. Similarly, the user could have noticed that many sources (11) had "Eyal Oren" as main label, while only one had "Eyal Podell". Hovering on top of the wrong label and pressing the

[7] `http://www.deri.ie/about/team/`

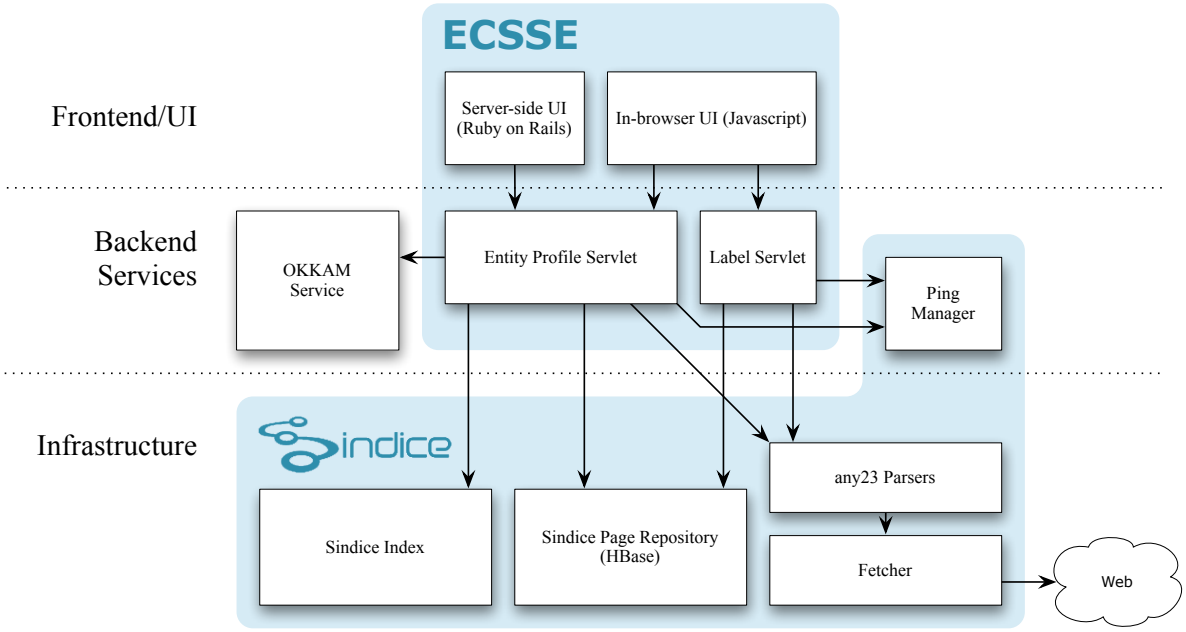


**Figure 1: Implementation architecture of ECSSE**

X button next to the datasource that becomes highlighted is all that it takes to clean the information profile of all the bits of information from that source.

In this case the wrong data source could have been avoided by a better entity matching algorithm. This query, and the consequent workflow, are interesting however as they make clear that the visual inspection of data, even in its mashed up form, can be effective and efficient at spotting wrong data sources.

*ECSSE for Trento*

Querying for the Italian city "Trento" ecsse doesn't return many hits mostly becouse a limited number of semantic data sources are available. We do see, however, data from DBPedia, Geonames, and data coming from several people profile where trento is mentioned as home town.

Possibly the most interesting aspect of this query, however, is how multiple sources that describe formally different entities can be put together in the mash up, resulting in a practically useful unified profile. As an example, the city of Trento is Disjoint from the entity "province of Trento", e.g. in wikipedia. Yet using the two dbpedia entries at the same time in ECSSE (which is suggested by the set of sources automatically returns) yields to results which are meaningful to a human user. For example the final entity profile about trento would contain also an overall map of Italy with the trento province highlighted and the name of many neighboring towns (these pieces of info coming from the province source), which in certain context can be considered relevant pieces of information for the Trento entity.

## 5. DISCUSSION AND CONCLUSIONS

While ECSSE is by no mean the first data aggregator for the Semantic Web, its contribution is to show that exciting possibilities lie in an holistic approach for data discovery and consolidation.

In ECSSE, elements such as large scale semantic web indexing, logic reasoning, data aggregation heuristics, ad hoc ontology consolidation and, last but not least, user interaction and refinement, all play together to provide entity descriptions which become live, embeddable data mash ups.

As a result, preliminary results indicate that ECSSE manages to work surprisingly well, finding and aggregating different sources in a useful way. A query for the name of active semantic web researchers often finds publications from many sites, pictures, social networking and contact info and relate projects.

When ECSSE Automatic selection fails, the user can intuitively recognize this by spotting, for example, a wrong entity type or wrong data statements or by seeing that multiple sources confirm one statement while others come from only one source. When this happens, the user can intuitively eliminate the wrong datasource. Similarly, the possibility of adding new datasources and considering two otherwise separated entities as one is intuitive and practically useful.

The way by which the user can quickly adapt the mash up possibly for his/her intended target goal inspires the general thought that a little semantic might in fact go a long way, *at least* in making it easy for a *human* to do the last step (and validate, by accepting the final list of sources "crystallized" in the permalink).

As a result, we believe that ECSSE style embeddable mash ups could be effective at providing incentives for for pub-


**Figure 2: ECSSE Screenshot for the query "Giovanni Tummarello"**

lishing Semantic Web data: data elements from any web source could potentially be shown on any web page, possibly rewarding the data provider with back link and branding opportunity.

Similarly, user themselves who would use ECSSE mash ups, e.g. to display CV like information on one's homepage, would have a reason to ask for the creation of more semantic data, e.g. by a conference web site about a published paper.

## 6. ACKNOWLEDGMENTS

This work was partially supported by the FP7 EU Large-scale Integrating Project OKKAM - Enabling a Web of Entities (contract no. ICT-215032). For more details, visit `http://www.okkam.org/`.

## 7. REFERENCES

[1] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. Rdfa in xhtml: Syntax and processing. Technical Report http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014, W3C, October 2008.

[2] R. Cyganiak, H. Stenzhorn, R. Delbru, S. Decker1, and G. Tummarello. Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In *Proceedings of the European Semantic Web Conference*, 2008.

[3] R. Delbru, A. Polleres, S. Decker, and G. Tummarello. Context dependent reasoning for semantic documents in sindice. In *In Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2008.

[4] A. Harth, A. Hogan, R. Delbru, J. Umbrich, S. O'Riain, and S. Decker. Swse: Answers before links! In *Semantic Web Challenge, ISWC*, pages –1–1, 2007.

[5] M. Hausenblas, Halb, Raimond, and T. Heath. What is the size of the semantic web? In *I-Semantics 2008: International Conference on Semantic Systems, 2008*, 2008.

[6] A. Hogan, A. Harth, and S. Decker. Performing object consolidation on the semantic web data graph. In *Proceedings of 1st I3: Identity, Identifiers, Identification Workshop*, pages –1–1. I3, 2007.

[7] T. B. Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *In Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06*, page 06, 2006.

[8] P. Mika and G. Tummarello. Web semantics in the clouds. *Intelligent Systems, IEEE*, 23(5):82–87, 2008.

[9] B. B. Paolo Bouquet, Heiko Stoermer. An entity name system (ens) for the semantic web. In *Proceedings of the European Semantic Web Conference*, 2008.

[10] G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the open linked data. pages 552–565. 2008.