Conference Organization

Programme Chairs

Raoul-Sam Daruwala
Cong Yu

Programme Committee

Gustavo Alonso
Srikanta Bedathur
Kevin Chang
Isabel Drost
Ariel Fuxman
Lee Giles
Sharad Goel
Richard Hankins
Jeffrey Korn
Chris Mattmann
Charles McCathieNevile
Peter Mika
Stelios Paparizos
Eugene Shekita
Jimeng Sun
Jian Shuo Wang
Ding Zhou
Aoying Zhou

Local Organization

Table of Contents

Creating Your Own Web-Deployed Street Map Using Open Source Software and Free Data

Christopher Adams
GenieKnows.com
1567 Argyle Street
Halifax, Nova Scotia, Canada
+1 902-431-4847
chris@genieknows.com

Tony Abou-Assaleh
GenieKnows.com
1567 Argyle Street
Halifax, Nova Scotia, Canada
+1 902-431-4847
taa@genieknows.com

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services.*

General Terms

Design.

Keywords

Map, Web-map, Open Source, Geographic Information Systems, Local Search, MapServer, OpenLayers.

1. INTRODUCTION

Street maps are a key element in Web-based geographic information systems. They can be used in business directories, mash-ups (e.g., hiking trails and event locations), geo-social or geo-political applications, and various other geographic based systems including local search. They are important because they make the connection between information and geography, allowing the user to quickly get a frame of reference based on their current knowledge, and learn more about the spatial relationships between locations. Although street maps can be added to websites easily using an API, this is not always desired. We outline the free/open source software and data to use to create customized street maps. In section 2 we discuss related works. Section 3 outlines the system architecture for a Web-deployed street map. We discuss a process for developing a street map in section 4 followed by the conclusion in section 5.

2. RELATED WORK

There are several ways one can add a street map to their Web site. One approach is to use third party APIs which are available to the general public, are quick to deploy and require few resources. However the level of customization is often restricted to simple operations and you may be limited to a certain number of transactions. In the case of Local Search you may also be in competition with the provider of the API which can bring up legal issues. Google, Yahoo!, Microsoft, MapQuest, and GenieKnows[1] all provide such APIs. Another option is to buy a custom solution. A custom solution can come in the form of a service or an out-of-the-box product that you can deploy yourself. However, custom solutions often require one to pay for both the data and the software, and there is often a transactional fee as well. Map 24 and ESRI are two examples of providers of such services. A third option, and the one we discuss in this paper, is to build a street map yourself, using open source software and free data. This approach generally allows for more freedom of customization and the cost is the hardware required to run your map servers and the time it takes to create your solution. Although this approach will take more time than the other options, as long as your data is without restrictions, there are no limits on usage outside of those imposed by the hardware, and in the case of Local Search, you will not necessarily be supporting a competitor.

[1] At the time of writing, GenieKnows offered a map API in private beta.

WWW 2009, April 20-24, 2009, Madrid, Spain.

3. SYSTEM ARCHITECTURE

The mapping solution discussed here conforms to a client-server architecture, as shown in Figure 1. The architecture described consists of the map server, the map client, and the map data. A map server stores the map data and is used to handle requests from a map client. The map server renders portions of the map into images while facilitating customization of the styles used to render the map. Map clients are typically implemented in JavaScript or Flash and run in a Web browser. The map data, as described later in section 4.1, and comes in vector format. Types of vector data include points, lines, and polygons and often have non-geographic information attached. The map client-side software must facilitate complete customization of the appearance of the controls used to navigate the street map.

4. BUILDING A STREET MAP

There are many components in building a street map. These include data and data refinement, server side software, and client side software.

4.1 Map Data

Data comes in many open formats such as Shapefiles, GML, or easy-parsing text based formats. When making your own street map, you have the choice between free data and paid data. The US Census Bureau produces a free product called Tiger/LINE [9] which unlike many paid alternatives does not have a base cost or transactional fee. Tiger/LINE data contains among other things; Roads, water bodies, areas of interest (e.g., parks, hospitals, cemeteries, shopping centres, etc.), points of interest (e.g., landmarks, businesses, churches, etc.), railways, ferry routes, and county subdivisions (city areas). Outside of the USA, OpenStreetMap [5] provides free data for many countries, including European countries. The data is assembled from different sources, including a lot of manual effort, and its correctness is verified by people. The drawback to using OpenStreetMap data is that there are areas where the data is incomplete. Data can also be purchased. Some data providers

include Navteq, and TeleAtlas. Paid data is generally complete, accurate, and includes attributes not given in many free data sources (i.e. one-way streets, and routing hierarchies). However, this data often comes with a large base cost, and a transactional cost depending on the use. For maps of the USA, we suggest the use of Tiger/LINE data. For other parts of the world, OpenStreetMap is a good choice. We use Tiger/LINE data in our example.

4.2 Data Refinement

In many cases the data does not come in a form that is the best for the application. For example a popular method for labeling cities is using a central point, which could be based on geography or some social or political factor. Tiger/LINE has city polygons which can be useful for generating this data. In another example, roads most often come in the form of centre lines but it may be desired to regenerate this data as polygons and add width based on a measurement system not dependant on scale. This can improve the performance of rendering in some cases, and will make size setting and styling of roads much simpler. If you have data from multiple sources with different record structures, which needs to be used in the same manner, it may be desirable to convert it into a single format. The Python programming language has extensions that can be used to make many of these tasks easier and faster to implement. For example, `ShapeLib` [7] has a python extension that can be used to parse shape files, `Polygon` [6] can be used to perform operations on polygons, and the GIS-Python-Plone Laboratory [8] provides a spatial index library, which can help optimize algorithms working on spatial data. Another alternative would be PostgreSQL with PostGIS for such tasks.

4.3 Server/Rendering Software

MapServer [3] is our package of choice for rendering maps. An alternative to MapServer is mapnik [2], which is the rendering engine of choice for OpenStreetMap. MapServer also provides a CGI interface that handles web requests for map renderings. MapServer supports many of data formats, which include raster, vector, and database formats. MapServer can be run on Windows, Linux, and Mac OS X, and provides a scripting interface that can be used for rendering maps in PHP, Python, Perl, Ruby, Java, and .NET. MapServer also supports many coordinate systems (projections). Using the Anti-Grain Geometry library, MapServer can produce high quality renderings, and can output images in formats including bitmaps, and SVG. MapServer is used in many ready-to-use open source application environments. The styles and data sources used by MapServer to render a map are specified using a MapFile. MapFiles are in the form of a script in which you specify a data type (roads, areas of interest, etc.) as a layer. Within each layer, classes can be defined, which contain styles, and are enabled based on expressions that match numeric and string records within the data, as well as the current scale. MapServer also includes useful utility programs. If it is required to merge data of the same type and record layout then `tile4ms` can be used, and the utility program `shptree` can be used to optimize shape files using quad-tree indexes.

4.4 Client-Side Software

The client side software we believe is best for the task at hand is OpenLayers [4]. It is written entirely in JavaScript, is actively developed and supports the majority of Web browsers. OpenLayers is easy to extend and customize. An example of such customization can be seen at GenieKnows [1] (see Figure 2). OpenLayers provides real-time, interactive map navigation, and is generally fast. OpenLayers has support for adding overlays (GML, KML, etc.) as well as markers and pop-up windows. OpenLayers can request map images from various sources including MapServer's CGI Interface. OpenStreetMap employs OpenLayers as its map client.

4.5 Ten Steps to Creating the Map

1. Download and install MapServer [3].
2. Set up your Web server, and install the MapServer CGI script. You should also create a basic map file at this stage to test your installation.
3. Download the Tiger/LINE data [9]. We suggest using an FTP client for this, as the data is split into folders by state and county.
4. Perform data refinement; for example, create road encasements, city centres, or any data refinement that is desired or required for your application.
5. Merge similar data with tile indexes using MapServer's `tile4ms` and add layers to your map file. Most data useful in creating a street map in Tiger/LINE requires merging if more than one county is used.
6. Optimize data using MapServer quad-tree indexes using `shptree`.
7. Create a test page using OpenLayers [4] in a Web page.
8. Customize your MapFile and OpenLayers Layer settings such as scales, projections, styles, etc.
9. Customize OpenLayers by adding buttons, custom controls, overview map, etc.
10. Test and deploy.

Figure 2 presents a screenshot that illustrates a highly customized version of MapServer, OpenLayers, and the Tiger/LINE data as as used by the local search engine GenieKnows. Figure 3 shows a screenshot of a map of Washington, DC that uses vanilla MapServer, OpenLayers, and Tiger/LINE data. This map was created in about six hours.

5. CONCLUSION

We have outlined a solution to creating a Web-deployed street map using open source software (MapServer, OpenLayers), and free data (Tiger/LINE or OpenStreetMap). We have shown by example that creating a fully customized street map solution is not only possible but is a viable alternative to other existing non-free solutions.

6. REFERENCES

[1] GenieKnows. http://www.genieknows.com

[2] mapnik. http://mapnik.org

[3] MapServer. http://mapserver.org

[4] OpenLayers. http://www.openlayers.org

[5] OpenStreetMap. http://www.openstreetmap.org

[6] Polygon. http://polygon.origo.ethz.ch

[7] Shapelib. http://shapelib.maptools.org

[8] The GIS-Python-Plone Laboratory. http://gispython.org

[9] US Census Bureau Tiger/LINE. http://www.census.gov/geo/www/tiger

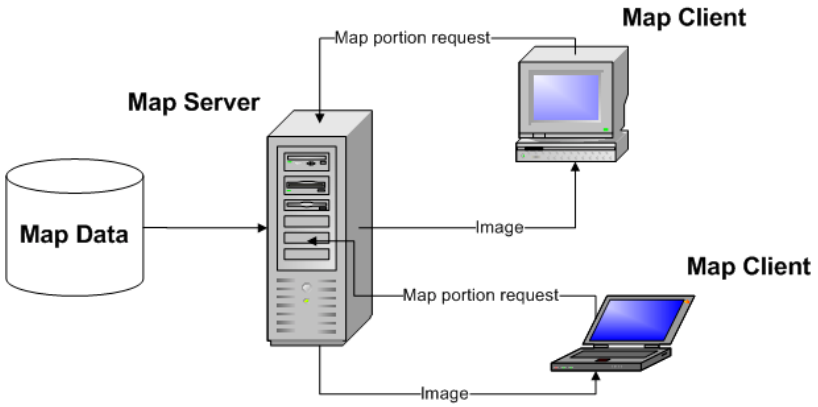


Figure 1. The Architecture of a Web-deployed street map.

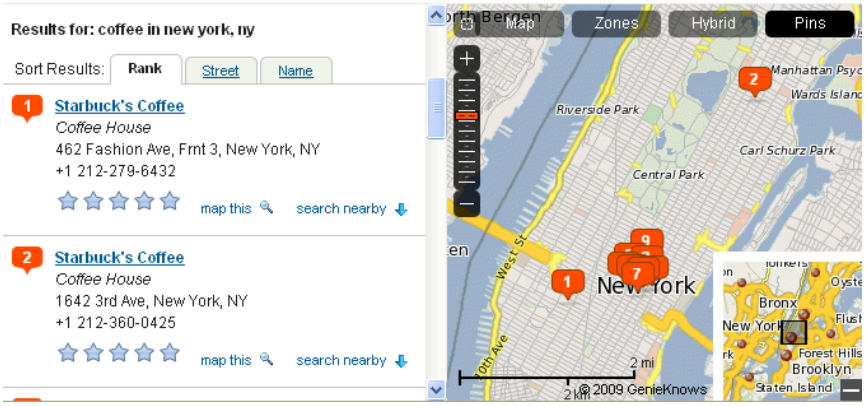


Figure 2. Example of customized street map using MapServer, OpenLayers, and Tiger/LINE data

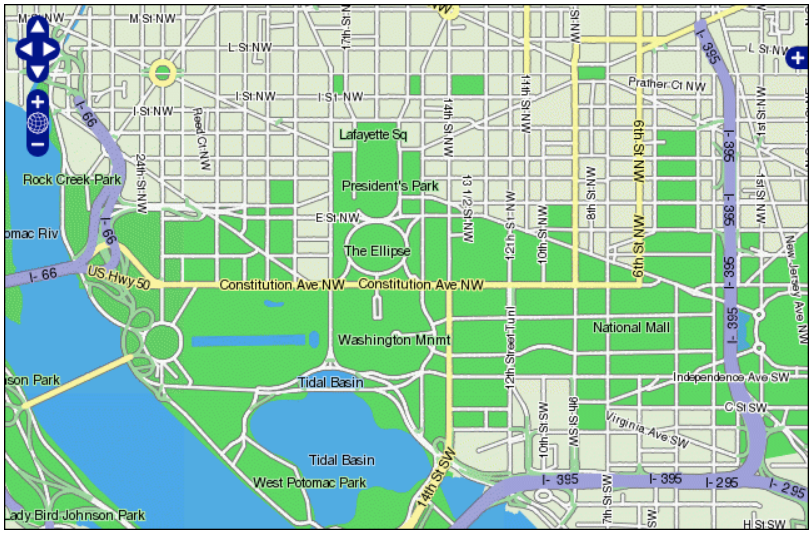


Figure 3. Test map made using MapServer, OpenLayers, and Tiger/LINE data for Washington, DC

Query Portals: Dynamically Generating Portals for Web Search Queries

Sanjay Agrawal, Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti
Arnd Christian König, Dong Xin

Microsoft Research
{sagrawal, kaushik, surajitc, vganti, chrisko, dongxin}@microsoft.com

1. INTRODUCTION

Many informational web search queries seek information about named entities residing in structured databases. The information about entities that a user is seeking could be available from the structured database, or from other web pages. For example, consider an informational query such as [lowlight digital cameras]. A large structured database consisting of products may contain several products that are relevant for the user's query. Surfacing a set of the most relevant products to the user and then enabling her to obtain more information about one or more of them would be very useful. Ideally, we would like a *portal- like* functionality which provides an *overview* of all relevant products, and further allows *drill down* on them.

The need for structured data search is illustrated by the proliferation of vertical search engines for products [3, 1], celebrities [2], etc. Current web search engines already federate queries to one or more structured databases containing information about named entities products, people, movies and locations. Each structured database is searched individually and the relevant structured data items are returned to the web search engine. The search engine gathers the structured search results and displays them along side the web search results. However, this approach does not enable a portal-like functionality due to two key limitations.

Incomplete Results: Current federated search over each structured database is *"silo-ed"* in that it exclusively uses the information in the structured database to find matching entities. That is, the query keywords are matched *only* against the information in the structured database. The results from the structured database search are therefore independent of the results from web search. We refer to this type of structured data search as *silo-ed search*.

While the silo-ed search works well for some queries, it would return *incomplete or even empty results* for a broad class of queries. Consider the query [lowlight digital cameras] against a product database containing the name, description, and technical specifications for each product. `Canon EOS Digital Rebel Xti` may be a relevant product but the query keyword {lowlight} may not occur in its name, description or technical specifications. Silo-ed search over the above product database would fail to return this relevant product. Reviews of the product may describe it using those keywords and can help deduce that the product is relevant to the query. However, the structured database may not contain the comprehensive set of reviews of each product necessary to identify the relevant products. Hence, a silo-ed search against the structured database may miss very relevant results [5]. In fact, current search engines which adopt such a silo-ed search approach over product databases do not return any product entity for our example query.

Inadequate Information for Drill Down: When entities in a structured database are found to be relevant for a user's query, current approaches return *only* information about the entity that is available within the database. The information in the database might be inadequate. Often, the user's need might be better served by information on the web. For example, consider the product `Canon EOS Digital Rebel Xti`. The database might have information about the technical specifications, price, and may be the manual for this product. However, a user might also be interested in reviews, device drivers which are available on the web. Providing links to that information would satisfy her information requirement.

In this paper, we propose the *Query Portals* technology to address the above two limitations by (i) *leveraging web search results* to identify entities in structured databases relevant for informational queries, and (ii) enabling users to *drill down* and obtain specific information from the web on any of these entities. Thus, we attempt to create a dynamic *portal-like* functionality by providing an overview with a set of entities relevant to a web search query, and then allowing users to drill down into one or more of these entities.

We now briefly discuss the intuition behind our approach for addressing the two limitations discussed earlier.

Addressing the Limitation of Incomplete Results: Our main insight for addressing this limitation is to leverage web search results [5]. Our approach is to first establish the *relationships* between web documents and the entities in structured databases. Subsequently, we leverage the top web search results and the relationships between the result documents and the entities to identify the most relevant entities. Consider our example query [lowlight digital cameras]. Several web documents from product review sites, blogs and discussion forums may mention the relevant products in the context of the query keywords {lowlight, digital, cameras}. Therefore, the documents returned by a web search engine are also likely to mention products that are relevant to the user query. We identify the relevant products using the documents returned by a web search engine. Since we leverage web search results, our approach can return entity results for a much wider range of queries compared to silo-ed search [5].

A screenshot of the set of relevant product entities, e.g.

WWW2009, April 20-24, 2009, Madrid, Spain.
.

Figure 1: Related product entities

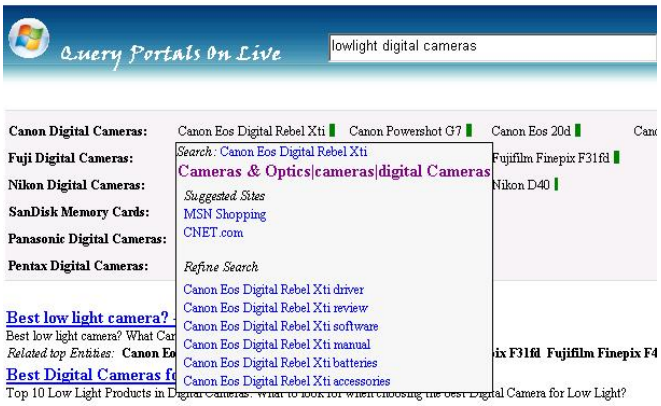


Figure 2: Information links for an entity

`Canon EOS Digital Rebel Xti`, returned by our Query Portals system for the query [lowlight digital cameras] is shown in Figure 1. The set of relevant entities (grouped by the brand name—Canon, Fuji, Nikon, etc.—in this particular example) provides the user with a quick overview of the product entities related to her query.

Overcoming Information Inadequacy for Drill Down: After looking at the set of all relevant entities, the user may want to obtain more entity-specific information, which may not be available in the structured database. Our approach for addressing this limitation is to exploit the content on the web and the web search engines. Specifically, we consider two ways of enabling users to obtain more information on the web about an entity. First, when available, we suggest *authoritative* web sites where a user can find extensive information about an entity. Second, we surface *focused* web search queries per entity to enable a user obtain very specific information on the web about an entity. We refer to the union of authoritative web sites and focused web search queries for a specific entity as *entity information links*. We rely on the query logs, category information about entities, and the web document snapshot in order to automatically identify information links per entity.

The entity information links we show for the example entity `Canon EOS Digital Rebel Xti` is illustrated in Figure 2. We suggest authoritative web sites such as `CNet.com`, or comparison shopping sites such as `MSN Shopping`. We also suggest focused web search queries (such as [Canon EOS Digital Rebel Xti reviews]) to enable a user obtain reviews, batteries, accessories, drivers, RAM memory for this product. Depending on the user's information requirement, she may choose one or more of these suggestions. Note that our approach is complementary to the information about an entity already available in a structured database.

In summary, the Query portals system presents a user with an overview of the entities (along with web search results) relevant to her query and further enables her to obtain specific information about any of the entities.

2. ARCHITECTURE

We now describe the architecture of the Query Portals system. As shown in Figure 3, the system has pre-processing and query-time processing components. The pre-processing components prepare the Query Portals system so that while processing a web search query, the query-time processing components can efficiently identify relevant entities and information links per entity.

The query portals system has two pre-processing components: *entity extraction* and *entity information links identification* components. We first briefly discuss these two components.

Entity Extraction: The first *entity extraction* component takes a collection of web documents as input along with the entity database and outputs a relation consisting for each URL in the web snapshot mentions of entities in the given entity database. The entity extraction component analyzes each web document in the collection and outputs a list of [url, entity name, id, position in document] tuples. We refer to this relation as the *URL-EntityList* relation.

In general, our system can build upon any entity extraction technology [6]. We now sketch the approach adopted in the Query Portals system. Our observation is that the entity database defines the universe of entities we need to extract from web documents. For any entity not in the entity database, we cannot provide the rich experience to a user as the entity is not associated with any additional information. Such entities would therefore not be returned as results. We therefore constrain the set of entities that need to be identified in a web document to be from the given entity database. By doing so, we can avoid the additional effort and time spent by current entity extraction techniques to extract entities not in the target entity database. We leverage this entity database membership constraint to develop techniques (i) which can handle a *wide variety of structured data domains*, and (ii) which are also significantly more *efficient* than traditional entity extraction techniques. The task now is to analyze document sub-strings which match with an entity name in the given database [4].

However, in most realistic scenarios, say for extracting product names, expecting that a sub-string in a web document matches exactly with an entry in a structured database table is very limiting. For example, consider the product entity `Canon EOS Digital Rebel Xti`. In many documents, users may just refer to this product by writing `Canon XTI`. Insisting that sub-strings in documents match exactly with entity names in the reference table may likely cause these product mentions to be not extracted. Therefore, it is very important to consider *approximate matches* between document sub-strings and entity names in a reference table [10, 8]. The approximate matching technology we use in the Query Portals system is described in [9, 8].

Another issue is that of pruning out document sub-strings which match an entity name in the database without intending to refer to the entity. For example, the distinction between the movie "60 seconds" versus a phrase "60 seconds" (in reference to time) is important while extracting a set of movies. We apply known techniques for the entity recognition step [10, 11].

Entity Information Links Identification: The second component identifies authoritative web sites and focused web search queries for each entity in the given structured database. The output of this component is the *Entity-InformationLinks* relation. The information links for an entity consists of a set of authoritative web sites which provide detailed information for the entity, and a set of focused web search queries which may obtain informative web page results about more specific aspects of the entity.

We rely on entity attribute and category information, domain knowledge, query logs, and web document snapshot to identify information links per entity. Due to space constraints, we skip details of the specific techniques. The basic intuition behind our techniques is as follows. If for a number of entities within a certain category, many users are issuing queries of the form [e X], then we hypothesize that for every entity e in the category X is important. In this paper, we focus on X being either a web site domain (such as `CNet` or `MSN shopping`) or a keyword (such as `batteries` or `software manuals`). We also apply other processing (such as stop word removal, stemming, removing approximate duplicates, and a few domain-specific filters) over the web site domains and keywords to ensure that the resulting suggestions are robust and accurate. We use the web site domains as authoritative web sites and the keywords to generate focused web search queries for all entities in the category.

We now briefly discuss the three query-time components.

Entity Retrieval: The task of entity retrieval is to lookup the URL-EntityList relation (materialized by the Entity Extraction pre-processing component) for each of the URLs in the top results from a web search engine for the user's query, and retrieve the set of entities mentioned in the document along with their positions. To enable efficient retrieval of entities per URL, we store the URL-EntityList relation in a database and index it on the URL column.

Entity Aggregation and Ranking: This component ranks the set of all entities returned by the entity retrieval component. We rely on a custom scoring function which takes into account several features for each entity: the number of times the entity is mentioned, the ranks assigned by the web search engine to the documents mentioning the entity, the positions of the entity mentions within the document, the category to which the entity belongs. We only select the entities whose scores are above a pre-determined threshold, and rank them in the descending order of their scores. In general, other scoring functions or rankers based on machine learning techniques may be used in this context, provided we have the required training data [5, 7]. We will investigate such alternative techniques in future.

Information Link Retrieval: The task of this component is to lookup the Entity-InformationLinks relation (materialized by the Entity Information Links Identification preprocessing component) to retrieve the information links for each relevant entity. To enable efficient retrieval of the information links per entity, we store the Entity-InformationLinks relation in a database and index it on the entity id column.

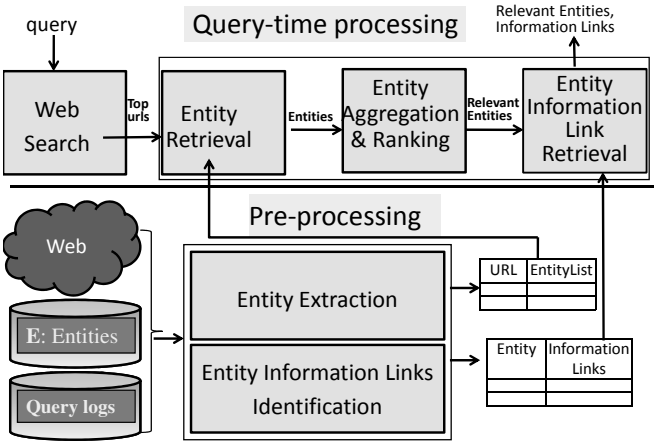


Figure 3: Query Portals Architecture

We then display the entities by grouping them on entity type (people or products) and an interesting attribute of the entity, say, the brand name as shown in Figure 1. Currently, the attributes on which we group relevant entities by are determined upfront per entity category.

3. SUMMARY

The Query Portals system dynamically creates a portal like functionality by creating an overview of all entities relevant to a given query, and then enabling users to drill down and obtain information from the web on specific aspects of an entity. We address the two key limitations of current vertical search engines which *only* search and surface information about entities in a structured database. We address these limitations by establishing the connections between web documents and entities in the given database. We effectively leverage these connections along with a web search engine to achieve the portal like functionality.

4. REFERENCES

[1] http://search.live.com/products/.

[2] http://search.live.com/xrank?form=xrank1.

[3] http://www.google.com/products.

[4] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable ad-hoc entity extraction from text collections. In *VLDB*, 2008.

[5] S. Agrawal, K. Chakrabarti, S. Chaudhuri, V. Ganti, C. König, and D. Xin. Exploiting web search engines to search structured databases. In *WWW Conference*, 2009.

[6] D. E. Appelt and D. Israel. Introduction to Information Extraction Technology. *IJCAI-99 Tutorial*, 1999.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank using Gradient Descent. In *ICML*, 2005.

[8] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin. An efficient filter for approximate membership checking. In *ACM SIGMOD Conference*, 2008.

[9] S. Chaudhuri, V. Ganti, and D. Xin. Exploiting web search to generate synonyms for entities. In *WWW Conference*, 2009.

[10] W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *ACM SIGKDD Conference*, 2004.

[11] V. Ganti, A. C. König, and R. Vernica. Entity Categorization over Large Document Collections. In *ACM SIGKDD*, 2008.

A Semantic Web Ready Service Language for Large-Scale Earth Science Archives

Peter Baumann
Jacobs University Bremen, Germany Campus Ring 12 Bremen, Germany
p.baumann@jacobs-university.de

ABSTRACT

The discussion about Semantic Web ready services usually concentrates on alphanumeric information. In this contribution we consider semantic services on raster data instead, focusing on geo-spatial application domains.

Concretely, the Open GeoSpatial Consortium (OGC) Web Coverage Processing Service (WCPS) Interface Standard defines a raster retrieval language suitable for ad-hoc navigation, extraction, and analysis of large, multi-dimensional data sets. Due to its formalized semantics definition it is suitable for reasoning and automatic service chaining. We present the language concept, also considering discoverability, chaining, declarativeness, safe evaluation, and optimizability. Further, implementation details are briefly addressed.

Categories and Subject Descriptors

H.5.3 [**Information Systems**]: Information Interfaces and Presentation—*Group and Organization Interfaces*

General Terms

Languages, Standardization

Keywords

multi-dimensional raster, geo service standards, OGC, WCPS

1. INTRODUCTION

Geo data traditionally are categorized into vector, raster, and meta data. The latter category receives plentiful attention by the Web research community, as it can be addressed by common Semantic Web techniques. Vector data are considered to some extent. Raster data, however, although contributing the largest data volumes, are neglected hitherto.

Recently, Earth sciences more and more strive for integrated services for uniform, cross-discpline access to such data. For 2-D maps this is common already, with many services being in operation or under construction; for further dimensions, such as 1-D sensor time series, 3-D x/y/t time series, 3-D x/y/z geophysical data, and 4-D x/y/z/t oceanographic or atmospheric data, usually bespoke extract-and-download interfaces are deployed where the common denominator is a file format like NetCDF[1]. Notably data sizes frequently amount to multi-Terabyte volumes for single objects, soon Petabytes. In the end, services on the level of file access lack qualities like automatic contents discovery, service chaining, as well as flexible retrieval and analysis.

The Open GeoSpatial Consortium (OGC)[2] is the main driving force in establishing standards for open, interoperable geo services, in close collaboration with other standardization bodies like ISO, OASIS, and the W3C. Among its modular family of standards is the Web Coverage Service (WCS) standard for open, interoperable access to large-scale multi-dimensional geo raster data [9], in OGC speak "coverages".

Several requests have been brought forward over time to extend WCS with some particular processing functionality. The WCS group decided to keep WCS simple and, instead of adding a plethora of unrelated functions, offer a request language which allows expressing an open-ended set of sensor, imaging, and statistical operations. The result is the OGC Web Coverage Processing Service (WCPS) [1]. It shares its conceptual coverage model with WCS and adds a coverage expression language for server-side extraction, processing, and analysis. In its design, several conflicting requirements had to be observed:

- readers without background and inclination for a formal treatment must feel comfortable with it;
- to achieve interoperability the specification must be sufficiently concise and complete to obtain an unambiguous description;
- the resulting syntax should be convenient to read for humans and easy to parse for machine consumption;
- the language concept must be prepared for automatic discovery and service chaining;
- the language must be declarative, optimizable, and safe in evaluation;
- capabilities and resources required by a particular query must be detectable easily;
- the concept should comply with "the usual way of doing it" and avoid cumbersome particularities.

The standard documents and XML schemas can be downloaded from www.opengeospatial.org/standards/wcps. A

WWW2009, April 20-24, 2009, Madrid, Spain.

[1] available from www.unidata.ucar.edu/packages/netcdf/
[2] www.opengeospatial.org

WCPS demo website showcasing 1-D to 4-D retrieval use cases is under construction at www.earthlook.org; additionally, a demo video is available on www.earthlook.org/videos/.

In this contribution the WCPS language concept, design rationales, and features like discoverability, chaining, declarativeness, safe evaluation, and optimizability are presented. Further, the reference implementation stack, which will be released in open source, is detailed. The author is co-chair of the WCS Working Group and the Coverages Working Group; he has developed the WCPS specification and is principal architect of the WCPS reference implementation.

2. CONCEPTUAL MODEL

Like all OGC standards, WCPS is crafted as a Web service offering initialization, information, and action requests. A client initially submits a `GetCapabilities` request to the server which responds with a summary of its capabilities and a list of the coverages served. For each such coverage, a `DescribeCoverage` request will produce further details, such as spatial reference system, bounding box coordinates, cell type[3]. Finally, a `ProcessCoverages` request ships a processing expression to the server, which responds with either a list of coverages, or a list of scalars, or a URL under which the result can be downloaded subsequently.

ISO 19123 [5] defines a coverage as "a function from a spatio-temporal domain to an attribute domain". This is adopted by WCS and likewise WCPS. Simplified, a coverage consists of a multi-dimensional array, a locally unique title (i.e., identifier), a bounding box (its *domain*), one or more coordinate reference systems (one of which must be an *Image CRS* allowing to address pixels by integer array coordinates), and the array cell type (its *range type*), which is a list of named atomic components. For each range component its null values can be indicated, plus the interpolation methods applicable to this component.

A WCPS expression iterates over lists of such server-side stored coverages; several lists can be indicated emulating "nested loops". A variable is bound to each element inspected. In each iteration an optional filter predicate is applied which can suppress this item in the result list. The result of each iteration is generated according to a processing clause using the coverage variables. Among the processing operators available are

- *induced operations* which apply an operation available on the cell type simultaneously to all cell values; these include unary and binary arithmetic, exponential, trigonometric, and boolean operations;
- *spatio-temporal subsetting operations*, in particular cutout (*trim*) and slicing;
- *condensers* allow to derive summary information (such as *count, min, max, some, all*);
- a *general array constructor* to derive completely new arrays (such as histograms);
- *reprojection*, i.e., transformation of the coverage into another coordinate reference system;
- auxiliary functions, such as metadata extraction.

[3]We collectively refer to pixel, voxel, etc. as *cell*.

The following request assumes three coverage objects `A`, `B`, and `C` on the server. These coverages are inspected in sequence, binding it to variable `$c`. Coverages are only considered for the result set if the coverage's `red` maximum exceeds threshold `127`. For the coverages selected, their `red` and `nir` (near infra-red) components are added pixelwise. The result image is encoded in TIFF and shipped back to the client together with accompanying metadata:

```
for $c in ( A, B, C )
where  max( $c.red ) > 127
return encode( $c.red + $c.nir, "tiff" )
```

This encoding-independent *Abstract Syntax* of WCPS is accompanied by separate protocols which define HTTP GET / KVP and POST / XML bindings.

The expressive power of this language allows to perform a range of statistics, imaging, and signal processing operations, such as aggregation/roll-up, modes, convolutions and filter kernels. Recursion tentatively has been omitted to obtain a language which is "safe in evaluation"; this term, stemming from database technology, indicates that any request submitted is guaranteed to terminate in finite time. This is not a real limitation - the goal is not to establish image processing functionality in general, rather to allow for flexible server-side data extraction, which foremost means data reduction. Advanced image processing may well be situated on client side to perform analysis of the data set, possibly reduced from a Terabyte to a Gigabyte, on the user's local machine.

3. SEMANTIC WEB FACETS

Based on the sketch of WCPS presented we briefly reflect on core properties relevant for high-level raster semantics. *Discoverability* is achieved through a mechanism generic to all OGC services. A *GetCapabilities* request, which usually is issued first in a client-server interaction sequence, informs about the request types the server supports plus the coverages (i.e., raster objects) offered by this server. This allows for automatic harvesting of capabilities from WCPS servers. *Safe evaluation*, a concept developed in the database domain, indicates that requests are guaranteed to terminate after finite time. This can be proven based on the semantics definition. *Declarativeness* denotes that a language does not prescribe what operations the server should execute in sequence, but rather describes the result. For example, the SQL database language does not foresee explicit iteration over table rows, but describes the intended result in a set-oriented manner. Similarly, WCPS does not include loops for iterating over coverage cells, but guides users to operation formulations which leave open the array cell inspection sequence. This forms the basis for *optimizability*, which has been studied in depth with the conceptually similar rasql language [6]; both logical optimization (i.e., algebraic rewriting) [7] and physical optimization (such as preaggregation [3], just-in-time compilation [4], and GPU code generation [8]) have been investigated. *Automatic chaining* is possible due to the machine-readable semantics of a processing request. Queries can be rephrased – in particular: split and distributed – based on syntactic criteria. It is one of our current research topics to achieve optimal orchestration within clouds with given node properties (such as data location, computing power, capabilities, etc.) and given networks performance.

4. IMPLEMENTATION

The WCPS reference implementation stack (Figure 1) consists of a standard WCS service which additionally implements the *ProcessCoverages* request type. Both the syntax above (using an ANTLR parser) and XML (using a Xerces parser) are accepted.

On server side, a Java servlet handles incoming requests, resolves the geo semantics (such as different coordinate reference systems) and transforms them into rasql requests [6] which then are passed on to the rasdaman array server, the actual workhorse. Results obtained are MIME-encoded and shipped back to the client. Rasdaman is a multi-dimensional array DBMS, implemented in C++, that is capable of storing and querying raster data of freely definable size, dimension, and cell type [2]. Array data are partitioned (*tiled*) into BLOBs ("Binary Large Objects") of a few Megabytes each and stored inside some relational DBMS. A simple driver interface allows to attach to virtually any DBMS, in case of earthlook.org the PostgreSQL open-source DBMS. The WCPS component itself additionally maintains several tables for the metadata of the coverages served.

Several client interfaces are under development. The WCPS GUI employs a visual programming paradigm (Figure 2) which allows to directly create, fold, unfold, and edit a parse tree based on meta-information automatically retrieved from the server. The final query can be submitted or saved in Abstract Syntax or XML. Further, collaborative research has started aiming at developing a toolkit for rapid deployment of bespoke clients, such as 1-D time series display, 2-D map navigation, and 3-D data cubes.

5. SUMMARY AND OUTLOOK

WCPS has been adopted as OGC standard in December 2008. The reference implementation is done to a large extent; it will be published in open source upon completion. Currently, several projects are being launched to showcase and evaluate WCPS in operational services. Research threads include cost-based query optimization based on dynamic pre-aggregation; automatic resource-aware request distribution in a dynamic cloud; studies of how to design specifications in way to automatically derive conformance test while maintaining understandability (i.e., limiting use of formal language); extending WCPS to general meshes.

6. REFERENCES

[1] P. Baumann, editor. *Web Coverage Processing Service (WCPS) Language Interface Specification*. OGC 08-068r1, 2008.

[2] P. Baumann. Large-scale raster services: A case for databases (invited keynote). In *3rd Intl Workshop on Conceptual Modeling for Geographic Information Systems (CoMoGIS)*, volume LNCS 4231, pages 75 – 84. Springer, 6 - 9 November 2006.

[3] A. Garcia-Gutierrez. Applying olap pre-aggregation techniques to speed up query response times in raster image databases. In *Proc. 2nd Intl. Conf. on Software & Data Technologies (ICSOFT)*, volume ISDM / EHST / DC, pages 259–266, Barcelona, Spain, July 2007.

[4] C. Jucovschi, P. Baumann, and S. Stancu-Mara. Speeding up array query processing by just-in-time compilation. In *IEEE Intl Workshop on Spatial and Spatiotemporal Data Mining (SSTDM-08)*, 15 December 2008.

[5] n.n. *Geographic Information - Coverage Geometry and Functions*. Number 19123:2005. ISO, 2005.

[6] n.n. *rasdaman query language guide*. rasdaman GmbH, 7.0 edition, 2008.

[7] R. Ritsch. *Optimization and Evaluation of Array Queries in Database Management Systems*. Phd thesis, Technical University Munich, 2002.

[8] S. Stancu-Mara. *Using Graphic Cards for Accelerating Raster Database Query Processing*. Bachelor thesis, Jacobs University Bremen, 2008.

[9] A. Whiteside and J. Evans, editors. *Web Coverage Service (WCS) Implementation Specification*. OGC 07-067r5, 2008.

7. FIGURES

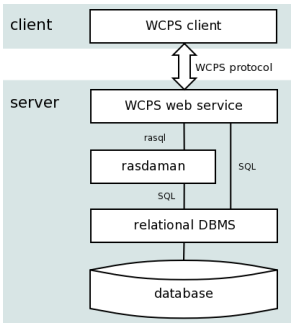


Figure 1: WCPS reference implementation stack

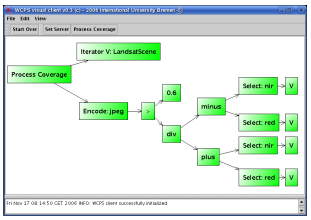


Figure 2: WCPS visual client with NDVI query $(c.nir - c.red)/(c.nir + c.red) > 0.6$ loaded.

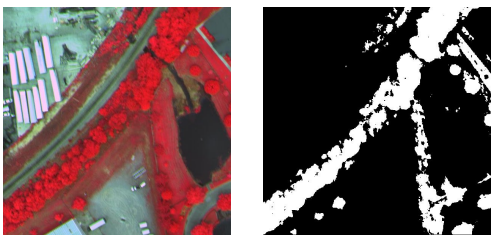


Figure 3: Band recomposition and NDVI as query results. Left: false color image, with nir/red/green bands mapped to RGB; right: NDVI thresholded to 0.6, resulting in a boolean vegetation map.

Silk – A Link Discovery Framework for the Web of Data

Christian Bizer
Freie Universität Berlin
Web-based Systems Group
Garystr. 21
D-14195 Berlin
chris@bizer.de

Julius Volz
Chemnitz University of Technology
Straße der Nationen 62
D-09107 Chemnitz
volz@hrz.tu-chemnitz.de

Georgi Kobilarov
Freie Universität Berlin
Web-based Systems Group
Garystr. 21
D-14195 Berlin
georgi.kobilarov@fu-berlin.de

Martin Gaedke
Chemnitz University of Technology
Straße der Nationen 62
D-09107 Chemnitz
gaedke@cs.tu-chemnitz.de

ABSTRACT
The Web of Data is built upon two simple ideas: Employ the RDF data model to publish structured data on the Web and to set explicit RDF links between entities within different data sources. This paper presents the Silk – Link Discovery Framework, a tool for finding relationships between entities within different data sources. Data publishers can use Silk to set RDF links from their data sources to other data sources on the Web. Silk features a declarative language for specifying which types of RDF links should be discovered between data sources as well as which conditions entities must fulfill in order to be interlinked. Link conditions may be based on various similarity metrics and can take the graph around entities into account, which is addressed using a path-based selector language. Silk accesses data sources over the SPARQL protocol and can thus be used without having to replicate datasets locally.

Categories and Subject Descriptors
H.2.3 [Database Management]: Languages

General Terms
Measurement, Languages

Keywords
Linked data, link discovery, record linkage, similarity, RDF

1. INTRODUCTION
The Web of Data has grown significantly over the last two years and has started to span data sources from a wide range of domains such as geographic information, people, companies, music, life-science data, books, and scientific publications.

While there are more and more tools available for publishing Linked Data on the Web [1], there is still a lack of tools that support data publishers in setting RDF links to other data sources on the Web. The *Silk - Link Discovery Framework* contributes to filling this gap. Using the declarative *Silk - Link Specification Language* (Silk-LSL), data publishers can specify which types of RDF links should be discovered between data sources as well as which conditions data items must fulfill in order to be interlinked. These link conditions can apply different similarity metrics to multiple properties of an entity or related entities which are addressed using a path-based selector language. The resulting similarity scores can be weighted and combined using various similarity aggregation functions. Silk accesses data sources via the SPARQL protocol and can thus be used to discover links between local and remote data sources.

WWW 2009, April 20-24, 2009, Madrid, Spain.

The main features of the Silk framework are:

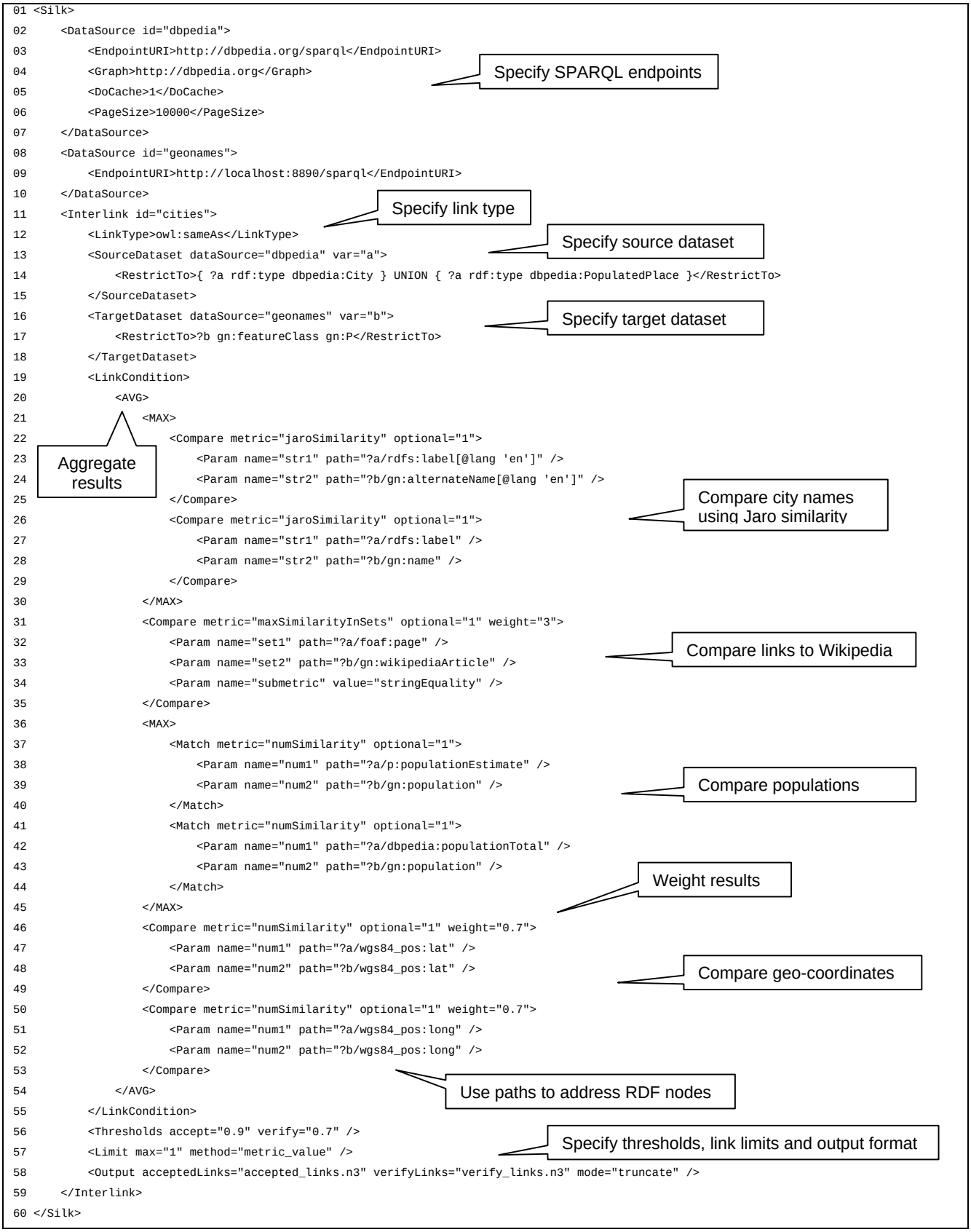
- it supports the generation of owl:sameAs links as well as other types of RDF links.
- it provides a flexible, declarative language for specifying link conditions.
- it can be employed in distributed environments without having to replicate datasets locally.
- it can be used in situations where terms from different vocabularies are mixed and where no consistent RDFS or OWL schemata exist.
- it implements various caching, indexing and entity pre-selection methods to increase performance and reduce network load.

2. LINK SPECIFICATION LANGUAGE
The *Silk - Link Specification Language* (Silk-LSL) is used to express heuristics for deciding whether a semantic relationship exists between two entities. The language is also used to specify the access parameters for the involved data sources, and to configure the caching, indexing and preselection features of the framework. Link conditions can use different aggregation functions to combine similarity scores. These aggregation functions as well as the implemented similarity metrics and value transformation functions were chosen by abstracting from the link heuristics that were used to establish links between different data sources in the Linking Open Data cloud.

Figure 1 contains a complete Silk-LSL example. In this particular use case, we want to discover `owl:SameAs` links between the URIs that are used by DBpedia and by GeoNames to identify cities. In line 12 of the link specification, we thus configure the `<LinkType>` to be `owl:sameAs`.

2.1 Data Access
For accessing the source and target datasources, we first configure the DBpedia and GeoNames SPARQL endpoints using the `<DataSource>` directive. This directive allows the specification of various access parameters, including the SPARQL endpoint URI, the graph name and the enabling of query caching. Furthermore, it is possible to set result page sizes and query pause times to decrease the load on public SPARQL servers. Lines 2 to 7 within the example show how the access parameters for the DBpedia datasource are set to select only resources from the named graph `http://dbpedia.org`, enable caching and limit the page size to 10,000 results per query.

The configured data sources are later referenced in the `<SourceDataset>` and `<TargetDataset>` clauses of the "cities" link specification. Since we only want to match cities, we restrict the sets of examined resources to instances of the classes `dbpedia:City` and `dbpedia:PopulatedPlace` and the




```
01 <Silk>
02     <DataSource id="dbpedia">
03         <EndpointURI>http://dbpedia.org/sparql</EndpointURI>
04         <Graph>http://dbpedia.org</Graph>
05         <DoCache>1</DoCache>
06         <PageSize>10000</PageSize>
07     </DataSource>
08     <DataSource id="geonames">
09         <EndpointURI>http://localhost:8890/sparql</EndpointURI>
10     </DataSource>
11     <Interlink id="cities">
12         <LinkType>owl:sameAs</LinkType>
13         <SourceDataset dataSource="dbpedia" var="a">
14             <RestrictTo>{ ?a rdf:type dbpedia:City } UNION { ?a rdf:type dbpedia:PopulatedPlace }</RestrictTo>
15         </SourceDataset>
16         <TargetDataset dataSource="geonames" var="b">
17             <RestrictTo>?b gn:featureClass gn:P</RestrictTo>
18         </TargetDataset>
19         <LinkCondition>
20             <AVG>
21                 <MAX>
22                     <Compare metric="jaroSimilarity" optional="1">
23                         <Param name="str1" path="?a/rdfs:label[@lang 'en']" />
24                         <Param name="str2" path="?b/gn:alternateName[@lang 'en']" />
25                     </Compare>
26                     <Compare metric="jaroSimilarity" optional="1">
27                         <Param name="str1" path="?a/rdfs:label" />
28                         <Param name="str2" path="?b/gn:name" />
29                     </Compare>
30                 </MAX>
31                 <Compare metric="maxSimilarityInSets" optional="1" weight="3">
32                     <Param name="set1" path="?a/foaf:page" />
33                     <Param name="set2" path="?b/gn:wikipediaArticle" />
34                     <Param name="submetric" value="stringEquality" />
35                 </Compare>
36                 <MAX>
37                     <Match metric="numSimilarity" optional="1">
38                         <Param name="num1" path="?a/p:populationEstimate" />
39                         <Param name="num2" path="?b/gn:population" />
40                     </Match>
41                     <Match metric="numSimilarity" optional="1">
42                         <Param name="num1" path="?a/dbpedia:populationTotal" />
43                         <Param name="num2" path="?b/gn:population" />
44                     </Match>
45                 </MAX>
46                 <Compare metric="numSimilarity" optional="1" weight="0.7">
47                     <Param name="num1" path="?a/wgs84_pos:lat" />
48                     <Param name="num2" path="?b/wgs84_pos:lat" />
49                 </Compare>
50                 <Compare metric="numSimilarity" optional="1" weight="0.7">
51                     <Param name="num1" path="?a/wgs84_pos:long" />
52                     <Param name="num2" path="?b/wgs84_pos:long" />
53                 </Compare>
54             </AVG>
55         </LinkCondition>
56         <Thresholds accept="0.9" verify="0.7" />
57         <Limit max="1" method="metric_value" />
58         <Output acceptedLinks="accepted_links.n3" verifyLinks="verify_links.n3" mode="truncate" />
59     </Interlink>
60 </Silk>
```

Figure 1. Interlinking cities in DBpedia and GeoNames.

GeoNames feature class `gn:P` by supplying SPARQL conditions within the `<RestrictTo>` directives in lines 14 and 17.

2.2 Link Conditions

The `<LinkCondition>` section is the heart of a Silk link specification and defines how similarity metrics are combined in order to calculate a total similarity value for an entity pair.

For comparing property values or sets of entities, Silk provides a number of builtin similarity metrics. The implemented metrics include string-, numeric-, date-, URI-, and set comparison methods as well as a taxonomic matcher that calculates the semantic distance between two concepts within a concept hierarchy. Each metric evaluates to a similarity value between 0 or 1, with higher values indicating a greater similarity. The individual similarity metrics may be aggregated hierarchically using weighted averages, Euclidian distances and weighted products or by choosing the maximum or minimum of a set of metrics.

In the `<LinkCondition>` section of the example (lines 19 to 55), we compute similarity values for the the labels, Wikipedia links, population counts and geographic coordinates of cities between datasets and calculate a weighted average of these values. In cases where alternating properties refer to an equivalent feature (such as `dbpedia:populationEstimate` and `dbpedia:populationTotal`), we choose to perform comparisons for both properties and select the best evaluation by using the `<MAX>` aggregation operator. Weighting of results is used within the metrics comparing the geographical coordinates (lines 46 and 50), with the longitude and latitude similarity weights lowered to 0.7 each.

After specifying the link condition, we finally specify within the `<Thresholds>` clause that resource pairs with a similarity score above 0.9 are to be interlinked, whereas pairs between 0.7 and 0.9 should be written to a separate output file and be reviewed by an expert. The `<Limit>` clause is used to limit the number of outgoing links from a particular entity within the source data set. Only a specified number of best-rated links are kept. In this example, we permit only one outgoing `owl:sameAs` link from each resource.

Discovered links are outputted either as simple RDF triples or in reified form together with their creation date, and confidence score.

2.3 Silk Selector Language

To take the RDF graph around a resource into account, Silk uses a simple RDF path selector language for providing parameter values to similarity metrics and transformation functions. A Silk selector language path starts with a variable referring to an RDF resource and may then use one of several operators to navigate the graph surrounding this resource. To simply access a particular property of a resource, the forward operator (/) may be used, while a backward operator (\ ) allows the navigation along a property edge in the reverse direction. For example, the path "`?artist/rdfs:label`" would select the set of label values associated with an artist referred to by the `?artist` variable, whereas "`?artist\dbpedia:artist`" selecteds all works created by the same artist. Further, a filter operator ([ ]) allows the selected resources to be restricted to match a certain predicate. In this example, we could use the RDF path "`?artist\ dbpedia:artist[rdf:type dbpedia:Album]`" to select only albums amongst the works of a musical artist in DBpedia.

2.4 Pre-Matching

To compare all pairs of entities of a source dataset S and a target dataset T would result in an unsatisfactory runtime complexity of O(|S|·|T|). Even after using SPARQL restrictions to select suitable subsets of each dataset, the required time and network load to perform all pair comparisons might prove to be impractical in many cases. Silk avoids this problem by allowing index pre-matching to find a limited set of target entities that are likely to match a given source entity. Target resources are indexed by one or more specified property values (most commonly, their labels) before any detailed comparisons are performed. During the subsequent resource comparison phase, the previously generated index is used to look up potential matches for a given source resource. Only these candidates are considered for detailed comparisons. We thus achieve a runtime complexity closer to O(|S| + |T|), making it feasible to interlink even large datasets under practical time constraints.

3. RELATED WORK

There is a large body of related work on record linkage and duplicate detection within the database community as well as on ontology matching in the knowledge representation community. Silk builds on this work by implementing similarity metrics and aggregation functions that proved successful within other scenarios. What distinguishes Silk from this work is its focus on the Linked Data scenario where different types of semantic links should be discovered between Web data sources that often mix terms from different vocabularies and where no consistent RDFS or OWL schemata spanning the data sources exist.

Related work that also focuses on Linked Data includes Raimond et al. [2] who propose a link discovery algorithm that takes into account both the similarities of web resources and of their neighbors, and Hassanzadeh et al. [3] who describe a framework for the discovery of semantic links over relational data which also introduces a declarative language for specifying link conditions. A main difference between LinQL and Silk-LSL is the underlying data model and Silk's ability to more flexibly combine metrics through aggregation functions.

4. CONCLUSIONS

The value of the Web of Data rises and falls with the amount and the quality of links between data sources. We hope that Silk and other similar tools will help to strengthen the linkage between data sources and therefore contribute to the overall utility of the network.

The complete Silk- LSL language specification and further Silk usage examples are found on the Silk project website at http://www4.wiwiss.fu-berlin.de/bizer/silk/.

5. REFERENCES

[1] Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web. http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/, 2007

[2] Raimond, Y., Sutton, C., Sandler, M.: Automatic Interlinking of Music Datasets on the Semantic Web. In: Linked Data on the Web Workshop (LDOW2008), 2008.

[3] Hassanzadeh, O., et al.: A Declarative Framework for Semantic Link Discovery over Relational Data. Poster at 18th World Wide Web Conference (WWW2009), 2009.

Improving Interaction via Screen Reader Using ARIA: An Example

Marina Buzzi, M. Claudia Buzzi
IIT-National Research Council
via Moruzzi 1
I-56124 Pisa, Italy
+39-050-3152632/1
{Marina,Claudia}.Buzzi@iit.cnr.it

Barbara Leporini
ISTI-National Research Council
via Moruzzi 1
I-56124 Pisa, Italy
+39-050-3152034
Barbara.Leporini@isti.cnr.it

Caterina Senette
IIT-National Research Council
via Moruzzi 1
I-56124 Pisa, Italy
+39-050-3152195
Caterina.Senette@iit.cnr.it

ABSTRACT
An interface conforming to W3C ARIA (Accessible Rich Internet Applications) suite would overcome many accessibility and usability problems that prevent disabled users from actively contributing to collaborative knowledge. In a previous phase of our study we first identified problems of interaction via screen reader with Wikipedia, then proposed an ARIA-based modified Wikipedia editing page. At this stage we only focused on the main content for editing/formatting purposes. To evaluate the effectiveness of an ARIA-based formatting toolbar, we only dealt with the main content of the Wikipedia editing page, not the navigation and footer sections. The next step using ARIA is to introduce landmarks (regions) and use the "flowto" property to be able to change the order of how page content is announced via screen reader. In this way the new user interface (UI) is functionally equivalent to the original Wikipedia editing page, and its appearance is very similar (apart from an additional combobox instead of a list of links), but usability is greatly enhanced. In this demo we will show interaction via Jaws screen reader using both the original and the proposed Wikipedia editing pages.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI)*. K.4.2 [**Social Issues**]: Assistive technologies for persons with disabilities.

General Terms
Design, Human Factors.

Keywords
Wikipedia, accessibility, usability, ARIA, blind.

1. WIKIPEDIA AND BLIND USERS
Web navigation is quite difficult for blind persons using a screen reader, since the pages are read sequentially, one row at a time according to the page code structure, starting from the top left corner of the page and losing all layout, style and font information [2].

The Wikipedia editing page [3] presents three main usability issues for totally blind users, as described in greater detail in [1]:

1. The formatting toolbar may be difficult to access. Navigating via Tab key, the user never perceives the presence of a toolbar on the page since its graphic icons are generated by JavaScript, so the browser is unable to recognize them as active elements.
2. It is difficult to select special characters and symbols. To insert a special character, the user must select an alphabet from a combo-box, and on the right side, a list of links displays its characters. Since this list may contain more than one hundred links, it is not suitable for navigation via Tab key. More, Jaws does not recognize uncommon characters and announces ambiguous text. For instance Jaws announces "link a" for each character in the group à, á, À, Á.
3. The user may lose the focus when editing and formatting text. The focus is managed via JavaScript: when one or more words in the text area are selected, all related values are stored by the script in order to apply the formatting correctly. When interacting via screen reader a user may not understand how the focus is processed since the screen reader operates with a "virtual focus", which may not coincide with the system focus (see in [1] for further details).

2. The new Wikipedia editing page
The World Wide Web Consortium promotes accessibility on the Web through its Web Accessibility Initiative (WAI). Recently (Dec 2008) the WAI group has produced a new version of the Web Content Accessibility Guidelines - WCAG 2.0 [5], which greatly improves the 1.0 version, including usability as a key factor to be closely coupled with accessibility. Furthermore, usability aspects concerning navigation via screen reader (such as overview of the page, a different flow of the reading compared to the sequential announcing of the page source code, interface navigability) are provided by W3C Accessible Rich Internet Applications specification (ARIA). ARIA aims to make dynamic web content and applications (developed with Ajax, (X)HTML, JavaScript) more accessible to people with disabilities [4]. Using ARIA, web designers can specify roles to add semantic information to interface objects, can mark regions of the page so users can move rapidly around the page via keyboard, etc. [4].

WWW 2009, April 20-24, 2009, Madrid, Spain.

To create the formatting toolbar using ARIA, we defined roles and properties as shown in Fig. 1. We used the *activedescendant* attribute in order to make the toolbar navigable via arrow keys when the editing modality is activated. Once the toolbar receives the focus via TAB key, the child elements -- i.e. each formatting widget -- can be accessed by left and right arrows, and can be activated by pressing the ENTER key, which applies the associated formatting function (Bold, Italics, etc.).

To avoid many links that make navigation difficult and can disorientate the blind user, we grouped characters from each group in a combobox located close to the first one, as shown in Fig. 1. Users first choose a 'language' and then select the desired character of that language. This compact solution is faster, since when navigating the combobox with arrow keys, the screen reader announces the character name directly while in the original interface every character read is preceded by the word 'link' (e.g. "Link è", "Link é" and so on). The latter modality is annoying, due to hearing the word "Link" over and over. Furthermore, the combobox can be skipped by just pressing the tab key once, while in the original interface it is necessary to run through all the links. We also associate a label with each character in the second combo-box, so the screen reader can announce this "clearer" description.
The focus problem is partially resolved by our new Wikipedia Editing Page. With Jaws English Version 9, the new interface allows users to insert and edit text without having to switch to navigation modality in order to find the active elements (widgets and comboboxes). The user activates the Editing modality and this remains for the entire editing/formatting process, reducing the number of steps needed to complete the entire task. Instead, with Jaws Italian Version 9 the focus is in the correct position but Jaws loses the editing modality.

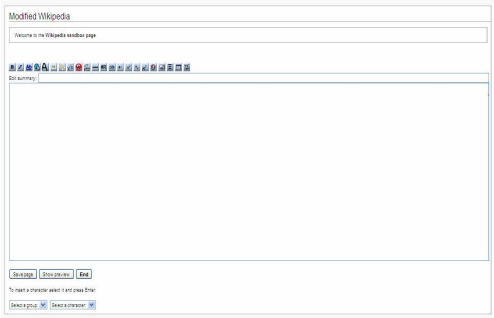


Figure 1. The new Wikipedia editing page

Different browsers may have different rendering and behaviors. The new Wikipedia editing page performs well with Mozilla Firefox v. 3.0.5, while at the moment, IE 7.0 does not interpret ARIA specification, which will be included in IE 8.

Once the prototype was tested and revised by the blind author of this paper, we conducted qualitative usability testing with a group of blind users who navigate via screen reader, in order to gather comments and suggestions and highlight any interaction problems. Our results show that it is possible to have a nice interface "look & feel" while assuring satisfaction and efficiency of use for all, and in particular for a blind user interacting via screen reader with a voice synthesizer. As an example, we chose Wikipedia to show that UIs interactions can be improved while maintaining a very similar graphic layout.

3. USABILITY TEST

A test description was sent to each participant in electronic format by e-mail. The test is accessible via web.

The protocol included a preliminary questionnaire, a set of tasks, and a post-test questionnaire:

1) In the preliminary questionnaire, participants provided information about their technical expertise, age, and knowledge of Wikipedia as well as of screen readers.

2) The remote testing procedure involved two simple tasks: (1) applying a formatting style, and (2) inserting a special character, to be completed on both the original and modified Wikipedia interfaces. We decided to perform a remote test in order to allow users to use their own computers and assistive technologies. The environment for executing the search tasks was available online at a specific URL and contained only a text box for the login (password was not required) and two buttons: one to the user interface reproducing the original Wikipedia Editing Page and the other to our Modified one.

3) The post-test questionnaire consisted of 16 questions divided into three sections: information regarding the subject's experience performing the assigned tasks, difficulties in carrying out the task, and degree of satisfaction.

The sample comprised 5 women and 10 men, age ranging from 18 to more than 75 years (only 1 person) as shown in Fig. 2.

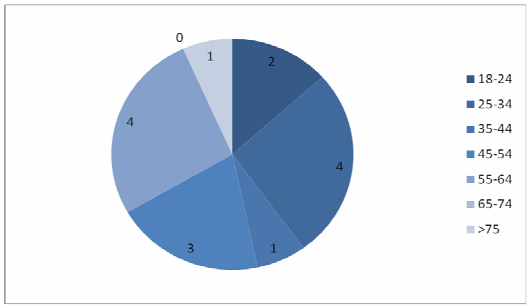


Figure 2. – User Age

Regarding Internet use, three subjects were novices, seven intermediate and five had experience. Only one of the users knew Wikipedia very well while five were fairly familiar, six mostly unfamiliar and three did not know it at all. Almost all the sample was using JAWS on Windows: six used v.8.0, four v. 9.0 (or higher) and five used v. 7.10 (or lower).

Preliminary data concerning test results of 15 totally blind users revealed that most users appreciated the simplified interaction; in particular 87% of users declare that applying a formatting style is more rapid in the modified UI (13% believe the 2 UIs are equivalent), and 73% declare that selecting of a special character is faster in the modified UI (20% users believe the two UIs are equivalent and for 7% original one is more rapid).

Concerning task performance, 80% of the sample completed the test in the modified UI respect to 47% on the original one; correctness of test results also improved: 33% for modified UI and 13% in the original one. Average execution time was decreased by 11% (from 7.1 min in the original UI to 6.3 min for the new one).

4. REFERENCES

[1] Buzzi, M.C., Buzzi M., Leporini B., Senette C. (2008). Making Wikipedia Editing Easier for the Blind. In proceedings of ACM NordiCHI2008, 20-22 October Lund, Sweden, 423-426.

[2] Goble C., Harper S., Stevens R. The Travails of Visually impaired Web Travellers. In Proceedings of Hypertext '00 (San Antonio, June 2000), ACM Press, 1-10.

[3] Wikipedia.Wikipedia English Version, http://en.wikipedia.org/.

[4] W3C. WAI-ARIA Overview, http://www.w3.org/WAI/intro/aria.php.

[5] W3C. Web Content Accessibility Guidelines 2.0. http://www.w3.org/TR/WCAG20/, 5 Dec 2008.

A REST Architecture for Social Disaster Management

Julio Camarero Puras
E.T.S.I. Telecomunicación (UPM)
Ciudad Universitaria
28040 Madrid, España
+34 609250 751
jcp@gsi.dit.upm.es

Carlos A. Iglesias Fernández
E.T.S.I. Telecomunicación (UPM)
Ciudad Universitaria
28040 Madrid, España
+34 91 549 5700 ext. 3004
cif@gsi.dit.upm.es

ABSTRACT
This article presents a social approach for disaster management, based on a public portal, so-called Disasters2.0, which provides facilities for integrating and sharing user generated information about disasters. The architecture of Disasters2.0 is designed following *REST* principles and integrates external mashups, such as Google Maps. This architecture has been integrated with different clients, including a mobile client, a multiagent system for assisting in the decentralized management of disasters, and an expert system for automatic assignment of resources to disasters. As a result, the platform allows seamless collaboration of humans and intelligent agents, and provides a novel web2.0 approach for multiagent and disaster management research and artificial intelligence teaching.

Categories and Subject Descriptors
H.3.5 [**Online Information services**]: Data sharing, web based services. I.2.11 [**Distributed artificial Intelligence**]: Intelligent Agents, multiagent systems.

General Terms
Experimentation.

Keywords
Disasters, social, Web2.0, Mobile, mashup, REST, Intelligent techniques.

1. INTRODUCTION
Natural disasters are associated to chaotic situations in which information is usually incomplete and imprecise, and this lack of information makes difficult the process of making decisions and managing effectively catastrophes. The secretariat of the International Strategy for Disaster reductions of the United Nations (UN/ISDR) proposes eleven lessons for disaster management; the first two are [1]:

Public awareness is an essential element of preparedness for saving lives and livelihoods.

Individuals and communities play important roles in managing risks from natural hazards.

This project proposes that Web2.0 technologies are a valuable tool to contribute to both lessons, enabling public knowledge and both individual and social participation in disaster management. Web2.0 [2] has proven the power of users' participation to create content, give opinions and organize themselves in social networks. Examples such as Wikipedia or del.icio.us show us the potential of this collective intelligence when it is used appropriately. This article proposes integrating different technologies usually grouped as Web2.0 technologies.

The management of natural disasters is a potential application of this collective intelligence. If everybody could report in real time the location and magnitude of a disaster, being even able to monitor it, then the response could be much more effective and immediate. In order to make this information available to anybody, our system has been designed to provide REST [3] services which can be combined using *mashups*.

2. DISASTERS2.0
The system presented in this article, Disasters2.0, is a complete platform for managing information about disasters. The core of this platform is user-generated data shared all around the world. Anybody can use the system to provide information about disasters using a computer, a mobile phone or any device with internet connection. This information can be instantly visualized in a map by anyone using simple symbols in a user-friendly interface. Moreover, resources (such as firemen or ambulances) and casualties can be displayed in the map and utilized to manage disaster situations.

In addition to the web and mobile clients, there are other elements interacting with the application: an expert system and a multiagent platform. The expert system has been designed to assign resources to disasters following rules which are based on the severity of the disasters. The multiagent platform models every resource as an intelligent agent and allows the user to see how these agents interact with the application by rescuing victims and fixing disasters.

3. ARCHITECTURE
Disasters 2.0 has been designed following REST principles. The main entities of a disaster have been modeled as resources. The system has considered the following entities: events such as fire or flood, allocated resources such as policemen or firemen, and damages such as victims. These resources are accessible from a REST interface using standard HTTP methods so that they can be obtained (GET), modified (PUT), created (POST) and deleted (DELETE).

In order to provide REST resources, the system Disasters 2.0 follows a client-server architecture, as shown in Figure 1.

The server is responsible for saving persistently all the information about disasters in a database, implementing the business logic to update and recover that information and serving it through REST services in JSON [4] format. This REST interface has been implemented using the open source framework Restlet [5].

The web client allows users to add information to the system very easily and to visualize all the activity of the world in a map. An AJAX Engine has been introduced to make this client more functional.

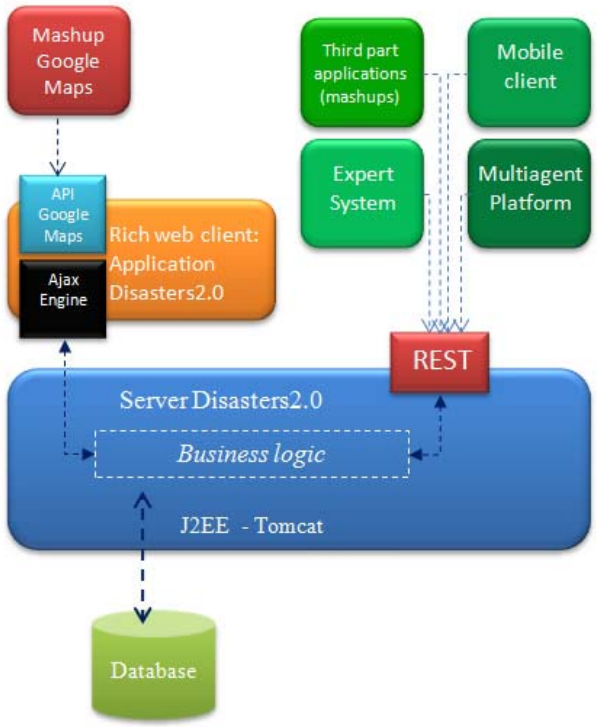


Figure 1. Architecture of Disastres2.0

The mobile client, developed with Mojax[6], has focused on the use of mobile web technologies (mCSS, mJavaScript...) for providing a web2.0 interface, and integrating 'mobile mashups' such as Yahoo Maps and Disasters2.0.

4. INTELLIGENT TECHNIQUES

This project has researched the application of intelligent techniques in the system Disasters2.0 aiming to improve the assignment and coordination of resources in disaster situations.

We have developed an expert system which assigns free resources (which are available in a database) such as policemen, firemen and ambulances to active events, such as fires, floods and collapses. This expert system has been developed using Jess [7], a rule engine for Java which allows the creation of highly complex rules by applying a pattern matching system.

A multiagent system has also been developed using Jadex [8]. In a first version, every resource (firemen, policemen and ambulances) can decide on its own how to act in an emergency situation. A second version of the system develops several levels of coordination and a hierarchical structure in which an emergency central service coordinates the resources in order to cover all the disasters while avoiding the blocking of roads or small areas.

This system communicates with the user interface of the web client and allows the user to see how the agents are moving to the assigned disasters, taking the victims to the closest hospital and resolving the disaster.

5. CONCLUSIONS

Internet has become an ubicuous participation platform, which enables unprecedented applications. This brief article shows how the social approach can help in the management of disaster situations. From a technical perspective, the system has researched on the usage of web2.0 interaction in both mobile and web browser clients. In order to fulfill this goal, a REST architecture which provides a resource-oriented interface for disasters has been defined. This interface has been validated with different clients, such as a social web application and a mobile client. In addition, this REST architecture has proven to be effective for integrating external systems, such as an expert and a multiagent system.

Disasters2.0 is available as an open source project and is currently being evaluated by local authorities in order to analyze its integration with a real system. In addition, the implementation of a disaster simulator is currently under progress. This simulator will provide researchers a benchmark for disaster management.

More information, demonstrations and videos about the project can be found at [9].

6. ACKNOWLEDGMENTS

This research project has been co-funded by the Spanish Education Ministry in the project TSI Improvisa (TSI2005-07384-C03-01).

7. REFERENCES

[1] Secretariat of the International Strategy for Disaster Reduction / United Nations. (2007) “Lessons for a safer future: Drawing on the experience of the Indian Ocean tsunami disaster,” International Strategy for Disaster Reduction (ISDR), Tech. Rep.

[2] T. O’Reilly (September 2005), “What is web 2.0: Design patterns and business models for the next generation of software,” O’Reilly Media. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/whatis-web-20

[3] R. T. Fielding, (2000) “REST: architectural styles and the design of network-based software architectures,” Doctoral dissertation, University of California, Irvine. http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm

[4] JSON Project, (December 20008) “Introducin JSON,” homepage of JSON project. http://www.json.org

[5] Noelios Consulting, (December 2008) “Restlet, Lightweight REST framework for Java,” homepage of the Restlet project. [Online]. Available: http://www.restlet.org

[6] mFoundry Inc, (December 2008) “Mojax, framework for mobile ajax” official website of Mojax. http://mojax.mfoundry.com

[7] Sandia National Labs, (December 2008) “Jess, the rule engine for the java platform,” official website of Jess Project. http://www.jessrules.com

[8] A. Pokahr, L. Braubach, and W. Lamersdorf, (2005) “Jadex: A bdi reasoning engine.” in Multi-Agent Programming, ser. Multiagent Systems, Artificial Societies, and Simulated Organizations, R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Eds. Springer, vol. 15, pp. 149–174.

[9] Disasters2.0, official website of the project (January 2009) http://lab.gsi.dit.upm.es/web/disasters/home

SCREEN-SHOTS

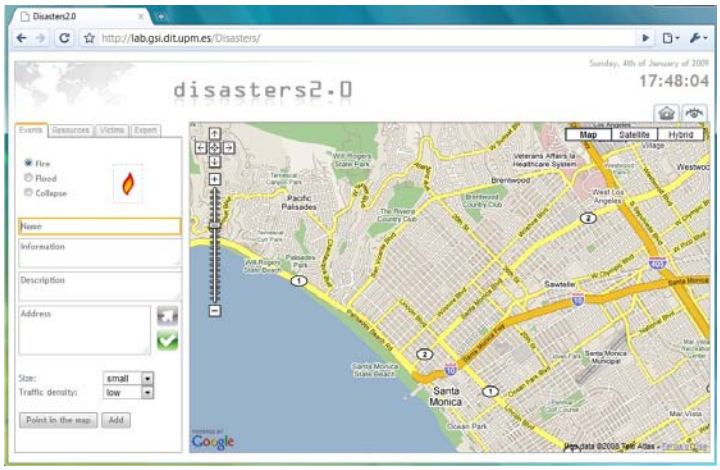


Figure 2. Web Client

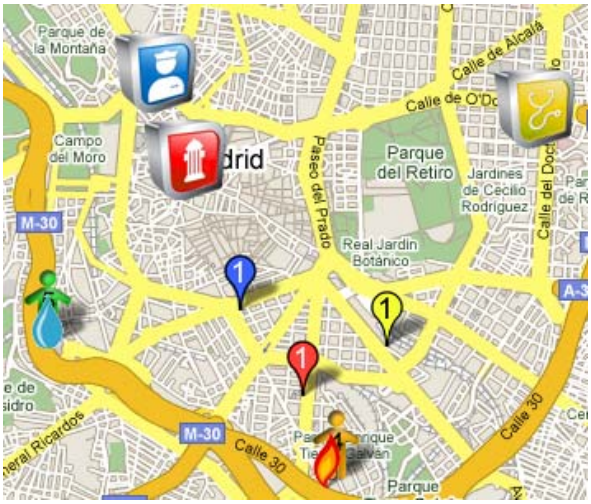


Figure 3. Multiagent Intelligent Platform

Figure 4. Mobile Client

Integration at Web-scale: Cazoodle's Agent Technology for Enabling Vertical Search

Kevin Chen-Chuan Chang Govind Kabra Quoc Le Yuping Tseng
Cazoodle Inc.
Emails: {kevin.chang, govind.kabra, quoc.le, yuping.tseng}@cazoodle.com

ABSTRACT

The Web today has "everything": Every object of interest in real world is starting to find its presence in the online World. As such, the search needs of users are getting increasingly sophisticated. How do you search for apartments? How do you find products to buy? Traditional paradigm of Web search, starting from keyword input, and ending in Web pages as output, stifles users—requiring intensive manual post-processing of search results.

As solution, Cazoodle has developed technology for enabling Web-scale vertical search. At the core is our agent technology for accurately crawling and indexing of structured data on the Web, with low cost of maintenance. We demonstrate the possibilities of integration at Web-scale using vertical search applications in two domains—Apartment Search (http://apartments.cazoodle.com) for finding rental apartments anywhere in US, and Shopping Search (http://shopping.cazoodle.com) for purchase of online consumer electronics.

1. PROBLEM

Web today has "everything"—every object in real world is starting to get an online presence. As such, the search needs of users are increasingly getting more sophisticated. We go on the Web for many of our everyday needs such as looking for apartments to rent, or products to buy, as illustrated in Figure 1. Thus, Web has become the ultimate destination for variety of search needs.

There is a need for vertical search engines to serve specialized search needs, as, often, the current search engines are not adequate. Consider, for example, a user looking for rental apartments. The current search engines accept keyword queries as input, and return Web pages as output. In this paradigm, it is not possible for user to specify the search query such as "2 Bedroom and 2 Bathroom Apartments, located within 2 miles of Chicago Downtown and rent below $2000." The user would search for general keyword queries such as "apartments in Chicago", then manually go through many resulting Web pages to identify the apartments that match her constraints.

2. SOLUTION

2.1 Observation: Prevalence of Structured Data

A key requirement for developing specialized vertical search engines is to integrate the structured data prevalent on the Web. Large amounts of Web data is structured, however, published in semi-structured format, as shown in Figure 2 (a). Consider, for example,

WWW2009, April 20-24, 2009, Madrid, Spain.

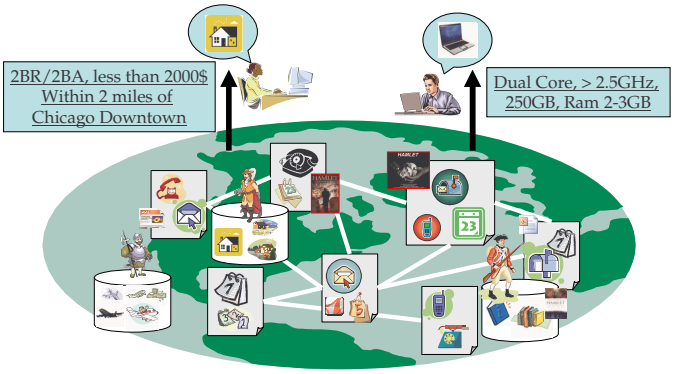


Figure 1: Web has everything: Search needs are getting sophisticated.

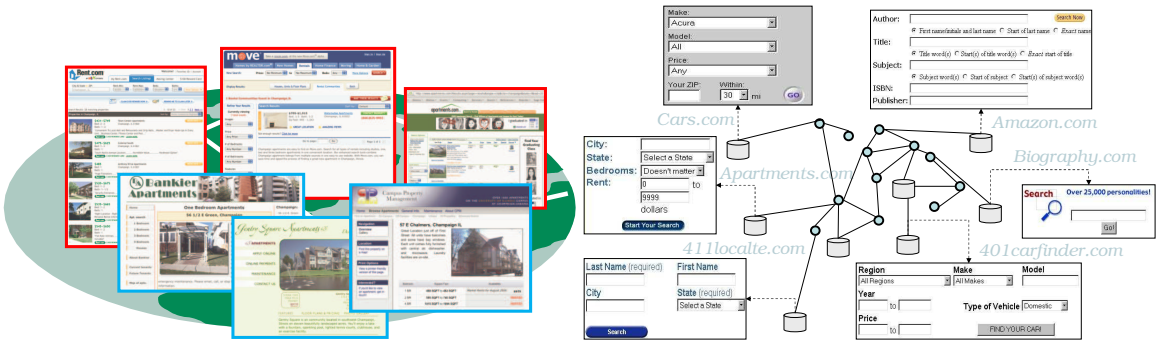
an apartment rental site such as www.rent.com. It consists of apartment listings describing number of bedroom, number of bathrooms, rent, street address, images, *etc.*Such structured data is prevalent in all domains, not just online commerce, such as bibliographic records, personal home pages, business directories, *etc.*

The prevalence of structured data goes beyond the statically linked *surface Web* pages; huge amounts of data is hidden in the *deep Web* behind the query forms. Many studies [1, 2], including our own [3], estimate the content on deep Web to be far greater than the surface Web. As shown in Figure 2 (b), users need to interact with different query forms on different sites to access the data hidden on the deep Web. The prevalence of huge amount of structured data on Web makes it mandatory to crawl and integrate the structured data across myriads of online sources—both on the surface and the deep Web.

2.2 Technology: Agents for Structured Crawling

As our core technology, we have developed intelligent *agent* technology that make it easy to model the semi-structured Web content into structured format. The agents can be trained to crawl and index new sites and domains with high accuracy, yet requiring minimal human effort.

The goal of integration at Web-scale introduces its unique challenges. On the one hand, for enabling useful applications, it is crucial that agents provide high extraction accuracy. On the other hand, to really scale well, the human labor cost for training of agents should be minimal; for example, it should not require the sophisticated skills of software engineers. Finally, the web sources change quite often, and therefore, the agent program needs to be robust to these changes; and when the agent is really broken, it should



(a) Surface Web (b) Deep Web

Figure 2: Prevalence of Structured Data

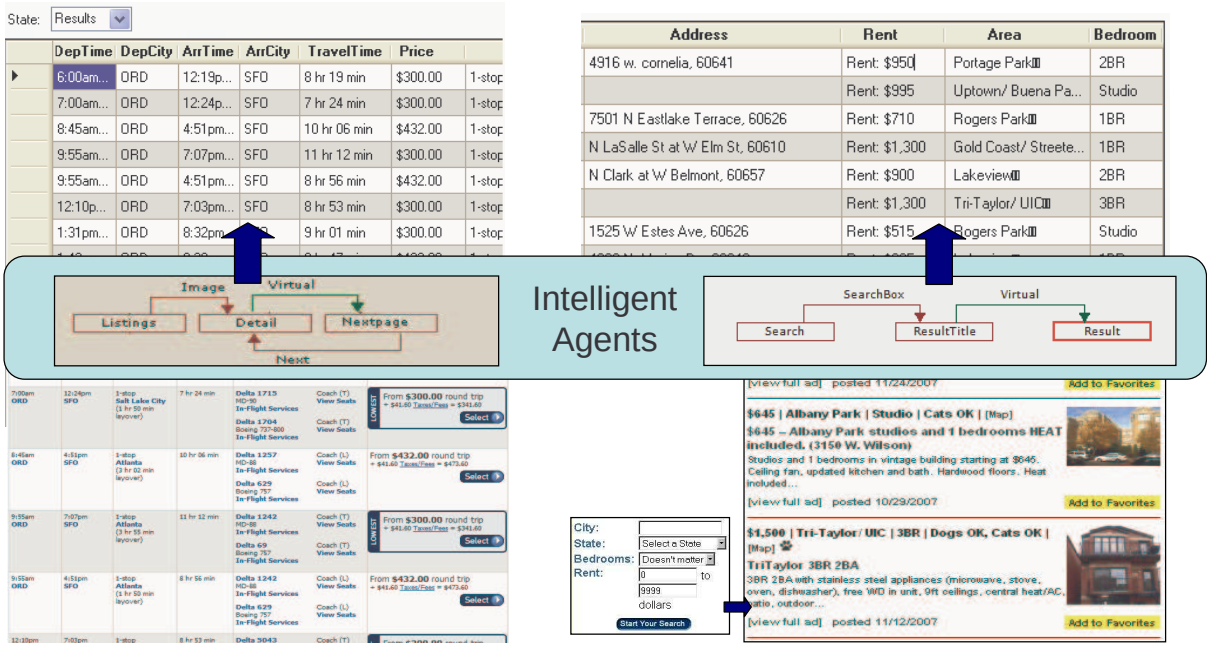


Figure 3: Agents for Structured Crawling.

be possible to adapt the agent to the new structure of the site with low cost of maintenance.

Our agent technology models Web sources using a set of expressive and persistent features that reflect how the sources present themselves to the world. While the expressive features allow us to describe all kinds of source data precisely, the persistent features provide resiliency to occasional changes in data presentation. Furthermore, the list of features used are not fixed. They are statistically analyzed and chosen, over both the source and the time dimensions, to keep the maintenance cost of agents low.

The manual effort required for training agents is quite minimal—at par with the skills high-school students—thus, allowing the technology to really scale to Web-scale. First, an initial version of source model is created automatically. Then, human operators can inspect the accuracy, to make any further necessary modifications. With much of the training automated, and a mix of intuitive, easy-to-use and robust set of features, users with no computer science education, including high-school students, are able to facilitate in completion and maintenance of agent programs.

The problem of structured data extraction and integration has been widely studied in both academia (*e.g.*, [4], [5]) and in industry (*e.g.*, Kapow RoboSuite [6], Connotate AgentSuite [7], and Dapper Factory [8]). Our solution provides a novel combination of high accuracy, and yet low cost of training and maintenance. Thus, we are able to rapidly scale to integrate thousands of online sources and provide useful vertical search applications.

3. TECHNOLOGY DEMONSTRATION

3.1 Agent Technology

The agent tool provides a GUI interface for human operator to train agent. As shown in Figure 3, the agent program operates as an automaton, to crawl and index structured data from Web sources. To train the agent, the human operator can begin with automatically analyzed model of the site, and add more labeled training examples. The system then adapts its model based on features of these labeled

(a) Apartment Search: http://apartments.cazoodle.com (b) Shopping Search: http://shopping.cazoodle.com

Figure 4: Web-scale Vertical Search Applications.

examples. The agent program thus generated can then be deployed into our data center for regular crawling.

3.2 Vertical Applications

Apartment Search

As our first product, Cazoodle is developing Apartment Search system, the first-ever one-stop search engine for rental apartments all over United States. The system is publicly available online at http://apartments.cazoodle.com and integrates apartments from thousands of online sources, including posting sites, user forums, and landlord sites. With its large scale integration capability, the system provides search over far more apartments than any of the popular search services.

With its precise understanding of structure of apartment listings, as shown in Figure 4 (a), the system is also able provide rich search capabilities, including rent, bedroom, bathroom, distance from any given address, *etc.*. Using mash-up with maps and street view images, users can take virtual tours of the surroundings of the apartments.

Shopping Search

Cazoodle Shopping Search system applies the agent technology to the popular domain of online consumer electronics shopping, to provide organic search of products and prices from hundreds of online vendors. The system currently supports search for laptop category, and is available online at http://shopping.cazoodle.com.

As shown in Figure 4 (b), for a given keyword query, the system will provide a list of matching products. For each product, the system shows organically crawled pricing information from all the online vendors. Once again, with the precise understanding of the structured attributes of the products, system allows users to filter results by CPU speed, RAM size, Hard Disk space, and of-course, price.

4. REFERENCES

[1] Michael K. Bergman. The deep web: Surfacing hidden value. Technical report, BrightPlanet LLC, December 2000.

[2] BrightPlanet.com. The deep web: Surfacing hidden value. Accessible at `http://www.press.umich.edu/jep/-07-01/bergman.html`, 2001.

[3] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, 2005.

[4] V Crescenzi, G Mecca, and P Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.

[5] Craig Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the web: A machine learning approach. 1999.

[6] http://www.kapowtech.com/.

[7] http://www.connotate.com/.

[8] http://www.dapper.net/.

The Future of Vertical Search Engines with Yahoo! BOSS

Ted DRAKE
Yahoo! France
17-19 rue Guillaume Tell
Paris, France 75017
+33 01 7091 3021
tdrake@yahoo-inc.com

ABSTRACT

While general search engines, such as Google, Yahoo!, and Ask, dominate the search industry; there is a new batch of vertical, niche search engines that could fundamentally change the behavior of search. These search engines are built on the open API's of Yahoo, Google, and other major players. However, Yahoo's recently released BOSS API has made these engines more powerful, more specialized, and easier to build and maintain.

These specialized search engines are finding the quality information in the haystack of general search terms. The general search engines, in turn, surface the niche results pages. This talk will discuss how to use the Yahoo! Boss search API to create a niche search engine. It will also show how this information can be mashed together with other APIs and finally how the results pages begin appearing in the general search engines. The talk is appropriate for engineers, entrepreneurs, and managers.

Categories and Subject Descriptors

Information Integration and Mash-ups, Search, Web Software and Tools

General Terms

Design, Experimentation,

Keywords

Search, yahoo, boss, api, mash, niche, vertical search, google, yql

1. INTRODUCTION

Congratulations, we've found 4,070,000 results for your search of "Chevre". Aren't you Lucky?

Internet users are accustomed to receiving millions of extraneous results for their search queries and they rarely go past a 3rd page. Generic search engines handle tremendous volumes of data yet they are not always the best source of answers.

There's a growing army of Vertical Search Engines finding the needles in these Google and Yahoo haystacks. They focus on specific topics or tasks and filter out much of the spam and irrelevant results. These niche search result pages are also showing up in the big search engines as nuggets of expertise.

Niche search engines range from task to content orientation. Two examples out of Paris exemplify this spectrum. Coloralo (http://coloralo.com) discovers images for children to color. Insider Food (http://insiderfood.com) is a regional food search engine that highlights recipes, restaurants, and foodie buzz from passionate bloggers and chefs within a region. Both of these are built on the Yahoo! BOSS API.

2. Building upon Web Services

Niche search engines are thriving with today's API availability. A developer can harness the indexes of the larger search engines while focusing on their particular market. Google, MSN, Ask, AOL, and Yahoo! offer robust search API's. Each has its strengths and weaknesses, however the biggest variation is the control given to a developer over presentation, ability to rerank results, availability of an SLA, and the limits on the number of requests.

Yahoo's BOSS search service gives the developers the most flexibility at this time. BOSS also offers unique data, such as RDF and microformat information, and the key terms associated with the page in Yahoo's index.

However, the search API is just a starting point. The niche magic arrives with the merging of data with other API's as well as local databases and and manipulation logic.

3. Creating a Platform for Mashing

While APIs have made building a search engine easy from the data side. There are new tools that simplify the manipulation and distribution. Yahoo! Pipes and Y.Q.L. (Yahoo Query Language) remove the burden of multiple API requests, caching, and mashing. Google's Application Engine, http://code.google.com/appengine/ , provides a stable platform for simplifying complex parsing via Python. Yahoo! BOSS even provides an SDK for AppEngine: http://developer.yahoo.com/search/boss/mashup.html.

4. The Vertical Search Mojo

Now we have data, the ability to combine it with other sources, and a platform to host. It's time to introduce the logic that makes a vertical search engine special.

Colaralo: A user requests an image of "horses" for their child to color. Colaralo requests a large set of images via Yahoo! Boss and analyzes the images to determine if they are black and white line drawings. The images are cached and the user is given a set of images they can print for their children.

InsiderFood: Search results are customized per region (Paris, San Francisco, New York,…). Each region depends on a set of custom data that includes a set of local experts. InsiderFood uses Yahoo! Boss's "sites" attribute to restrict the results to a set of resources. InsiderFood also mixes information from Technorati, Flickr, Twitter, Pipes, and Yahoo! Local.

TechCrunch: TechCrunch was the first site to use BOSS Custom. This advanced feature allows a developer to define specific data sources (including internal databases). BOSS is able to more intelligently parse data that may otherwise be hidden. TechCrunch combines their CrunchBase database, the blog archives, and other TechCrunch related sites in the main results. They are also able to specify alternate sources if there are no native results.

Sagoon: This search engine combines BOSS with its own index as well as other APIs to create clusters of information. Search results combine the basic search results with suggested clusters.

5. Introducing YQL and Personalization

YQL is a MySQL like syntax to a Yahoo! API that treats all forms of online data as tables. Programmers can access and mash data from virtually any web service, rss feed, HTML, or even static XML and spreadsheets. This includes access to social relationships via Oauth security.

Social networks make it easier for search engines to determine what a person may be interested in. YQL now allows developers access to user's profiles, relationships, and recent activities to drive their search results.

5.1 Examples Inspired by WWW2009 Papers

Mining Interesting Locations and Travel Sequences by Yu Zheng, Lizhu Zhang, Xing Xie and Wei-Ying Ma uses historical GPS data to create more accurate location-based knowledge. A search engine could use YQL to access a user's current location via **Fire Eagle**; a location standardization and sharing platform. It can check if the user has any friends that are nearby with the social.contacts data table. This near real-time information can be stored for tracking the movement and provide more relevant information. Knowing someone is inside Madrid is great, but it's even better to know they are heading towards the airport or Casa de Campo.

Understanding User's Query Intent with Wikipedia by Jian Hu, Gang Wang, Fred Lochovsky and Zheng Chen uses Wikipedia concepts to help determine a user's intent. Vik Singh, the architect of Yahoo! BOSS, has created a similar project that uses Twitter to determine what news articles are important. http://tweetnews.appspot.com/fresh?q=madrid .

6. A Symbiotic Relationship

This is not a one-way relationship. While the niche search engines are obviously benefiting from the largesse of the search API's; the big engines also benefit. Yahoo, as an example, collects click-through rates for better relevancy and may display the niche search engines result pages in the top results.

The BOSS API requires users to use a redirect link that allows Yahoo! to track usage. Combining these clicks with their focused context gives Yahoo a better idea of the target page's relevance. For instance, a click on a web page about puppies from the standard Yahoo! SERP may signify an importance of 5. But the importance could be higher when someone chooses that link over others from a puppy search engine.

Carefully designed search result pages can also be seen as a quality source of information. 123People, http://www.123people.co.uk , compiles information about people from many different data sources. Their result pages are one of the most relevant results, especially for the average person.

Creating a search engine like Google or Yahoo! requires enormous time, money, and resources. Luckily you can tap into their indexes to create your own specialized search engine. Vertical search engines solve problems; whether it is finding an image for your child to color or a recipe for a souffle. These search engines are easy to build, provide great user experience, and are gaining relevance amongst the big engines.

CentMail: Rate Limiting via Certified Micro-Donations

Sharad Goel Jake Hofman John Langford

David M. Pennock Daniel M. Reeves

{goel, hofman, jl, pennockd, dreeves}@yahoo-inc.com

Yahoo! Research, New York

1. INTRODUCTION

The Internet has reduced the cost of communication to near zero, benefiting billions of people around the world. One consequence, however, is that unsolicited and unaccountable commercial communication, or *spam*, is also sent indiscriminately in massive quantities at low cost, imposing a large burden on recipients and on systems. Spammers have infiltrated nearly every form of online communication, including email, instant messaging, blog comments/trackbacks, and web pages/links. We propose a system for rate limiting Internet communications broadly, emphasizing the case of email.

Domain and content filtering are currently the first line of defense against spam. But domain filtering is difficult to apply when spammers send from legitimate domains (e.g., by opening email accounts at Yahoo! and Gmail). It also places an onerous burden on new domains to establish themselves as legitimate senders. And content filtering requires considerable effort to maintain as spammers constantly evolve to circumvent the latest filters.

Many people, most notably Bill Gates, have observed that adding a modest cost to sending email, for example by requiring postage stamps like ordinary mail, could significantly deter spam. Researchers have proposed and analyzed several such systems, including variations where the recipient keeps the payment, the recipient has the option of either keeping or refunding the payment [7], the sender "burns" human time or CPU cycles [1, 5], or the sender pays to a charity of their or the recipient's choice [3].

Although an equilibrium where senders and receivers all adopt email stamps benefits nearly everyone, there is a serious *flag day* problem, or coordination failure, that makes the equilibrium hard to reach from the status quo. Senders do not want to spend money buying stamps if recipients are not checking stamps, and recipients would not bother to check stamps if few senders use them. The hurdle for senders is heightened by the very real possibility that spammers who already hijack other people's computers may now in addition drain the users' stamp accounts of money to send spam or, worse, to funnel the money to themselves. It seems that the prospect of some day reducing spam is not enough to convince a critical mass of both senders and recipients to adopt a new protocol and monetary accounting infrastructure.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

```
--
Alice supports the Sierra Club
Her donation was matched by Bob's Widgets
Powered by CentMail.net -- Do good. Fight spam.
```

Figure 1: CentMail stamps appear as email signatures that promote the sender's cause.

2. CENTMAIL

CentMail is a general economic framework for rate-limiting that directly addresses the issue of incentives, providing tangible net benefit to even the earliest adopters with no need for coordination. This tailoring of incentives is threefold. First, users receive stamps in exchange for making donations to charitable organizations of their choice. In this way, many users would incur no added financial burden since they already make these donations regardless of their participation in CentMail. In fact, 89% of U.S. households already make annual donations, with an average household contribution of $1620 (or 3.1% of income) [2] and a median on the order of $100. (A donation of $100 yields enough stamps to send email to 10,000 people—27 recipients per day, every day of the year.) Second, the stamps themselves are implemented as email signatures (see Figure 1) that promote both the sender's cause and the sender's support of that cause. A recent survey estimates that "people are 100 times more likely to donate when asked by a friend or family member than an anonymous solicitation" [4]. Third, users amplify their impact via matching donations[1], either by a corporate sponsor acknowledged in the message signature, or by the mail provider who may eventually see a reduction in spam-associated costs, estimated to be on the order of billions of dollars per year worldwide [6]. For many potential users, these design choices in sum yield net benefit, even in the absence of other participants in the system.

Although CentMail offers users benefits even in the absence of coordination, cooperation is still required for it to function as a spam deterrent. We note that in this regard, CentMail improves upon existing proposals in that CentMail stamps serve as advertisements for the system itself. This allows us to leverage the latent social network of email contacts to encourage adoption of the system. If enough senders join, recipients may take notice and begin to whitelist stamped

[1]Although users may attempt to defraud sponsors by effectively stamping fake messages, we think the likelihood of and potential damage from this type of behavior are small and can often be detected.

email, allowing them to tune their content-based filters more aggressively, increasing the incentive for senders to stamp email, forming a virtuous cycle.

We do not expect to see complete adoption of CentMail. However, even with limited adoption, we believe stamping to be an effective tool that works in conjunction with aggressive domain and content-based filtering to detect and deter spam.

A similar and independent effort at IBM Research in 2004 aimed to promote "charity seals" in email [3]. To our knowledge, that system was not implemented. Our main contribution is to make the idea concrete by defining a formal protocol, implementing a working prototype of the service, and analyzing the benefits and drawbacks of the approach and how we envision the service's promotion and adoption.

2.1 The Protocol

The CentMail protocol supports authentication of both emails and arbitrary text documents. For example, a "document" could correspond to a comment on a weblog, or a listing of links on a web page. In each case, CentMail certifies that the content was validated by a charitable donation. The two key operations are certification and verification:

```
Centmail.certify(amount, digest)
    return null

Centmail.verify(digest)
    return {amount, queries}
```

These function calls are authenticated, and in particular, the user making the call is identified by the global parameter `Centmail.user`.

The `certify` function takes as input an `amount` to donate and the `digest`, or SHA-1 hash, of the content.[2] It is generally in the sender's interest to append a *nonce* (i.e., a randomly generated string) to their content to ensure each message is unique—although this is not explicitly required by the protocol. When a message is certified, the CentMail server debits `CentMail.user` and stores the `digest` for later verification. In order to efficiently scale, message digests are eventually expired, and hence are maintained on the CentMail server only temporarily.

To verify content, the user passes the document `digest` to `Centmail.verify`. This call returns the `amount` which was donated, and `queries`, the number of times the content has been verified. The return value `queries` is a crucial piece of information since the certifier's "payment" (i.e., donation) is less meaningful when the content is consumed by multiple individuals. For example, in the case of email, donating $0.01 for an email which is ultimately sent to 1000 people is less of a commitment than donating $0.01 for an email sent to a single individual.[3] Typically, recipients would accept messages when `amount/queries` is at least $0.01, and treat messages not meeting this threshold as effectively unstamped. In this latter case of unstamped—or effectively unstamped—messages, existing domain-based and content-based techniques could still be applied to classify email. The recipient, however, is free to enforce any filtering policy of their choice.

Aside from certifying messages, it is often useful for applications to rate limit requests (e.g., for account creation, or for posting comments to blogs). CAPTCHAs are typically used in these contexts, requiring users to burn "human cycles." CentMail facilitates an alternative, economic approach to rate limiting which allows third parties to ask users to burn (i.e., donate) money. This feature is intended for web-based applications, and is implemented through an additional function call:

```
Centmail.request(amount, transID, returnURL)
    return null
```

When the requester makes this call, the end user (i.e., the individual being asked to make a donation) is redirected to the CentMail website to confirm the donation. Afterward, CentMail posts an authenticated response to `returnURL` and redirects the user back to the originating site. The requester receives confirmation that a donation was made, but no other identifiable information about the user.

2.2 The Implementation

A beta implementation of CentMail is available at `CentMail.net`. In addition to an initial implementation of the CentMail API on the server side, we have developed a CentMail plug-in for Thunderbird, the popular open source email client, a Firefox plug-in for web-based email services, including Yahoo! Mail and Gmail, an Apple Mail plug-in, and perl scripts for clients such as Pine, Mutt, and Evolution that support filtering email through arbitrary scripts.

Full Version of the Paper

At `CentMail.net` we have available the full version of this paper, which details protocol specifics, proves correctness properties, discusses additional related work, and addresses concerns such as how to deal with mailing lists and possible attacks.

3. REFERENCES

[1] A. Back. Hashcash—a denial of service counter-measure (5 years on). Tech Report, 2002.

[2] M. S. Brown. *Giving USA: The Annual Report on Philanthropy for the Year 2007*. Giving USA Foundation, 2007.

[3] P. Capek, B. Leiba, and M. N. Wegman. Charity begins at ... your mail program. `http://www.research.ibm.com/spam/papers/charity-seals.pdf`, 2004.

[4] P. B Carroll. Charity cases, July 14, 2008. The Wall Street Journal, `http://online.wsj.com/article/SB121554292423936539.html`.

[5] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of CRYPTO'92*, pages 137–147, 1993.

[6] D. Ferris, R. Jennings, and C. Williams. The Global Economic Impact of Spam, 2005. Technical Report 409, 2005. `http://www.ferris.com`.

[7] T. C. Loder, M. W. Van Alstyne, and R. Walsh. An economic response to unsolicited communication. *Advances in Economic Analysis & Policy*, 6(1), 2006.

[2] In the case of a plain text document, computing the message hash is straightforward. For email, however, care must be taken so that determining which header fields to include, and their order, is unambiguous.

[3] Often one can avoid the problem of multiple recipients consuming the same content by using different nonces for each recipient. Then instead of sending 1000 people the "same" certified email message, each recipient would in fact be verifying their own unique copy.

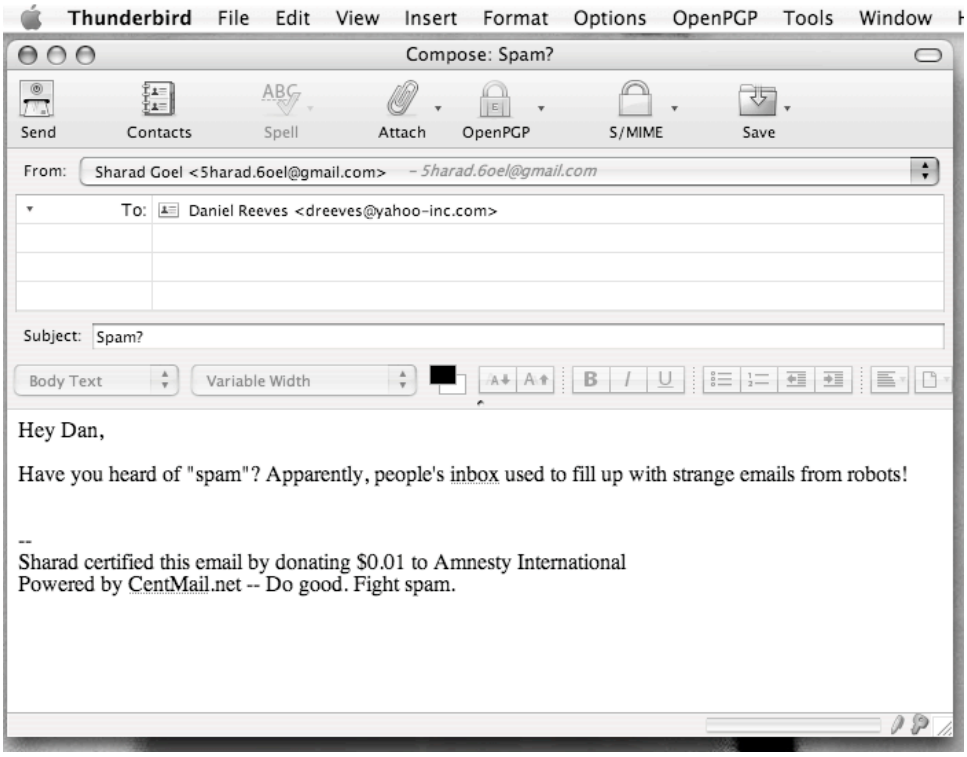


Figure 2: CentMail plug-in for the Thunderbird email client.

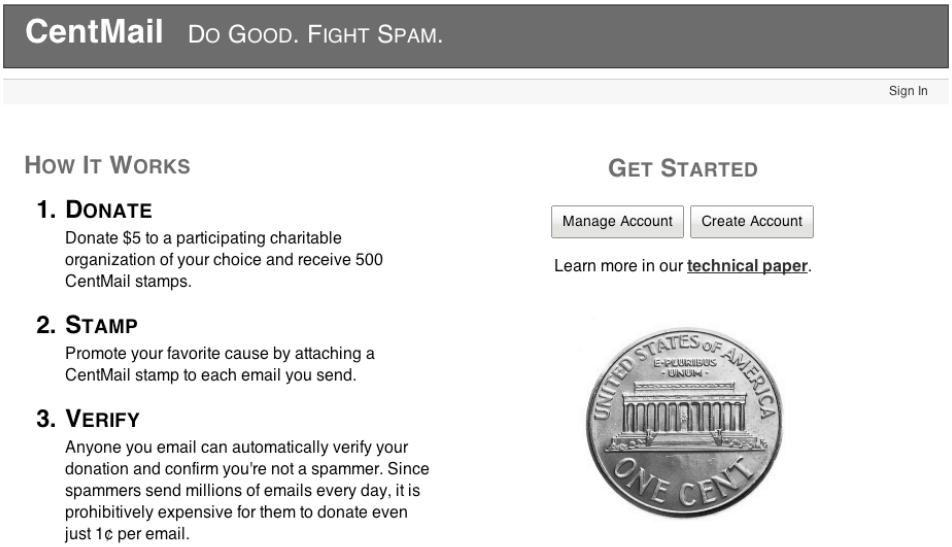


Figure 3: http://CentMail.net

Bootstrapping Web Pages for Accessibility and Performance

Clint Andrew Hall
Cerner Corporation
2800 Rockcreek Parkway
Kansas City, MO 64011
011 (816) 201-5045

clint.hall@cerner.com

ABSTRACT

The construction of the typical web page has changed significantly (and for the better) over the last ten years. New and more sophisticated user interface technologies and the availability of higher bandwidth speeds have transformed the once rather bland, text-based documents into entire experiences. Web "pages" have now taken on the role of web "solutions," entities that can accept much more complex user input and thus respond just as richly.

Today, the more portable a document attempts to become, the more metadata, CSS selectors and scripts unrelated to the desired experience are downloaded and included in the document. Indeed, many CSS and JavaScript frameworks include compatibility code that is ignored unless applicable to that particular browser, (and is thus wasted in those contexts). Older versions of browsers—considered a less-than-significant demographic—are often ignored, allowing sites to fail visually when they cannot interpret the CSS selectors and script correctly.

There have been attempts to optimize the potentially excessive or incompatible code. Server-side techniques use unreliable request headers, such as the User Agent, and can only make assumptions about client-side capability. Considering the depth of the browser market combined with the breadth of available devices and platforms, User Agent detection can be daunting, expensive and worse still, error-prone. Client-side approaches push detection and inclusion logic to the browser, often using the same unreliable request headers and rarely taking the form factor into account. JavaScript and CSS frameworks have taken cross-browser compatibility only so far; they primarily focus on the desktop and do come with a cost. Neither process is configurable in and of itself, and both require changes to source code when a new skin is created.

In computing, *bootstrapping* ("to pull oneself up by one's bootstraps") refers to techniques that allow a simple system to activate a more complicated system. In this talk, I introduce *Web Bootstrapping*, a process by which an accurate collection of only those static resources and metadata necessary for a unique experience be delivered passively, by the most performant means possible. In further contrast to existing methodologies, this approach determines resources based on capability, form factor and platform by targeting and collecting the often-immutable attributes of the client, not specifically its identity or version.

Web Bootstrapping allows for rule-based, externalized, server-side configuration, further promoting progressive enhancement and client performance. Rather than interrogate the User Agent string, the bootstrapper applies rules such as, "if the screen size is larger than 800x600, deliver the following resources," or, "if the client supports Java and Adobe Flash, deliver the following script to inject components." Even proprietary means of version detection—such as conditional comments in Internet Explorer—can be exploited for rules processing. The bootstrapper gives a developer reliable access to a host of client-side information on the server, and can thus apply generic, intelligent and performant rules to the delivery of resources.

Individual presentation collections, or *skins*, can be edited independently of each other, and new collections can be added at run-time without changing any source code of the document. The bootstrapper also supports on-demand resource inclusion, (e.g. Ajax) with identical capability consideration. As an added benefit, by virtue of being a server-side approach, this process is also capable of concatenating static resource content from remote sources, thereby avoiding cross-site scripting and mixed-content warnings.

Using an implementation of this bootstrapping method, we have been able to demonstrate several solutions with different skins based on browser and device. At runtime, the bootstrapper allows developers to create, repair or remove both the skins and the rules that provide them without requiring a code release. This gives us greater flexibility toward the browser landscape, as well as agile reaction to any changes within it, all the while coexisting effectively in our release cycles. We've also seen significant performance improvements in our web solutions, and we continue to experiment with this technique.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Portability.*

H.5.2 [**Information Systems and Presentation**]: User Interfaces – *Theory and Methods.*

General Terms

Algorithms, Performance, Human Factors, Theory.

Keywords

Web, User Interface, Web Browsers, Performance, Accessibility, Cascading Styleheets, JavaScript

1. WEB BOOTSTRAPPER APPROACH

The following outlines in detail how the bootstrapper is set up and how it executes to effectively deliver static resources and content.

1.1 Setup

The first step in using the bootstrapper is defining the unique experiences to be delivered. These skins will be attributed to entire collections of devices based on their attributes and classified by the Bootstrapper ruleset. Table 1 defines a sample set of experiences.

Table 1. Sample set of Experience Definitions

Skin	Description
Plain text	Default skin, devoid of presentation or behavior other than applied by the client.
Mobile	Suitable for small screens; minor subset of CSS, minimal script.
Desktop	CSS 3, Javascript 1.5 compliant skin.

Each of these experiences has their own set of resource files and metadata.

Second, the Bootstrapper is provided a set of rules capable of classifying clients based on their attributes. Using Table 1, we can see that the most prominent rule is screen size, followed by discerning the difference between WebKit and other mobile devices.

Third, each page within the solution includes a single JavaScript include that triggers the Bootstrapper. Optionally, the source URI of this include can specify a *bundle*, a configuration key that corresponds to a group of pages that share a set of resources. For example, suppose several pages use the same set of resources because they all deal with a similar topic, such as allergies. These pages could then share the same bundle name, "allergies," which would direct the Bootstrapper to deliver only those resources necessary.

Finally, the resource delivery preference for the solution is configured. A Bootstrapper delivery preference defines how the resources are loaded; the selection of the method depends on the number and size of the resources as well as the requirements of the HTML.

There are three preferences currently employed by the Bootstrapper: *HTML*, *Network* and *Concurrency*.

1.1.1 Favoring HTML

This mode of resource delivery is the least performant, but most closely mimics how the resources are included at run time. The JavaScript files found by the server process are concatenated together and returned with the rules engine result. Metadata, CSS and other file paths are subsequently written to the document using `document.write`. Since `document.write` applies content to the document as it being process, this method matches exactly how the document would be interpreted if the includes were a part of the HEAD element source.

This methodology could be chosen in those cases where the HTML contains JavaScript that relies on functionality included before the load of the page is completed.

1.1.2 Favoring the Network

This mode of resource delivery is faster than the HTML method, as it concatenates and delivers all specified JavaScript in one network transfer. In addition, it adds inclusions via Document Object Model (DOM) manipulation; this allows resources to be downloaded concurrently.

This method assumes there does not exist any inline script within the document that must be executed before the document finishes loading. This method would also be preferred when there is very little JavaScript to load, thus favoring the network by not firing off any additional network traffic to load resources.

1.1.3 Favoring Concurrency

In some cases, the most performant method of resource delivery favors concurrency. In this mode, the Bootstrapper "phases" resource loading using two passes. In the first pass, CSS and metadata are added to the document, followed by an include requesting the second pass. In the second pass, JavaScript is added to the document. In this way, the presentation of the page is handled first, followed by its behavior.

While involving an additional request to the server, this method defers the processing of JavaScript to run concurrently with the download of other resources. This can be extremely effective when larger amounts of JavaScript are required.

1.2 Page Load Execution

Figure 2 provides a visual representation of the Bootstrapper methodology. The process executes as follows:

1. An HTML document is served with one static script include:

   ```
   <script type="text/javascript" src="/Bootstrap"
   type="text/javascript"></script>
   ```

   or, optionally:

   ```
   <script type="text/javascript"
   src="/Bootstrap?bundle=[bundleKey]"></script>
   ```

   where `?bundle=[bundleKey]` is an optional grouping identifier.

2. The `/Bootstrap` URL references a server-side process which delivers a configured Bootstrapper JavaScript capability detection object.
3. The Bootstrapper object attempts to collect a number of attributes from the client.
4. The Bootstrapper object appends a SCRIPT tag to the HEAD, resulting the following addition:

   ```
   <script type="text/javascript"
   src="/Bootstrap?r=1[&attribute=value...]>
   </script>
   ```

   where `r=1` informs the server-side process the Bootstrapper object has gathered attributes and is ready to receive resources.

5. The `/Bootstrap` URI maps to the server process, which then passes the request parameters and the `user-agent` to a rules engine.
6. A rule set is evaluated, resulting in a list of metadata, CSS and JavaScript file paths to be returned to the client.
7. The Bootstrapper appends, in order, metadata, CSS and JavaScript include script, based on the configured favoritism.

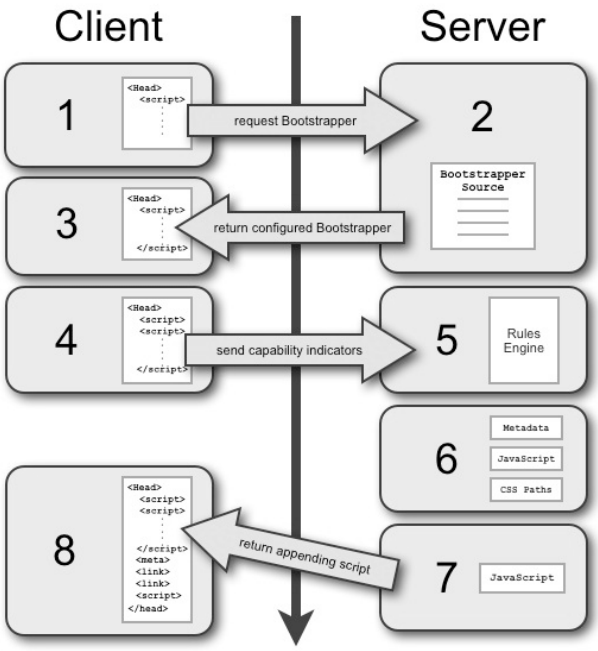


Figure 1. Bootstrapper Execution

1.2.1 The Bootstrapper Payload

The Bootstrapper JavaScript object is responsible for collecting, transmitting and appending resources to the document. The object can be implemented to look for any number of attributes to transmit to the server-side process. The attributes collected from the Bootstrapper are output as SCRIPT elements, typically within the HEAD. The script that arrives as a result of the Bootstrapper contains calls to its API that are responsible for appending elements to the document.

1.2.2 Resource Collection

The server-side process is provided with a collection of paths of resources from the rules engine. The resources can be local or remote.

Local JavaScript resources can be concatenated together if they are accessible from the process, (e.g. from a file stream). If not found, the paths are output directly to the HEAD, and are thus relative to the URI of the page.

Remote JavaScript resources can be concatenated with local JavaScript resources if the server side process can support loading these files via HTTP. This is particularly useful in order to avoid cross-site scripting warnings, as the source of these files originate within the domain of the solution, rather than remotely.

As they can contain references to relative images, local CSS resources are always output relative to the page URI.

1.3 On-demand Execution

On-demand inclusion, or the loading of resources without a page refresh, executes in much the same way as the on-load process. The only exception to this is that the favoritisms do not come into play; all resources are added to the HEAD using DOM manipulation. This method can be used to further increase performance by loading only that JavaScript and CSS necessary for the result of a behavior. The bundle attribute can be used to great effect in this use case.

2. IMPLEMENTATION

Our implementation of the Bootstrapper uses Java as its language, the J2EE container and JBoss Drools as the rules engine.

2.1 The Bootstrapper Payload

In our implementation, the Bootstrapper payload has been configured to gather and transmit the following attributes:

- Screen Height, (`screen.height`)
- Screen Width, (`screen.width`)
- IE Version, (if IE, using conditional comments)
- Color Depth, (`screen.colorDepth`)
- Java Enabled, (`navigator.javaEnabled`)
- Platform, (`navigator.platform`)
- Vendor, (`navigator.vendor`)

In most cases, these attributes are well supported, implemented as immutable within the client, and are fairly reliable. The absence of any attribute, however, need not halt execution; indeed, well-implemented rules can interpret and handle such cases.

2.2 A Bootstrapper Servlet

The server-side component of the Bootstrapper in our implementation is a J2EE servlet. This servlet has access to any local static resources on its CLASSPATH; this includes both JARs and the web container itself. We have also implemented remote resource access through the Apache HTTPClient. [23]

2.3 JBoss Drools Rules Engine

We chose to implement connect an instance of the Bootstrapper servlet to the JBoss Drools Rules Engine. [24] This enabled an expressive, static file-based rules language that could be externalized on the server and read at runtime.

3. RESULTS

We continue to apply and refine this technique. One of the primary goals of this talk is to introduce the concept and encourage vetting of its approach and appropriateness.

Query GeoParser: A Spatial-Keyword Query Parser Using Regular Expressions

Jason Hines
GenieKnows.com
1567 Argyle St
Halifax, NS, Canada
+1 902-431-4847
jasonh@genieknows.com

Tony Abou-Assaleh
GenieKnows.com
1567 Argyle St
Halifax, NS, Canada
+1 902-431-4847
taa@genieknows.com

ABSTRACT
Local search engines enable a user to search for entities that contain both textual and spatial information. Thus, queries to these systems may contain both spatial and textual components. Parsing the queries requires breaking the query into textual keywords and identifying the components of the spatial description. Our Query GeoParser is a simple, but powerful, regular-expression-based spatial-keyword query parser. Query GeoParser is implemented in Perl and utilizes many of Perl's capabilities in optimizing regular expressions. By starting with regular expression building blocks for common entities such as numbers and streets, and combining them into larger regular expressions, we are able handle over 400 different cases while keeping the code manageable.

Categories and Subject Descriptors
I.2.7 [**Artificial Intelligence**]: Natural Language Processing— *Language parsing and understanding*; I.5.4 [**Pattern Recognition**]: Applications— *Text processing*

General Terms
Algorithms, Performance, Design.

Keywords
Parsing, Geo-Spatial, Local Search, Query Processing.

1. INTRODUCTION
There has been a growing commercial interest in local information within Geographic Information Retrieval, or GIR, systems. Many local search engines exist to fulfill searches for entities that contain both textual and spatial information, such as Web pages containing addresses or a business directory. Queries to these systems may contain both spatial and textual components—spatial-keyword queries [2]. Parsing the queries requires breaking the query into textual keywords and identifying components of the spatial description. For example, the query 'Hotels near 1567 Argyle St, Halifax, NS' could be parsed as having the keyword 'Hotels', the preposition 'near', the street number '1567', the street name 'Argyle', the street suffix 'St', the city 'Halifax', and the province 'NS'. Developing an accurate query parser is essential to providing relevant search results. Such a query parser can also be utilized in extracting geographic information from Web pages. We discuss related works in section 2 followed by a description of our Query GeoParser in section 3. In section 4 we offer some comments on our approach followed by a conclusion in section 5.

WWW 2009, April 20-24, 2009, Madrid, Spain.

2. RELATED WORK
Many techniques can be applied to spatial-keyword query processing. For example, natural language processing techniques such as n-grams [5] or part-of-speech tagging combined with hidden Markov models are a viable approach. Other known techniques include statistical and hybrid template matching techniques such as those developed by [4] and [1], respectively. Although the literature suggests that hybrid approaches offer the highest precision and recall, these techniques can involve relatively complex code making the overall problem harder to manage. Furthermore, machine learning and hybrid techniques are generally slower due to the extra processing overhead involved in the underlying algorithms.

3. QUERY GEOPARSER
We call our parser a Query GeoParser and it is a regular-expression-based parser developed in Perl. By developing our parser in Perl we can take full advantage of its extensive regular expression capabilities such as wildcards, grouping, pre-compiling, look-ahead, and building blocks – a coding technique where one can insert smaller pre-compiled regular expressions into larger regular expressions. Our parser takes four steps to parse a query:

1. Clean the query
2. Encode, or mark, the query
3. Match the encoded query against query templates
4. Decode the query

3.1 Cleaning
Consider the query 'Hotels near 1567 Argyle St, Halifax, NS'. First, the query is converted to lowercase and all commas are padded with white space. Commas play a special role in spatial-keyword queries by acting as segmenting characters for the spatial query component. All other punctuation is removed from the query and the query is then split by whitespace. We end up with the set of tokens *{'hotels', 'near', '1567', 'argyle', 'st', ',', 'halifax', ',', 'ns'}*.

3.2 Encoding
A simple back-tracking algorithm is employed to match combinations of consecutive tokens against known entities such as prepositions, numbers, postal/zip codes, cities, states, and popular businesses (highly frequent business names that contain spatial entities). Some tokens may match multiple known entities

and in cases where there is overlap the first and longest match is chosen. For tokens that match a known entity, we append a pipe separated list to the token. For example, if a token was *argyle* and it matched a city and a popular business it would be encoded as *argyle|city|popb* where *popb* indicates a popular business and *city* indicates a city.

Also, during this step, the cleaned query tokens are encoded. The letter 'x' and any other character that is not a letter, digit, or underscore are converted to the format *xXX* where *XX* is a hexadecimal representation of the character. This is important for template manageability in the matching step (see section 3.3) because it allows us to make extensive use of the Perl regular expression wildcard \w.

For our example, the encoded query is '*hotels near|prep 1567|number argyle|city|popb st|type , halifax78|city|popb , ns|state'*. The token *near* was matched as a preposition (*prep*), *1567* was matched as a number (*number*), *argyle* was matched as a city (*city*) and a popular business (*popb*), *st* was matched as a street suffix (*type*), *halifax* was matched as a city (*city*) and a popular business (*popb*), and *ns* was matched as a state (*state*). Note that the original token *halifax* is now *halifax78*. The letter *x* was encoded as *x78*.

Marking requires an efficient lookup algorithm and data structure. The data structure employed should be able to store as many known entities as possible and be able to lookup potential candidates quickly. Possible solutions are in-memory tree structures or hash tables.

3.3 Matching

The encoded query is run through a set of query templates until there is a match or the list of possibilities is expired. The order of template matching is important because some tokens have multiple meanings. Therefore, templates must be given precedence based on ones needs. In general, stricter, or specific, templates should be matched before lenient templates. Our parser currently handles over 400 different templates. In our current example, the query matches a template such as '*<keywords> <preposition> <street>, <city>, <state>'*. As you can see, the template is very high level and easy to manage. Each component of the high level templates is broken down into what are called building blocks where each building block is a separate regular expression.

3.4 Decoding

The last step is to decode, or decipher, the encoded query. If a query successfully matches a query template then the remaining unknown parts of the query can be tagged. In our example, the final output of the parser would tell us that there is a keyword *Hotels*, a preposition *near*, a street number *1567*, a street name *Argyle*, a street suffix *St*, a city *Halifax*, and a province *NS*.

4. DISCUSSION

The main advantage of this approach is the overall simplicity and management of the source code. For example, popular tokens like streets may appear in many different query templates, and in many forms, hence the regular expressions to match just a street can be large enough that if you were not using building blocks it would be very difficult to maintain. We find that by employing a mark-and-match [3] approach we are able to increase the efficiency of our parser because many templates can be quickly ignored.

The main drawback of this approach is the handling of misspellings. Regular expressions require strict matches and dealing with misspellings requires developing further tools such as a basic spell checker, a geographically aware spell checker, or both. Correcting the spelling of tokens also depends on the context of the tokens, further complicating matters. For example, the token *piza* could be interpreted as a misspelling for *pizza* or it could be a correct spelling for *Piza Street in Lawrence, New York*.

In our experience, a regular-expression-based parser can also be troublesome when there are popular business names that include geographic entities (e.g., Kentucky Fried Chicken) or prepositions (e.g., Janes on the Common). There are also many city names that are common search terms or other spatial entities, such as *Street, Maryland*.

We have used Query GeoParser in a production environment[1] for more than one year and have learned that using only a regular-expression-based approach is not the most ideal solution. The approach must be combined with other techniques to correctly handle the drawbacks it imposes. This further reinforces the findings of [5].

5. CONCLUSION

Our Query GeoParser is a simple, but powerful, regular expression-based spatial-keyword query parser. Query GeoParser is implemented in Perl and utilizes many of Perl's capabilities in optimizing regular expressions. By starting with regular expression building blocks for common entities such as numbers and streets, and combining them into larger regular expressions, we are able handle over 400 different cases while keeping the code manageable and easy to maintain. We employ the mark-and-match technique to improve the parsing efficiency. First we mark prepositions, numbers, postal/zip codes, cities, and states. Following, we use matching to extract keywords and geographic entities. The advantages of our approach include manageability and performance. Drawbacks include a lack of geographic hierarchy and the inherent difficulty in dealing with misspellings.

6. REFERENCES

[1] Egnor, D., and Greenfield, L.E. Location Extraction. World Intellectual Property Organization. International Publication Number WO 2006/074055 A1 (2006).

[2] Hariharan, R., Hore, B., Li, C., and Mehrotra, S. 2007. Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In *Proceedings of the 19th international Conference on Scientific and Statistical Database Management* (July 09 - 11, 2007).

[3] Keselj, V., Abou-Assaleh, T., and Cercone, N. DalTREC 2006 QA System Jellyfish: Regular expressions mark-and-match approach to question answering. In The Fourtheenth Text Retrieval Conference (TREC 2006) Proceedings, Gaithersburg, Maryland, USA (2006).

[4] Riise, S., Patel, D., and Stripp, E.H. Geographical Location Extraction. U.S. Patent 7,257,570 (2007).

[5] Yu, Z. High Accuracy Postal Address Extraction From Web Pages. Master's Thesis. Dalhousie University. Halifax, Nova Scotia, Canada (2007)

[1] http://www.genieknows.com

DBpedia - A Linked Data Hub and Data Source for Web and Enterprise Applications

Georgi Kobilarov
Freie Universität Berlin
Garystr. 21
D-14195 Berlin, Germany
georgi.kobilarov@fu-berlin.de

Christian Bizer
Freie Universität Berlin
Garystr. 21
D-14195 Berlin, Germany
chris@bizer.de

Sören Auer
Universität Leipzig
Johannisgasse 26
D-04103 Leipzig, Germany
auer@uni-leipzig.de

Jens Lehmann
Universität Leipzig
Johannisgasse 26
D-04103 Leipzig, Germany
lehmann@informatik.uni-leipzig.de

ABSTRACT

The DBpedia project has extracted a rich knowledge base from Wikipedia and serves this knowledge base as Linked Data on the Web. DBpedia's knowledge base currently provides 274 million pieces of information about 2.6 million concepts. As DBpedia covers a wide range of domains and has a high degree of conceptual overlap with various open-license datasets that are already available on the Web, an increasing number of data publishers has started to set data links from their data sources to DBpedia, making DBpedia one of the central interlinking hubs of the emerging Web of Data. This paper gives an overview about the DBpedia project and describes how application developers can make use of DBpedia knowledge within their applications.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

Keywords

Web of Data, Linked Data, Knowledge Extraction, Wikipedia, DBpedia

1. INTRODUCTION

Knowledge bases are playing an increasingly important role in enhancing the intelligence of Web and enterprise search and in supporting information integration. Today, most knowledge bases cover only specific domains, are created by relatively small groups of knowledge engineers, and are very cost intensive to keep up-to-date as domains change. At the same time, Wikipedia has grown into one of the central knowledge sources of mankind, maintained by thousands of contributors. The DBpedia project [1] leverages this gigantic source of knowledge by extracting structured information from Wikipedia and by making this information accessible on the Web.

WWW2009, April 20-24, 2009, Madrid, Spain.

The DBpedia knowledge base currently describes more than 2.6 million things, including at least 213,000 persons, 328,000 places, 57,000 music albums, 36,000 films, 20,000 companies. The knowledge base consists of 274 million pieces of information (RDF triples). It features labels and short abstracts for these things in 15 different languages; 609,000 links to images and 3,150,000 links to external web pages; 4,878,100 external links into other RDF datasets. Entities are classified in 4 concept hierarchies: The manually build DBpedia ontology, the YAGO [6] ontology, the UMBEL[1] ontology and a SKOS representation of the Wikipedia category system. The DBpedia knowledge base has several advantages over existing knowledge bases: It covers many domains, it represents real community agreement, it automatically evolves as Wikipedia changes, and it is truly multilingual.

This paper is structured as follows: We give an overview of the DBpedia extraction framework and describe how Web applications can access the DBpedia knowledge base. Afterwards, we describe three use cases of the DBpedia knowledge base and its concept identifiers: Knowledge source for web applications; interlinking hub to connect data sources, and vocabulary for annotating web documents.

2. DBPEDIA EXTRACTION FRAMEWORK

While Wikipedia articles consist mostly of free text, they also contain various types of structured information, such as infobox templates, categorisation information, images, geo coordinates, links to external Web pages and other Wikipedia articles, disambiguation information, redirects and cross-language links. The DBpedia extraction framework extracts these different kinds of information and turns them into RDF data.

All entities in DBpedia are assigned a unique URI of the form `http://dbpedia.org/resource/`*Name*, where *Name* is taken from the URL of the source Wikipedia article, which has the form `http://en.wikipedia.org/wiki/`*Name*.

[1] http://umbel.org

The type of wiki contents that are most valuable for the DBpedia extraction are Wikipedia infoboxes. Infoboxes contain attribute value pairs and are used to display an article's most relevant facts as a table at the top right-hand side of the corresponding Wikipedia page. Wikipedia's infobox template system has evolved over time without central coordination. Therefore, there is a lack of uniformity of infoboxes. Different templates use different names for the same attribute (e.g. `birthplace` and `placeofbirth`). While the first version of our infobox extractor used a generic method to turn property value pairs into triples and hence struggled with the different names of attributes, our new mapping-based extractor aims to solve that problem by introducing a central DBpedia ontology and mappings between templates and the ontology.

This ontology was created by manually arranging the 350 most commonly used infobox templates within the English edition of Wikipedia into a subsumption hierarchy consisting of 170 classes and then mapping 2300 attributes from within these templates to 720 ontology properties. The property mappings define fine-grained rules on how to parse infobox values and define target datatypes, which help the parsers to process values.

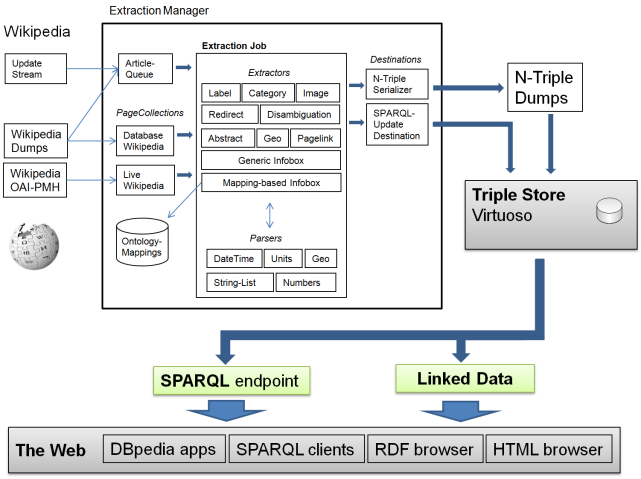


Figure 1: Overview of DBpedia components.

Figure 1 gives an overview of the open-source DBpedia extraction framework. The main components of the framework are: *PageCollections* which are an abstraction of local or remote sources of Wikipedia articles, *Destinations* that store or serialize extracted RDF triples, *Extractors* which turn a specific type of wiki markup into triples, *Parsers* which support the extractors by determining datatypes, conversion values between different units and splitting markup into lists. *ExtractionJobs* group a page collection, extractions and a destination into a workflow. The core of the framework is the *Extraction Manager* which manages the process of passing Wikipedia articles to the extractors and delivers their output to the destination.

3. ACESSING DBPEDIA OVER THE WEB

In order to fulfill the requirements of different client applications, we serve the DBpedia knowledge through four access mechanisms:

Linked Data. DBpedia URIs be dereferenced over the Web according to the Linked Data principles [2, 3]. DBpedia resource identifiers (such as http://dbpedia.org/resource/Berlin) are set up to return (a) RDF descriptions when accessed by Semantic Web agents (such as data browsers or crawlers of Semantic Web search engines), and (b) a simple HTML view of the same information to traditional Web browsers. HTTP content negotiation is used to deliver the appropriate format.

SPARQL Endpoint. We provide a SPARQL endpoint for querying the DBpedia knowledge base. Client applications can send queries over the SPARQL protocol to this endpoint at http://dbpedia.org/sparql.

RDF Dumps. N-Triple serializations of the datasets are available for download at the DBpedia website at http://wiki.dbpedia.org/Downloads32.

Lookup Index. In order to make it easy for Linked Data publishers to find DBpedia resource URIs to link to, we provide a lookup service that proposes DBpedia URIs for a given label. The Web service is available at http://lookup.dbpedia.org/api/search.asmx.

4. USE CASES

This section describes three use cases of the DBpedia knowledge base and its concept identifiers.

4.1 Data Source

The DBpedia knowledge base is served on the Web under the terms of the GNU Free Documentation License. Application can therefore query the knowledge and use the query results, including labels and abstracts in 15 languages, for their purposes. Did you ever need a list and abstracts about 'Dutch cities over 200 meters altitude', 'Italian musicians from the 18th century', 'episodes of the HBO television show "The Sopranos"' or 'Software developed by an organisation founded in California by a person born in a European country in the 1960s' for your application? The DBpedia knowledge base can provide them for you. More sample SPARQL queries can be found on the DBpedia wiki at `http://wiki.dbpedia.org`.

4.2 Interlinking Hub

Linked Data [2, 3] has become increasingly popular as a lightweight approach to publishing and connecting data on the Web. Over the last year, an increasing number of data publishers have started to set data links to DBpedia concepts, making DBpedia a central interlinking hub for the emerging Web of data. Currently, the Web of interlinked data sources around DBpedia provides around 4.5 billion pieces of information and covers domains such as geographic information, people, companies, films, music, genes, drugs, books, and scientific publications[2].

A major advantage of using DBpedia as linking hub is that it contains semantic relations bridging different domains. This way specialized domain-specific datasets linked to DBpedia can be leveraged in cross domain (and cross dataset) queries. Figure 2 shows the cloud of interlinked data sources

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

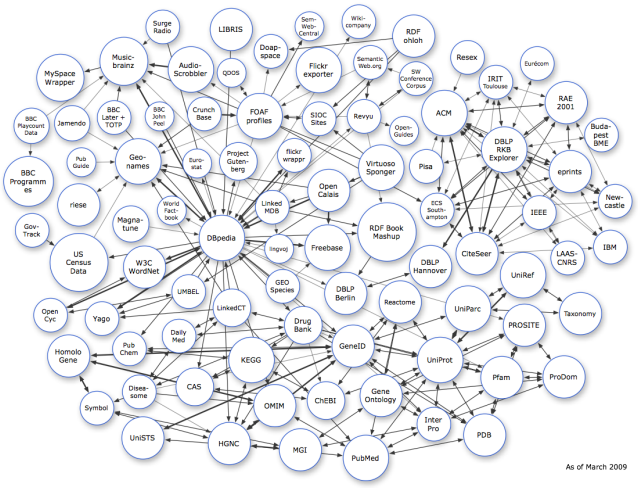


Figure 2: The Linking Open Data Cloud

Figure 3: Tagging BBC news articles with DBpedia URIs

around DBpedia in the Linking Open Data Projekt. DBpedia itself links to external datasets with currently around 5 million outgoing RDF links, making it a good starting point for Semantic Web crawlers. Up to our knowledge, there are currently 21 external data sources that set RDF links to DBpedia. As link discovery engines such as *Silk* [7] and *ODDlinker* [5] make it increasingly easy to generate links between datasets, the data source network around DBpedia will hopefully keep on growing in the future.

Using DBpedia as interlinking hub is also a promising way to connect data sources within organizations. The BBC, for example, is using DBpedia as their main source of Linked Data identifiers and at the same time to link their data to the public Data Web [4]. DBpedia identifiers are in the process of replacing other vocabularies, that were build by the BBC to connect diverse data sources, as maintaining these vocabularies proved to be too expensive.

4.3 Content Annotation

Annotating Web documents according to a global conceptual schema eases the discovery of content across Web sites. In addition to using DBpedia identifiers to interlink data sources on the Web, several projects have started to employ DBpedia URIs to annotate text documents. Named Entity Recognition (NER) services such as OpenCalais[3], MuddyBoots[4] or Zemanta[5] use DBpedia URIs as identifiers for extracted entities or link their own identifiers to DBpedia URIs. The semantic bookmarking service Faviki[6] uses DBpedia identifiers for tagging Web pages similar to Delicious.

A combination of named entity extraction and semantic content tagging can be found in the Content Ling Tool (see Figure 3) developed by the BBC. It allows BBC editors to annotated news articles [4]. Here, the Muddy Boots service extracts named entities from the article text and identifies those with DBpedia URIs. In addition, editors can add annotations determining the topics of the articles using the auto-completion API of the DBpedia lookup service.

[3]http://www.opencalais.com
[4]http://www.muddy.it
[5]http://www.zemanta.com
[6]http://www.faviki.com

Semantic annotations allow the discovery of Web content that is related to a DBpedia entity and enable 3rd parties to enrich their content with data provided by DBpedia and interlinked data sources. Multiple documents annotated with the same URI of e.g. a musical artist can be related together and DBpedia can serve the structured background information about that artist with information such as her birthplace, the list of her albums or even movies containing one of her songs as part of their soundtracks.

By using the same set of concept identifiers within textual documents and structured data, it therefore becomes possible to bridge between the classic document Web and the emerging Web of Data.

5. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: a nucleus for a web of open data. In *The 6th International Semantic Web Conference (ISWC 2007) Busan, Korea*, 2007.

[2] T. Berners-Lee. Linked data, 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[3] C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the web, 2007. http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/.

[4] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media meets semantic web - how the bbc uses dbpedia and linked data to make connections. In *6th European Semantic Web Conference (ESWC2009), Semantic Web In-Use Track*, 2009.

[5] Oktei Hassanzadeh, et al. A declarative framework for semantic link discovery over relational data. In *Poster at 18th World Wide Web Conference (WWW2009)*, 2009.

[6] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semant.*, 6(3):203–217, 2008.

[7] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - a link discovery framework for the web of data. In *2nd Linked Data on the Web Workshop (LDOW2009)*, 2009.

Creating Personal Mobile Widgets without Programming

Geetha Manjunath
geetha.manjunath@hp.com

S Thara
thara.s@hp.com

Hitesh Bosamiya
hitesh.bosamiya@hp.com

Santhi Guntupalli
santhi.guntupalli@hp.com

Vinay Kumar K
vinaykumar@hp.com

Ragu Raman G
ragu@hp.com

Hewlett Packard Research Labs,
No 24, Adugodi, Bangalore, India

ABSTRACT
Our goal is to radically simplify the web so that end-users can perform their personal tasks with just a single click. For this, we define a concept called *Tasklet* to represent a task-based personal interaction pattern and propose a platform for automatically creating, sharing & executing them. Tasklets can be deployed on the client, cloud or on telecom provider network – enabling intuitive web interaction through widgets, thin mobile browsers as well as from mobile phones via SMS and Voice. Our key innovation is a tool and platform that enables end-user to simplify personally valuable task on the web without any knowledge of programming.

Categories and Subject Descriptors
D.1.2 [**Automatic Programming**] H.5.3 [**Web-based Interaction**] H.5.4 [**Hypertext/Hypermedia**]

General Terms
Human Factors, Languages

Keywords
Programming-By-Example, Widgets, Mobile, Interaction, HCI

1. INTRODUCTION
The web has grown in multiple ways - content, complexity of navigation, number of web sites, web services and service providers. Current web interfaces require the user to remember websites, perform complex navigations, fill multiple forms and essentially figure out a sequence of actions that are needed to accomplish a simple task – often across multiple sites. Even when she succeeds in doing it once, the next time a similar goal arises, she has to recall and manually repeat the same exercise! This problem is even more acute in mobile context, when the device has interface challenges and user is very focused on getting a task done. We need to simplify this web interaction. By radically simplifying the usage of the web and providing familiar means of interacting with it, we believe more number of people will start deriving value from the web. Further, reduced interaction complexity of web can be a major productivity improvement for an IT-savvy user. Finally, 1-click access to personally valuable web tasks from mobile devices can be a compelling user experience of the mobile web.

An appliance or widget model of customized packaging of complex actions is a well-known paradigm of simplification. Since different users perform different personal tasks on the web, and possibly in diverse ways, use of pre-packaged widgets does not scale. Today, if a user wants to create a new widget that does

WWW 2009, April 20-24, 2009, Madrid, Spain.

Figure 1: User Created Personal Widgets

what he wants, it requires development of a new web application, which in turn mandates programming knowledge of one or more of Java, JSP, Servlets, AJAX, JavaScript, Flex or some similar technology. Our goal is to break this limitation and enable the end-user to create simple 'single-click' widgets for his personal task without programming. We propose to demonstrate a tool and platform for creating, sharing and executing such personal widgets. The target user for our tool is an end-consumer with just simple web-browsing knowledge. Advanced user can, of course customize the behavior of the auto-generated widget.

2. OUR SOLUTION
Our key approach to solve the above problem is to package a user's personally valuable web interaction across multiple web sites as a first class web-entity and provide an execution platform for robust execution of the same. For this, we define a concept called Tasklet to capture a user's preferred way of accomplishing a task – compressing the sequence of web actions needed to perform a specific task in a user-specified way. These Tasklets can be user-created, virally shared, customized and composed with other Tasklets and web services.

In order to enable an end user to program a Tasklet, we use the technique of Programming-By-Demonstration. All the user needs to do is to perform the task (web interaction) once using her web browser by giving some sample inputs on all the relevant web sites. We record and analyze these browser actions to auto-generate custom personal widgets which compress all the required actions into a single-click interface. Further, using our cloud-hosted services, these widgets can be executed from a mobile device (through a thin web browser, SMS or even voice), despite certain device limitations.

For instance, if a user frequently goes on 2-day trips from Bangalore to Delhi by Jet Airways, she can create a custom Tasklet which takes the date of travel as input and books her regular flights and reserves a hotel room at her favorite hotel using her credit card details for payment – all this with a single click! Another example is a Bill-Payment widget with local-

language interface that navigates to the right web portal, performs the payment, and translates the results to the user's language. A Tasklet captures a sequence of web actions across multiple sites under a widget. It can also be used by portal providers and service providers to mobile-enable their business.

3. THE DEMO

We have a working prototype of the Tasklet authoring tool, Tasklet Repository Service and Tasklet execution environment for both Windows PC and mobile devices (both PDA and phone). The demo will be as follows. Firstly, we will demonstrate the Tasklet execution environment. We will show the working of a system-generated widget to perform the task of downloading a copy of an ACM research paper (*given a title, go to the ACM digital library and fetch the soft copy of the complete paper*). We use this desktop icon/widget which takes the title of the paper and returns a local downloaded PDF file. While the widget is performing the task, we will open its 'debug' window to pictorially show the user all the hidden web actions that is performed by the widget in order to complete the current task of fetching a research paper.

Next, we will show how one can easily author new widgets by just '*doing it once on the browser'*. The actions required for a specific web task is first recorded using a browser plugin. This recording file will be registered with our Tasklet repository service to generate a desktop widget automatically. During Tasklet registration, the user will be shown a list of possible input parameters to select from (*do you want to change the flight 'date', 'destination' in every run*?). On execution of the newly generated widget, these selected parameters will be used as inputs to the new task. One can note that the parameter list is dependent upon the type of task being performed and hence requires sophisticated analysis of the actions as well as web pages, which is performed by our repository service before the program synthesis phase. The Tasklet can also be composed with post processing services (Language translation). The Tasklet repository allocates a unique Tasklet URL to enable simplified triggering of Tasklet Execution from mobile devices. The repository services can be used to share this URL over SMS or email; accessing this URL results in cloud-based execution of the Tasklet. Two types of widgets will be generated – a thick widget that runs as a Windows Desktop application and another 'thin widget' that is meant for Mobile devices.

Specifically for demonstrating the creation of a new widget described above, we will choose the task of retrieving a person's horoscope from a website she trusts. We will use a web browser (with a plugin to enable recording) to *navigate to MSN page, go to the Horoscope page and extract content for my Sun sign*. While registering this new Tasklet with our Tasklet repository web service, we will demonstrate its features for sharing, parameterization and post processing. We will then show the auto-generation of Windows Desktop and Google Android widgets. On execution, these widgets will show that day's Horoscope in a local language (Hindi) – without requiring the user to open a browser or go to the website.

Lastly, we will show voice-based invocation of personal web tasks through our Tasklet Voice Gateway developed over HP Opencall Telecom Infrastructure. To show the voice demo, we will dial the Tasklet number from an SIP client, to hear the output - text of *my personal MSN horoscope* (that we created on the spot). A second example of currency conversion that allows voice input for the task will also be shown.

4. RELATED WORK

Programmer-created mobile widgets and associated widget-stores such as Nokia Widsets, Google Apps, iPhone AppStore are becoming very popular. Developing a widget to do a personal task on any of the above platforms requires sophisticated web programming – which is not possible for a naïve web user or even for a tech-savvy person who has not kept abreast with web protocols and standards. Personalized portals, on the other hand, enable the user to customize only the layout of the website (iGoogle, My Yahoo!) by selecting elements from existing services. Mashup tools like Yahoo Pipes allow users to compose new services by 'visually' specifying their information access needs (through a data flow graph) which may be non-trivial for a naïve web user. Some tools allow creation of widgets for web forms (Intel MashMaker) or some specific RSS feeds (OpenKapow) through PBE. Our solution goes beyond all of the above and enables end-users to simplify a complete task on the web with multiple 2-way interactions. We use a technique of programming-by-demonstration to exclude need for programming. Finally, web testing tools (like iMacro, HP Mercury QTP), also provide a record and replay facility over websites. They have a very tight coupling between the recording and replay modules – and hence cannot work on mobile devices. Further, a lot more processing is needed to make a browser recording into a program and to accommodate changes in the inputs the next time the same task is repeated – bringing the need for parameterization.

5. CONCLUSION

In this paper, we presented our approach to radically simplify a user's personally valuable task on the web. We proposed the concept of a *Tasklet* to capture a sequence of web interactions needed to perform a web task. We presented a simple mechanism to create these Tasklets without explicit programming – through Programming-by-Doing. In this conference, we would like to demonstrate the creation, sharing and execution of these Tasklets on Desktops and mobile devices. Figure 2 shows some screenshots of the different phases of the demo. We believe that our proposal of end-user created widgets is unique and we look forward to test out the uptake of the proposed tool and solicit feedback from the expert Web developer community.

6. ACKNOWLEDGMENTS

We would like to thank our friends and colleagues at HP Labs India for their review comments and good Tasklet use cases. In particular, we acknowledge the interesting discussions and useful inputs from Ajay Gupta, Praphul Chandra and Nidhi Mathur. Special Thanks to Vimal, an interaction designer from HFI, Bangalore for the Portal design and Raghu for his support in the configuration of mobile internet devices.

7. REFERENCES

[1] Nokia Widsets http://widsets.com
[2] Yahoo! Pipes, http://pipes.yahoo.com.
[3] Intel MashMaker, http://mashmaker.intel.com
[4] OpenKapow, http://openkapow.com
[5] iMacro Macro Recording Tool, http://iopus.com

Figure 2: Screenshots of the proposed demo

A Virtual Oceanographic Data Center

Sean McCleese[1] Chris A. Mattmann[1,2] Rob Raskin[1] Daniel J. Crichton[1] Sean Hardman[1]

[1]Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, USA

[2]Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA

{smcclees,mattmann,raskin,crichton,shardman}@jpl.nasa.gov

mattmann@usc.edu

ABSTRACT

Oceanographic datacenters at the National Aeronautics and Space Administration (NASA) are geographically sparse and disparate in their technological strengths and standardization on common Internet data formats. Virtualized search across these national assets would significantly benefit the oceans research community. To date, the lack of common software infrastructure and open APIs to access the data and descriptive metadata available at each site has precluded virtualized search. In this paper, we describe a nascent effort, called the Virtual Oceanographic Data Center, or VODC, whose goals are to overcome the challenge of virtualized search and data access across oceanographic data centers, and to provide a common and reusable capability for increasing access to large catalogs of NASA oceanographic data.

Categories and Subject Descriptors

D.2 Software Engineering, D.2.11 Domain Specific Architectures

General Terms

Design, Experimentation, Languages, Theory.

Keywords

Virtual Oceanographic Data Center, OODT, VODC.

1. INTRODUCTION AND MOTIVATION

Oceanographic datacenters at NASA are tasked with the long-term storage and distribution of oceanic science datasets. These datasets vary in file size (megabytes to 100s of gigabytes), metadata structure (EOS-CORE, CF convention) and data formats (netCDF, HDF, etc.). Similarly, the datasets are stored across a variety of geographically separated datacenters divided along science discipline lines (e.g., buoy data versus altimeter data).

Existing ocean web search applications available at each datacenter to date have proven largely successful, focusing on human-centric, interactive search. However, the web search applications themselves do not directly enable programmatic access (the ability of a software program to rapidly and autonomously search and download data) to the underlying oceanographic datasets available from each site. This type of access would demand standard application programming interfaces (APIs) and data services that a software program could consume, interpret, and use as a building block for higher-level capabilities (e.g., bulk downloading and discovery of ocean data).

A review of two widely used ocean data centers, the National Virtual Oceanographic Data System (NVODS) and The National Oceanographic and Atmospheric Administration (NOAA)'s National Oceanographic Data Center (NODC) directly corroborated the lack of suitable data access and discovery APIs. These centers, along with others, do employ some level of data access APIs and services, such as the Thematic Realtime Environmental Distributed Data Services (THREDDS) catalog [1]. THREDDS creates an XML catalog with links to data accessible via the Open-source Project for a Network Data Access Protocol (OPeNDAP) technology [2]. However, THREDDS is simply a metadata-rich flat file representation of the data set's organizational structure, in contrast to a richer set of the ocean dataset metadata (e.g., coastal region, temporal constraints, etc.).

With the preponderance of free text and faceted search engines on the web (such as Google [3]), users are becoming increasingly familiar with the simplicity and expressiveness of modern search approaches and expect such approaches to be readily available at any ocean datacenter. However, the aforementioned simple web applications developed by these datacenters often lack the flexibility of a true free search engine. Similarly, the lack of accessible APIs for datacenter access and the lack of expressive metadata via existing services like THREDDS and OPeNDAP severely limits the ability to create query applications that would meet the needs of modern search expectations from users.

The Virtual Oceanographic Data Center (VODC) project directly addresses these issues through the use of proven software technologies and interoperable metadata language descriptors to create a homogenous, full text search-capable catalog of ocean datacenter metadata. VODC leverages the Object Oriented Data Technology (OODT) middleware [4] to instantiate data access and metadata discovery services at each oceanographic datacenter. Once available, these basic infrastructure services are crawled and indexed using Apache's open-source search software (Solr, Nutch, and Lucene [5]) to create virtualized search across these data holdings. This virtualized search is provided as a service, as well as available using an interactive Plone-based [6] search portal. Below, we describe in greater detail our VODC approach including our early experience to date implementing VODC. We round out the paper by pointing the reader to future work.

2. APPROACH

In order to enable an extensible search and query capability across multiple datacenters and data products, VODC has developed a common data dictionary. This comprehensive dictionary allows

WWW 2009, April 20–24, 2009, Madrid, Spain.

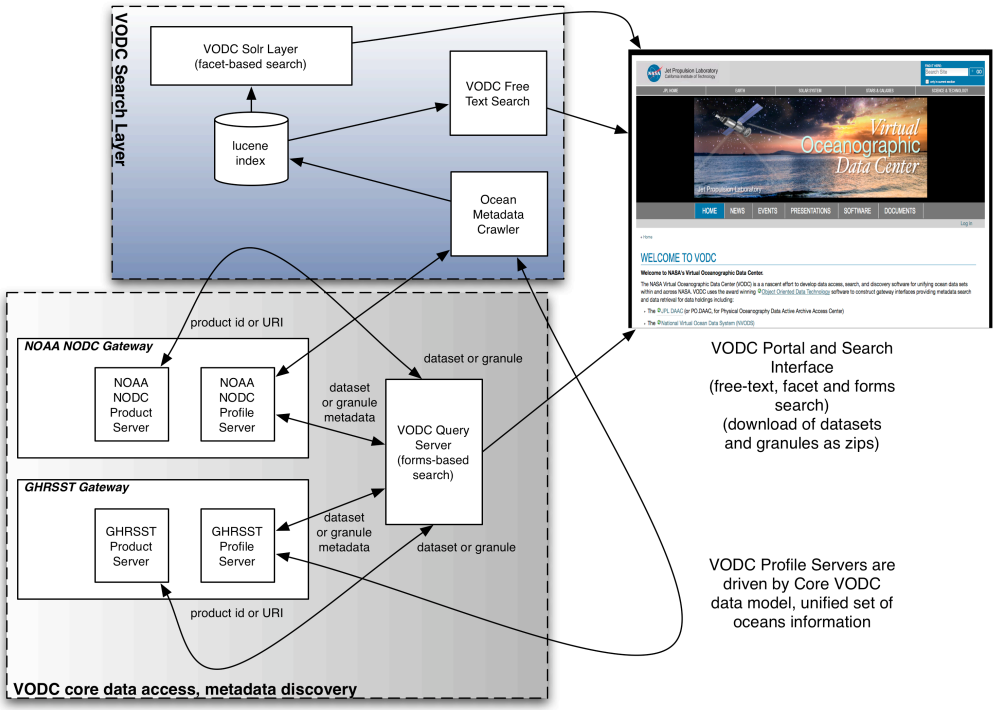


Figure 1. The VODC data access and search layers

the VODC software to transform datacenter-specific metadata into a common ontology, thereby enabling native datacenter-agnostic metadata queries without necessitating runtime translation.

In lieu of accessible APIs for catalog searching and data access and delivery, VODC leverages the OODT Profile and Product Server components, shown in the grey area of Figure 1. The OODT Profile server is a software service that takes in a keyword query and returns a standard W3C resource description framework (RDF) structure using the VODC data dictionary vocabulary. The returned RDF structure describes the underlying resources available that satisfy the provided query. The OODT Product server, on the other hand, presents a uniform software service for accessing and transforming datasets from an underlying datacenter. The combination of the OODT Profile server and Product server formulate a *VODC Gateway*, which enables a heterogeneous oceanographic data center to come online as a standard VODC accessible data/metadata source.

We have developed an Ocean Metadata Crawler component to crawl and index the metadata accessible via VODC Gateways using Apache Nutch (and its standard crawler and parsing framework) shown in the upper periphery of Figure 1. We are currently developing a standard Nutch parser plugin to extract metadata provided by the underlying VODC Gateway. Once the metadata is parsed, Nutch then takes the extracted metadata and indexes it in an Apache Lucene index. This index can then be used, as shown in the upper portion of Figure 1, to enable free-text and faceted search web-services provided by Apache Solr. Solr provides XML/HTTP APIs as well as native JavaScript Object Notation (JSON) for fast web service deployment. The VODC search, access, and retrieval software services are accessible independently, and also provided via our standard Plone portal as shown in the upper portion of Figure 1.

3. FUTURE WORK

The early experience with VODC has focused on global high-resolution sea surface temperature (GHRSST) datasets available via the JPL Physical Oceanography Distributed Active Archive Center (PO.DAAC) and on NOAA's National Oceanographic Data Center. Our future work is focused on adding more ocean data centers and building out client side tools and APIs.

4. ACKNOWLEDGMENTS

This work was conducted at the Jet Propulsion Laboratory, managed by the California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA). The authors would like to acknowledge the support of the NASA ROSES ACCESS program.

5. REFERENCES

[1] B. Domenico, et al., "Thematic Real-time Environmental Distributed Data Services (THREDDS): Incorporating Interactive Analysis Tools into NSDL," *J. Digital Information*, vol. 2, 2002.

[2] P. Cornillon, et al., "OPeNDAP: Accessing data in a distributed, heterogeneous environment," *Data Science Journal*, vol. 2, pp. 164-174, 2003.

[3] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, vol. 30, pp. 107-117, 1998.

[4] C. Mattmann, et al., "A Software Architecture-Based Framework for Highly Distributed and Data Intensive Scientific Applications," In Proc. *International Conference on Software Engineering (ICSE)*, Shanghai, China, 2006.

[5] "Apache Lucene, http://lucene.apache.org," 2009.

[6] A. McKay, *The Definitive Guide to Plone*. APress, 2004.

A Linguistic Solution to Perfecting Search Technology

Fernando Moreno-Torres Camy
MTC Soft
C/ Concepción, 47
18009 – Granada (Spain)
+34.958.215.280
direccion@mtcsoft.es

ABSTRACT

The amount of information available on the Internet is so vast that key words searching returns, in so many times, a huge number of documents. Therefore, the search is not effective, especially if you are using common words.

A solution for this problem is to enrich the texts before indexing, so we can use this additional information to elaborate more selective indexes. This provides search engines new possibilities for filtering.

We have developed a software application based in an exhaustive linguistic analysis. This tool labels the texts with a lot of linguistic attributes, with high quality standards, in a fully automatic way and with low computational cost.

Texts enriched with this software are valuable raw material, much better than plain texts, opening a wide range of new opportunities. Filtering searches on the Internet is one of the more interesting.

Categories and Subject Descriptors

Primary classification: H.3.1 [**Information Systems**]: information storage and retrieval, Content Analysis and Indexing - *Linguistic processing.* Additional classification: H.3.3 [**Information Systems**]: information storage and retrieval - *Information filtering.* E.2 [**Data**]: data storage representations - *Object representation.*

General Terms

Algorithms, Documentation, Standardization and Languages.

Keywords

Search engines, data mining, filtering, parser, labeller.

1. INTRODUCTION

The main problem with today's Internet search engines is that they return too much irrelevant information. The solution to a large part of this problem is to parse the language of the text to be searched, so as search motors can distinguish between different uses of the terms. This requires an application to perform automatically the linguistic analysis of the text in question, a "text labeller program."

Our company, MTC Soft, has developed such a tool, based on linguistic analysis done by a veteran European Commission translator. It analyses the documents and produces enriched texts with additional information, so a search engine is able to index this new raw material from a huge text database (locally or on the Internet). Advanced searches on these documents will obtain accurate results, even when common words are used, words which usually produce poor results in today's search portals. Thanks to this process we obtain a selective filtering of documents.

2. THE TOOL

The principal element of the software is the module that analyses the English text in detail. It then automatically labels the text with linguistic attributes and additional information. No previously inserted metadata is necessary for this process.

This tool obtains from every word in the text:

- The grammatical role carried out in the phrase: verb, substantive, preposition…

- The syntactic role: subject, verb, direct complement…

- Semantic information, "families" (words dealing with the same subject), synonyms…

- The syntactic groupings which form the different parts of each phrase.

Texts are generated in XML format, with the linguistic information in a structured way, to allow further automated processing.

It's important to point out that the connections between words are as relevant as the individual information from every word. The enriched text includes the relations of every word with the others in every concrete phrase. This is very useful information to work with.

3. THE TOOL'S APPLICATION

There are lots of things to do with these enriched texts. Improve the Internet search engines may be the most relevant. To prove the tool, which is ready since several months ago, we have included this technology in our document management application, and **it works**, on our local database. So, we can show

the use of the filtering options in a real test. Obviously, the main application is on the Internet.

3.1 The Objective

The results returned from web search engines include too many documents, most of them not related to the subject we are looking for. Filtering options to reduce the scope and the matches are useless in most cases. We need software with additional possibilities for the users in order to obtain accurate results in the searches.

3.2 How It Works

In this real test, as in the final software, there are three steps:

3.2.1 To Label the Texts

The first step is the most important one because what we finally obtain depends on the *raw material* quality. Poor information texts mean poor results.

So, we use our tool to enrich the texts and store them with all the additional information. It's important to point out that we use our own dictionary and every word is codified, as well as the attributes. This reduces very significantly the size of the database. There are other advantages of using our dictionary, created specifically for this analysis. We have all the writing forms for every word (singular or plural, the verb tenses, and so on), therefore we can include, or not, all the forms in our indexes and searches.

Another improvement is we have additional semantic information, codified in the dictionary, not in the texts. For example, there are word families to deal with. Fruits, buildings, anything can belong to a family. This information gives the final users additional possibilities in the advanced searches that require more specific filtering options.

3.2.2 To Index the Texts

We have a lot of new information from every word, but it is not necessary, nor recommended, to use it all. Instead of creating *all the possible indexes*, we just created the relevant ones. They will permit new searches. Words with no information ("the", "that", "a"…) are not included in the indexes. Since we use a coded words' database, the size of the index is not big, and the search are not slow.

3.2.3 New Advanced Search Options

Finally, when the user of the document management program chooses to make a search, he has a new "advanced search button" which opens a new window with new options. It allows users to include additional filtering conditions to the query.

3.3 A Note About This Test Application

We have not developed in detail this search interface because it's just a sample application of our new tool. There are many works dealing with "searching interfaces" so we have just spent a few days in this part of the software. The real tool is the labeller program.

4. AN EXAMPLE

Let's use a very common situation to illustrate the use of advanced filtering options in a search on the Internet.

If I want to know if Tom Cruise has bought a new house, and I write "Tom Cruise buy house" in the most popular search page, I get almost 4,000,000 web pages, only a handful of these related to my real question.

If the search query is: "Tom Cruise is the *subject* in a sentence where the *verb* is "to buy" and the *direct complement* is "house" (as a substantive, not as a verb)", there is no doubt I would get a much lower number of web pages, but most of them, if not all, would be related to the subject I am looking for.

The way to improve the search is to use a new interface for the "advanced search" page.

In table 1 there is a schematic representation of the way to introduce the user preferences in the query.

Table 1. Advanced search example

Who?	Tom Cruise	The sentence subject	The actor
What does he do?	To buy	The sentence verb	The action
Concerning **what?**	House	The sentence direct object	The object
Where?	In Hollywood	(optional)	The place
When?	Last Month	(optional)	The time

This query would find:

- Tom Cruise has bought a house…

- Tom Cruise and Katie Holmes will buy a new house ….

- Yesterday Tom Cruise bought two dogs and a red house…

But this query would not find:

- Tom Cruise bought a dog and visited a house in …

- Tom Cruise is very famous. He lives in an expensive house…

- Tom Cruise doesn't house any friends, because he has bought a new car and …

Both, syntactic relations between words and grammatical role of the key words, are determinant in text filtering.

Additionally, we can use semantic relations (the "word families") to deal with generic questions. For example, if we use *house*, as a *family*, we obtain:

- Tom Cruise has bought a house / loft / mansion / chalet…

5. TEXT ENRICHMENT EXAMPLE

To illustrate the results we obtain with the enrichment tool, let's see what we have with a very easy sentence. It is important to consider that the additional information is added to the text in a codified way, so a computer application can work with it. The following text is just a simplified representation of the final enriched text.

The original text to analyse is: "My father's house is white and green". The final text looks like this:

First level: grammatical functions: <the[DET]> <house[SUS]> <of[PRE]> <my[DET]> <father[SUS]> <is[VER]> <white[ADJ]> <and[CON]> <green[ADJ]>

Second level: grammatical functions and attributes: <the^{sin} [DET]$^{art+det}$> <house$^{infi+sin}$[SUS]^{sin}> <of^{sin}[PRE]^{otr}> <my^{sin}[DET]$^{anim+per_sin+plu+posv+pri+sin}$> <father$^{infi+sin}$[SUS]^{sin}> <is$^{plu+sin+v_pre3}$ [VER] $^{dur+est+idio+intran+irreg+p\_idio+p\_suj+prin+sin+v\_pre3+conj}$> <white$^{nor+sin}$ [ADJ]> <and^{sin}[CON]^{cop}> <green$^{nor+sin}$[ADJ]>

Third level: syntactic groups: <the house [subject]> <of my father [subject's prepositional sintagma]> < is [verb]> <white and green [direct complement]>

Grammatical plus syntactic information: {<the[DET]> <house [SUS]> [subject]} {<of[PRE]> <my[DET]> <father[SUS]> [subject's prepositional sintagma]} {<is[VER]> [verb]}> {<white [ADJ]> <and[CON]> <green[ADJ]> [direct complement]

6. ADDITIONAL CONSIDERATIONS

6.1 About The Tool

Comparing this tool with previous parsers, we find two important differences with most of them. First, we have developed a software tool, but based in a linguistic analysis. This is a ten years project and only the last three years have been dedicated to the software development. We don't use computers to extracts linguistic rules or patterns. This is a human work, from a very experimented translator. The tool includes this previous human knowledge to analyse the text.

Second, we use little statistical analysis and codify all the resulting text and the additional information, so we do not need big computational resources, neither a lot of time to process the text, nor a lot of hard disk space to store the results.

6.2 About the Test Program

The program we use to test the advanced searches is just a way to check the *real* new tool, the enrichment application. So, it does not look as final user software, but it's good enough to make some searches and verify the success in filtering the results.

The complete solution has three different and totally independent modules:

- The enrichment application, roughly 90% finished. The dictionary has to be completed, as any dictionary, but is a very easy work any user can do.

- The index module, just 15% to 20% finished. There is a lot of information to deal with. We have just used the most relevant, far enough for our test and most of the final user applications.

- The advanced search module, 20% to 30% finished. But we were concerned in functionality, but careless in the interface. We are focused in results, in accurate filtering, not in the final appearance. Is easy to improve our interface, but is not the question to solve now.

So, the enrichment tool is ready to work with other applications. The test program is just for showing the technology.

6.3 About the Semantic Web

There are many people, companies and products related with the semantic web, as there are a lot of works about Internet search engines. In our opinion, all of them have the same limitation: the raw material to deal with is plain text with little or no information at all. Therefore it is very difficult to obtain good results.

If you enrich the *raw material* before starting to work with, you have a lot of new possibilities, a lot of additional information to deal with.

Our tool has two important qualities to be used *at the beginning* of these tasks: the process is fully automatic and the final texts structure is a universal standard. Nobody is going to discuss about substantives, subjects, verbs…

7. CONCLUSIONS

Let's setup a new standard for texts and provide to the Internet tools a new starting point to improve the results in a wide variety of fields, regarding the Web development.

We can improve the accuracy of advanced searches. We can obtain better information from the documents to classify the web pages content. We can do a lot of tasks with better results.

Towards a Semantic Web Environment for XBRL

Sheila Mendez Nuñez
University of Oviedo
C/Calvo Sotelo, S/N
33007, Oviedo, Spain
+34985103287
sheilamendeznunez@gmail.com

Jose Emilio Labra Gayo
University of Oviedo
C/Calvo Sotelo, S/N
33007, Oviedo, Spain
+34985103287
labra@uniovi.es

Javier de Andres Suarez
School of Computing - University of Oviedo
C/ Valdés Salas s/n
33007, Oviedo, Spain
+34985103287
jdandres@uniovi.es

ABSTRACT
XBRL is an emerging standard that allows enterprises to present their accounting reports and other financial information in a XML format, according to the applicable standards. In this paper we present a system capable to add semantic annotations contained in a financial ontology to XBRL reports. Furthermore, we present a new approach which uses probabilistic methods to determine the degree of similarity between concepts of different XBRL taxonomies. This approach will allow investors, enterprises and XBRL users in general, to conduct comparative analysis using reports compliant with different taxonomies, even when they belong to different countries.

Categories and Subject Descriptors
J.4 Social and Behavioral Sciences. Economics.

General Terms
Economics, Standardization.

Keywords
XBRL, Semantic Web, Taxonomy Mapping, Collaborative Web Environment.

1. INTRODUCTION
XBRL is a XML vocabulary for the standardization of accounting reports and other financial information issued by firms and other organizations [1]. Before XBRL would become a standard, accounting information was presented in some different formats such as for example spreadsheets, which are proprietary formats that can be difficult to process.

Specifically, XBRL is made up by multiple taxonomies [2]. Taxonomies are created to standardize financial reports elaborated according to certain Generally Accepted Accounting Principles (GAAP). This leads to different taxonomies for the different countries and sectors of activity. Firms must generate instance documents according to at least one of them. Information contained in two XBRL instances of two similar enterprises from different countries may not be directly comparable because they have been elaborated using different schemas. Moreover, taxonomies must change when a legislative change takes place.

On the other hand, the semantic web can be defined as a long-term vision which pursues the development of technologies that facilitate the automatic manipulation of data published on the Web. Leaded by the World Wide Web Consortium (W3C), a number of semantic web technologies have appeared. Among them, we can highlight Resource Description Framework (RDF), which describes resources using a graph model, Web Ontology Language (OWL), which defines ontologies based on description logics, and SPARQL Protocol and RDF Query Language (SPARQL), which can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. An important feature of these technologies is that they can be neatly combined using several tools and even allowing the system to infer new knowledge using description logics capabilities.

XBRL uses XLink to provide semantic annotations of XBRL documents; however, the development of semantic web technologies has motivated the appearance of a new approach applying those technologies to XBRL. In this paper, we present a project that uses a financial ontology with about two hundred classes. This ontology is applied to XBRL instance documents to provide these semantic annotations.

Last, we present a new approach to make possible that anyone can compare XBRL instances no matter the taxonomy they are compliant with. Even if two taxonomies are not directly comparable, we propose a system that uses probabilistic methods to determine the degree of similarity between them. This system is described as a proposal for future work.

2. PROPOSED SYSTEM
There are some prior proposals which relate XBRL and the semantic web [3]. By now, the number of proposals is increasing with some efforts like XBRLImport, which is an open source project that will translate XBRL data into turtle RDF syntax [4] and several interest groups like “XBRL Ontology Specification Group” [5] and the “XBRL and the Semantic Web Interest Group” founded by the W3C in May of 2009.

The proposed system is basically composed by a financial ontology and a Java integrator program. The design of the financial ontology allows the user to make the real conversion between XBRL reports and semantic web. XBRL reports can be parsed to RDF (Resource Description Framework) [6] compliant with the designed ontology. The system can manage collections of RDF documents. Once we have one or more RDF documents, we can perform SPARQL queries over all of them with the SPARQL query editor included in the system. This SPARQL editor has options such as loading preconfigured queries, saving user edited queries and also allows the user to export the result in a XHTML format. Figure 1 shows a screenshot of the queries editor.

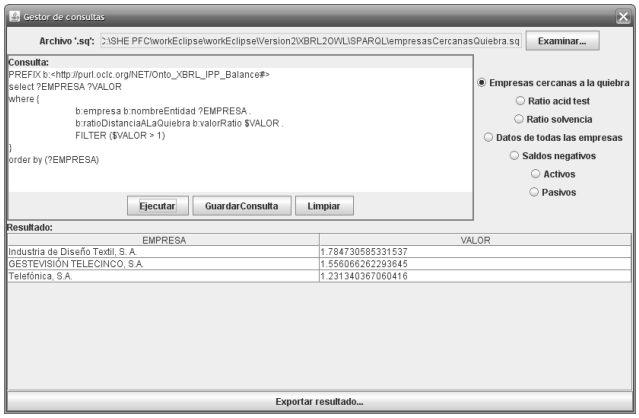


Figure 1. System SPARQL query editor

In Spain, the Spanish Securities Commission (*Comisión Nacional del Mercado de Valores*) (CNMV) is the governmental body that requires listed companies to submit XBRL reports containing financial information and that publish them through its Web site (www.cnmv.es). CNMV also provides a useful tool for the retrieval of XBRL reports by different criterions. Reports downloaded using this tool are the input of our system.

In a second phase, XBRL reports are transformed using XSLT (eXtensible Stylesheet Language Transformations). The output of this transformation are the RDF documents that make up the knowledge base managed by the Java system.

The knowledge base contains all the RDF files generated and gives the Query Manager the ability of performing searches over all of them. Queries are performed using the SPARQL query engine provided by Jena.

The architecture of the implemented system is shown in figure 2.

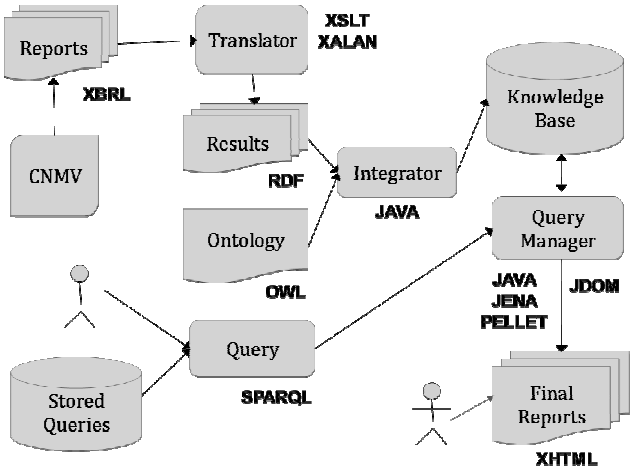


Figure 2. Architecture of the implemented system

3. FUTURE WORK

Our proposed system will be a collaborative web environment capable of the generalization of concepts from different taxonomies according to a generic ontology of financial concepts [8]. Figure 3 shows the architecture of the system.

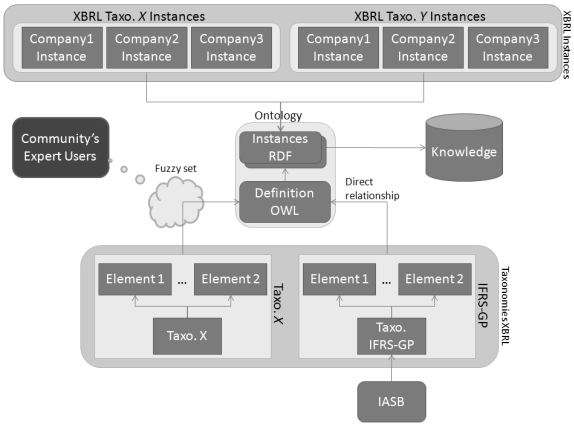


Figure 3. Architecture of the Collaborative System

As each XBRL jurisdiction has defined its own set of taxonomies, according to the financial reporting requirements in each country, the number of financial items is too high to be manually translated into the ontology. As a solution for this problem, we propose a collaborative environment, in which users (experts in the domain of the problem) will define the relationship between an item which is specific for a certain taxonomy and the corresponding concept in the ontology. Once this relation is defined, it will be incorporated into the knowledge base of the project, allowing the rest of the users to take benefit from it.

For the definition of such relationships, we must bear in mind that the differences in the reporting requirements across countries and sectors of activity imply that in many cases users can not establish a relationship of equivalence between an item belonging to certain taxonomy and the concept defined in the ontology. To model and represent these relationships we intend to use fuzzy sets. Applying the properties of fuzzy logic, the relationship between an item of a given taxonomy and the corresponding element of the ontology will be represented using a value which belongs to the range [0, 1]. As different users may assign different values to the degree of relationship between the items and the concept, statistic (i.e. weighted averages) and Artificial Intelligence (i.e. self-organizing maps) procedures will be used for knowledge representation.

So, the main goal of this part of the project is to design an OWL (Web Ontology Language) [7] ontology, which on the one hand defines groups of elements from different taxonomies which are common (or at least very close) and on the other hand is extensible, so the experts in the Accounting domain can add new knowledge to the system, by adding an element from a certain taxonomy to one of the existing groups or creating a new one.

Once this system has been developed and properly tested, we will upload the results of our research to the project's Web site: http://www.semanticxbrl.com, which will also contain all the relevant information about the evolution of the project.

4. CONCLUSIONS

The application of semantic web technologies to XBRL instance documents and taxonomies is a new approach to the analysis of accounting information. Under this approach, the semantics of the analyzed data plays a leading role. It would be very important for potential investors to have access to global databases with financial information having a high level of homogeneity.

Comparisons between reports which are compliant with different taxonomies are difficult. This project seeks to add a new level of

abstraction, allowing the users to work with XBRL instances regardless of their origin. We expect that the combination of uncertainty methods with description logics will provide the proposed system with the capacity to define the degree of correspondence between concepts from different taxonomies.

As issues which would take benefit from this approach, we can mention the bankruptcy prediction problem. Furthermore, there are also other tasks which are closely related to the assessment of the solvency of the firm and could be improved [8], such as for example the bond rating problem and the issuance of the auditor's going concern opinion.

5. REFERENCES

[1] eXtensible Business Reporting Language (XBRL) 2.1, http://www.xbrl.org

[2] XBRL Financial Reporting Taxonomies, http://www.xbrl.org/FRTaxonomies/

[3] Lara, R. XBRL Taxonomies and OWL Ontologies for Investment Funds, in Advances in Conceptual Modeling - Theory and Practice, 271-280.

[4] Dave Raggett, XBRL Import, http://xbrlimport.sourceforge.net/

[5] XBRL Ontology Specification Group, http://groups.google.com/group/xbrl-ontology-specification-group

[6] Resource Description Framework (RDF), http://www.w3.org/RDF/

[7] OWL Web Ontology Language, http://www.w3.org/TR/owl-features/

[8] Méndez Núñez, S.; Labra Gayo, J.E.; De Andrés Suárez, J.; Ordóñez De Pablos, P.A Semantic Based Collaborative System for the Interoperability of XBRL Accounting Information, in Miltiadis D. Lytras, John M. Carroll, Ernesto Damiani Robert D. Tennyson (Eds.), Emerging Technologies and Information Systems for the Knowledge Society, Springer Lecture Notes in Artificial Intelligence 5288, pp. 593-599

Porqpine: a Distributed Social Search Engine

Josep M. Pujol
Telefonica Research, Barcelona, Spain
jmps@tid.es

Pablo Rodriguez
Telefonica Research, Barcelona, Spain
pablorr@tid.es

ABSTRACT

We describe a distributed social search engine build upon open-source tools aiming to help the community to *take back the Search*. Access to the Web is universal and open, and so the mechanisms to search should be. We envision search as a basic service whose operation is controlled and maintained by the community itself. To that end we present an alpha version of what could become the platform of a distributed search engine fueled by the shared resources and collaboration of the community.

1. INTRODUCTION

PQ intends to serve as the cornerstone of an alternative approach to search that puts the emphasis on distribution and on the social dimension of their users. **PQ is inherently social.** Crawling, indexing and ranking of the Web is carried out collaboratively as users browse and search without their explicit intervention. Not requesting feedback from the users frees us of the main problem of collaborative systems: lack of collaboration fades out since collaboration have zero cost. PQ does not rely on explicit feedback to operate, therefore, it remains totally transparent to the user. We believe transparency is a major feature. First because it lowers the entry barrier for those people who do actively engage in collaborative systems. Second, because it does not interfere with those users who already have habits that do not wish to change. A user should not learn PQ, but PQ should learn from her and share her snippets of knowledge and experience among her social circle while providing privacy and anonymity. PQ only considers collaboration among friends or acquaintances, which are defined by the user's social network. This limits the accessible Web by means of searching, but the loss of recall comes with an increase of personalization, context awareness and byproducts such as serendipity and social glue. Another reason to limit the search to your social network is for efficiency and scalability as it limits propagation. **PQ is distributed.** The infrastructure required for a search engine already exists in the form of thousands of idle desktops and extensive residential broadband access. In PQ everyone runs their own local search engine. This distributed setting is not only more environmentally and economically sustainable. But it also leverages the end-user computing power to carry out functionalities such as contextual filtering or inline recommendation, which would be expensive to scale in a centralized architecture.

2. PQ'S ARCHITECTURE

WWW2009, April 20-24, 2009, Madrid, Spain.

Figure 1 depicts the architecture of PQ. There are 3 independent components that can run either in the user's desktop or distributed across several machines. The *pqbar* is a Firefox addon, which monitors browser events to detect browsing, searching as well as user actions while browsing (used to infer the page's relevance). The relation between *pqbar* and *pqnode* is N:1 as the user might access the Web from different machines. The *pqnode* is the personal search engine of the user. It is a Ruby process composed of a mongrel servlet for communication, the HPricot based parser, the Ruby Ferret indexes for pages and search sessions and the social network module. This module obtains the social network from different sources such as Facebook, Twitter and email accounts. It aggregates all sources into a single ego-network and calculates the community structure to find clusters. The *trusted proxy* component, also implemented in Ruby, consists of a mongrel servlet and the modules responsible of forwarding search requests and aggregating results. All communication between components is done via WS API with parameters encapsulated in JSON. The relationships between components is better described with the following two use cases.

2.1 Feeding your PQNode

Every time a user browses a page, the *pqbar* will send the url to the *pqnode* which will *wget* the url, parse the content and index it on the Ferret page index. When the url is not a standard web page but it is the result of a query (detected by the *pqbar* via regular expressions) the *pqbar* will start a search session. Once completed, the *pqbar* sends to the *pqnode* the query and the set of results browsed by the user. This set contains the order of visits, time, depth, navigation graph and the actions performed on the web page. Once the search session is sent to the *pqnode* it is stored in the search index. We mentioned that *pqbar* also reports the actions performed by the user on a web page. These actions range from trivial commands such as print, save, control+copy to more sophisticated and less noisy estimators of the user's interest on the page such as adding it to Firefox's bookmarks or sending it to sites such as Delicious or Reddit. *PQbar* can detect by checking Firefox's load event and analyzing the GET request parameters. Whenever the user sends a link to a services such as social bookmarking, online social networks, news aggregator, etc. the *pqbar* takes notice. Note that the scrapping of the page content is performed by *pqnode* instead of the *pqbar*. This introduces some network overhead but it guarantees that no private content rendered in the browser will ever be indexed.

PQ users contribute to the system without work overhead and in a total transparent way. Content in PQ is indexed in real time. As soon as one user in your social circle sees

a page, the page will be immediately available to you. How much information can be collected by users? As moderate PQ user collected 61Mb worth of data, 4863 unique web pages and 1640 queries, after 3 months of use. A crawling robot is able to index that much in a matter of hours. However, pages indexed by the user's *pqnode* are actual web pages that were interesting to her. Additionally, the pages indexed by the user might not be visible for the robot for a while. Transmission of links take place in many channels that robots do not have access to, e.g. emails, instant messages, facebook post and tweets to mention just a few.

2.2 Searching the Web through PQ

Search in PQ can be triggered either by querying directly your *pqnode* (e.g. via PQ web page) or — if the embedded search is enabled — by submitting a query to a search engine (e.g. Google, Yahoo, isoHunt, etc.). In the embedded case, results will be displayed side to side to the original search engine as depicted in Fig. 2. Search in PQ consists in:

Local search Each *pqnode*, upon receiving a search request, queries the Ferret page index. The TDIDF scores returned by Ferret are factored out as they are noisy due to our basic scrapping and the to the limited user's corpus that affects the IDF calculation. The basic relevance score of the page is a function of the hits on the snippets, the actions of the users on the page, the relevance of the incoming links, the number of visits and its freshness. Additionally, the query is also submitted to the query index to retrieve the top-20 similar queries. We check if an url from previous search results is also part of the current result set. If the url is also the quality result for a past query then the page's score is increased according to the distance between queries. We can estimate the quality of a search result because the search index records the browsing pattern, time and the actions of the user during that search session. The distance between the current query and the top-5 similar queries is used to assess the user expertise on the query.

Search propagation The search propagation is managed by the *trusted proxy* that receives the query, the social network and the community partition from the *pqnode*. The proxy uses the community partition to differentiate between the clusters within the *pqnode's* social network. The proxy queries a random sample of the neighboring nodes (up to 20 request) and then targets the cluster that returned more hits with the remaining 20 requests.[1] The results are finally aggregated and sent back to the *pqnode* that cannot identify which *pqnode* provided a particular result.

Ranking Upon receiving the results from the proxy, the *pqnode* ranks them using a basic collaborative filtering (using the expertise and the relevance score) weighted by the popularity and freshness of the page in the set of results from your social network. Finally, the results are sent back to the *pqbar* for display. At this stage, computationally intensive processes could be performed without an impact on the performance of the system. For instance, results could be clustered or rerank according to the content of other open web pages to introduce some level of context.

3. PRIVACY AND ANONYMITY

Unlike the draconian ToS of many systems that handle user's data, PQ operates in two simple premises. 1) The data belongs only to the user and she has absolute control over it. Therefore, we do not use aggregated data The *pqnode*'s role is to encapsulate and to isolate the data to guarantee privacy. And, 2) all contributions to the community are anonymous unless stated otherwise by the user. The search request operation could be used be a malicious agent to probe the content of other *pqnodes*. To avoid this misbehavior that would jeopardize privacy we enforce communication by proxy, although there are other alternatives.

4. DEPLOYMENT AND SCALABILITY

PQ is designed to require minimal infrastructure, the *pqnode* computation, storage and bandwith is carried out in the user's computer. Since search only propagates one step further thanks to the limits introduced by the social network, latency is kept on check. The *pqnode's* resources footprint is small (given that is a Ruby process). For the user mentioned before the local search takes between 0.05 to 0.2 seconds depending on cache misses. The network traffic overhead is small, the worse case scenario with 100 queries per day with a full neighborhood (40) would have an impact of 66MB/day or an average bandwith reduction of 0.4KB/s. The only limiting factor of the *pqnode* would be its memory footprint of 60 to 100MB. Ruby is quite a memory hog. After starting *pqnode* the process already allocates 48MB, 90% of them due to the mongrel servlet. Using ruby for the *pqnode* reference implementation in a stable release should be seriously reconsidered. The trusted proxies only operate on the control plane so few servers can host enough proxies for tens of thousands users. The cost of the proxy solution represents a marginal fraction of the cost of a hosting the data plane. Furthermore, proxies could be hosted by normal users provided they were trusted by the rest of the community.

Current deployment. Since the auto-update feature of the *pqnode* is not yet available we are temporarily hosting the user's *pqnodes* processes on the PQ cloud at Amazon EC2. However, this is only a temporal solution until the auto-update feature of *pqnode* is released. The temporal setting, however, poses a scalability problem due to the large memory footprint of the *pqnodes*. Although this problem will be reduced once the users' *pqnode* processes are back to the users' computers, it will not completely disappear. A non trivial number of users will suffer from low availability and suboptimal network connections. Offering a cost-wise solution to these users is perhaps our biggest challenge.

5. CONCLUSIONS

We presented that anatomy of Porqpine, a working prototype of a distributed social search engine targeting to take the search back to the people. This search engine is far from finished, and hopefully it will never will as the community engages on extending it.

[1] If the user does not provide her social network or she does not have an acquaintance within the system the proxy will use a random set of *pqnodes* instead. The quality of search results, however, will be drastically reduced.

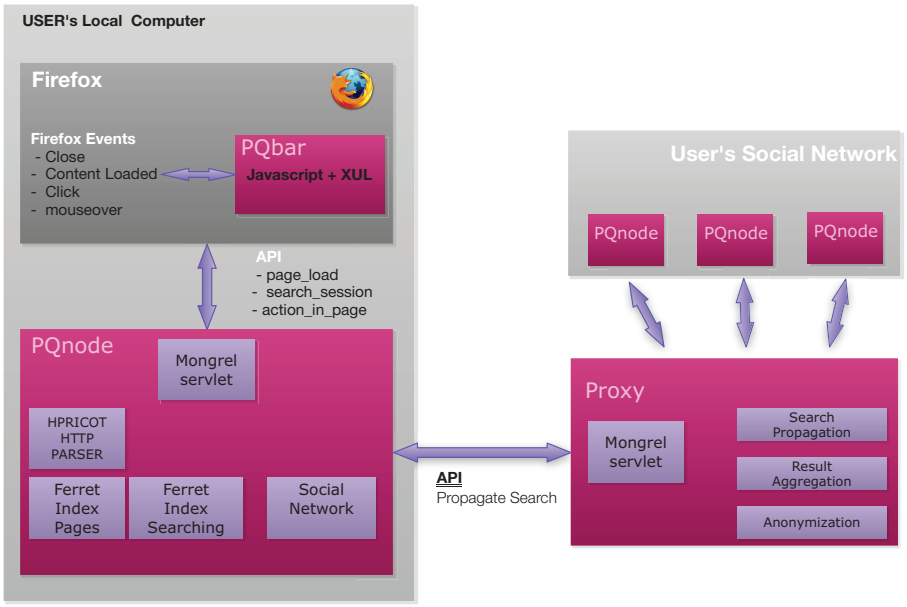


Figure 1: Architecture of PQ

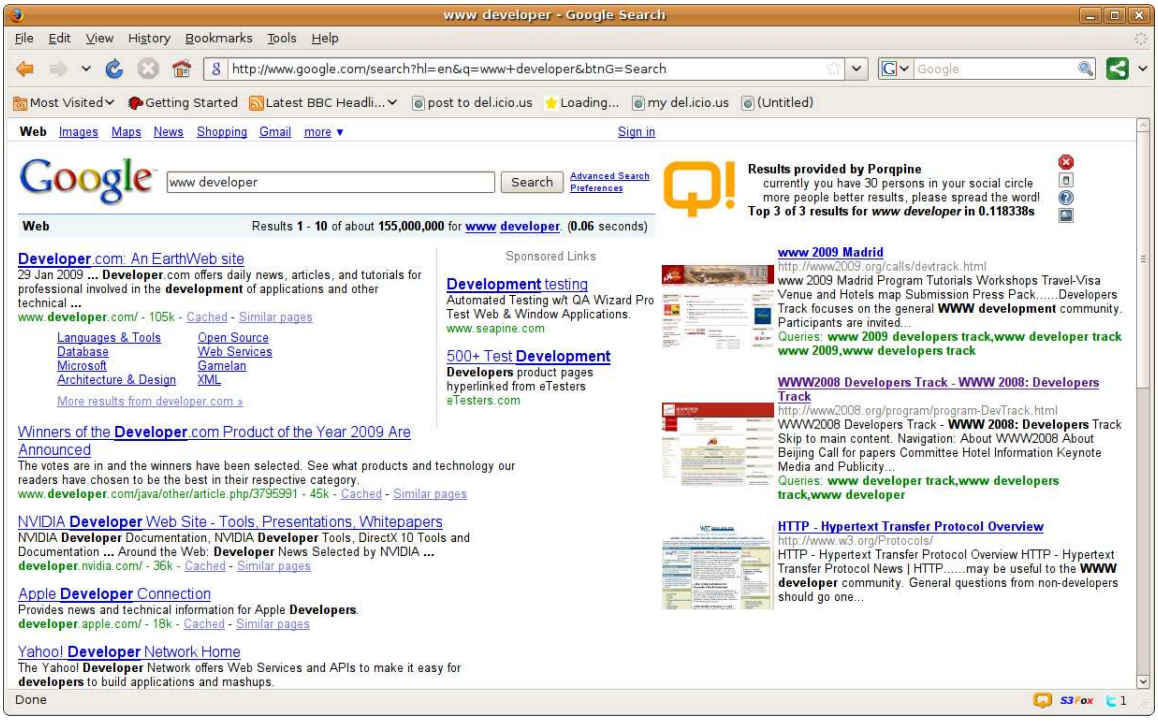


Figure 2: Results of the search *"www developer"* to Google and PQ embedded search. The search is replicated automatically by the *pqbar* whenever the user searches in a search engine (not only Google). Results from PQ come from past searches and browsing of member of her community and herself. If the PQ had no results, the system would learn from the current search session, and consequently, another user of the same community would find relevant results on the next query.

The Web, Smart and Fast

Olivier Rossel
Datao.net
8 place André Abbal
31100 Toulouse
Tel: +33 (0)6 16 39 45 44
datao@datao.net

ABSTRACT

The Web is considered literature by many users, who just want to access pertinent data sets as fast and as smartly as possible. This paper presents the *Smart Web* paradigm, that unifies data management, the Web infrastructure and end-users expectations. First, we define the user's expectations of what a browsing session should be. Then we identify some technical locks in the current architecture and implementations, and propose possible solutions. Last, we introduce *DataO*, a Java/Firefox web browser platform, that was used to prototype some of those technical solutions, for a smart and fast browsing session.

Keywords

Experimentation, Web Browser, Semantic Web, Browsing session

1. INTRODUCTION

Very few metrics are available to measure the global satisfaction of web users. Given the massive adoption of this technology in companies and homes, it is commonly admitted that the current situation is optimal, and that major improvements are not required.

Nevertheless some tasks on the Web are still very fastidious and frustrating. Achieving a complex goal involving several data sources and service providers quickly highlights the limits of the Web as a ubiquitous service provider. A typical example of such complex tasks is the "organize my holidays" use case.

Usually, it is up to the user to compensate for the limitations of the Web with his or her own intelligence and patience. Alternatively, if a goal is identified as common to many users and if there is an economic interest in it, service providers may develop a custom architecture to process and achieve that goal.

2. THE SMART WEB

To improve the user experience on the Web, it is important to understand what the user expects from the system and how this expectation may mismatch the existing implementation. Then it is possible to propose new features that may bridge the gap without breaking the existing nor degrading the global performance.

This continuous improvement will lead towards a *Smart Web* that permits unexpected new usages.

Our proposal is to formalize the browsing session as expected by the end-user, and identify the technical locks in existing implementations that create the mismatch.

3. THE BROWSING SESSION

3.1 The mental model

Web users usually have a rough mental model of a goal to achieve on the Web. Depending on their technical skills, they adapt that mental model to the tools available to them, i.e. mainly the Web browser and the available Web sites and services.

3.2 The Q&A paradigm

From the user's perspective, the Question and Answer model (Q&A) is pretty natural when dealing with a source of knowledge. The question captures some of the intention of the user. And the answer is expected to precise that intention or to resolve some of the pending issues for the achievement of that intention.

3.3 The personal knowledge store

But, because no single omniscient source of knowledge is available, no answer is ever complete at once. The only way to circumvent that truth is to apply the Q&A process iteratively and choose each next knowledge source in order to create a unique *local knowledge store* that happens to be the most accurate knowledge referential for the actual intention.

3.4 Summarization for a final answer

The goal of the 'Q&A' is to build the *local knowledge store* and then summarize a final answer set that fully and unambiguously solves the issues pending in the achievement of the user intention.

3.5 Qualifying the answer

Considering that the refinement process can be endless, it is up to the user to define his or her level of expectation for what he or she considers as an acceptable final answer. The refinement process will then fill the *local knowledge store* by various techniques until it contains all the pertinent data to infer the final answer set.

3.6 Iterating the Q&A

This process is then iterated and updated on and on, for the same user's intention, or merged with another user's intention.

3.7 The formulation of partial answers

The correct formulation of answers is crucial to insure their mashup in the *local knowledge store* which will ultimately infer the final answer.

Because the user does not know the content of any answer beforehand, it is up to the knowledge source or an intermediate process to insure that the answer is pertinent and is formulated in a "processable" manner.

The lack of this step often creates inconsistencies in the *local knowledge store* that lead to misconception, cognitive overload and ultimately an improper processing of the final answer. The process must then be discarded and done again.

4. THE WEB BROWSING SESSION

Today, interaction with the Web does not match the mental model of the user that we described above. We will not discuss about user interfaces, but focus on technical locks that participate to that mismatch.

4.1 A Web of document

A document is a set of data formatted in the most natural manner (depending on the data substance, it can be spreadsheet, formatted text, drawings). It presents a full coherent set of data. This feature permits the user to create connections between data that he or she would not have expressed but that are consistent with his or her intent.

4.2 Syntactic data format

HTML as the medium on the Web is very adapted to formatting, transmitting and displaying documents. It permits a simple authoring process (mostly a text editor is the only requirement) and the standardization process of the W3C has led to a ubiquitous document formatting system. Unfortunately, HTML is based on a tree structure which forces a fixed order for information serialization. It also mixes organizational and displaying information with the core data the document carries.

4.3 Server-driven browsing

The Web has evolved from a Web of document to a Web of services (i.e. the *Web 2.0*). It is now possible to achieve a given set of intentions by using custom architectures built by service providers and exposed via Web sites as HTML pages, HTML forms and server-side managed workflows. Unfortunately, very few integration *between services* has been done, and any orthogonal user intention becomes very tedious and error-prone to achieve.

4.4 Stateless browsing

For efficiency purpose, the HTTP protocol prevents advanced stateful management. Most efforts in the first years of the Web development have been to circumvent that design in order to provide human-oriented services with minimum cognitive effort. At the moment, this issue is solved by advanced technologies, such as continuations and server sessions.

5. THE SMART BROWSING SESSION

The technical details presented previously lead to a mismatch between the user's expectation of a browsing session and the technical implementations available. The following technologies and paradigms may help to build a better architecture for the Web to match user's expectations.

5.1 A Web of documents *and data*

The problem of a Web of documents is that boundaries are predefined by the authors in the knowledge stores. The user or any intermediate process cannot redefine the boundaries to match a relevant data set, usually because those boundaries would be orthogonal to the documents boundaries defined by the authors or would break the tree structure materialized in the HTML.

From that perspective, RDF offers an optimal granularity that permits to define any given data set for any given need with no definition of boundaries.

Hopefully in the near future, a standard system will permit to mix a Web of document and a Web of data.

5.2 Semantic data

The effort of the W3C with the Semantic Web is to provide a ubiquitous format to exchange information on the Web. RDF and OWL provide several new features that are currently missing or very difficult to achieve with HTML.

Those features are : graph-based modeling, automatic aggregation of data sources, adaptable data set boundaries, a global strategy for unique IDs, binding between data sets and conceptual descriptions (i.e. ontologies), etc.

5.3 Intention-driven browsing

The first critical step of any service on the Web is to understand the user intention. But delegating that intention management to disparate remote services ultimately leads the user never to fully express his or her own intention, and stick instead to a partial need that was designed *a priori*.

The development of *intention capturing* features on the client-side is crucial to unlock that situation, and a first step toward intention-driven browsing *across* services and data silos.

5.4 Stateful browsing

Stateful browsing is much more than server sessions. Only the user (and more generally the client side) can fully capture his or her intention and maintain a cross-site and cross-services session along with the corresponding *local knowledge store*. Because of the poor session management in Web browsers, an additional cognitive effort is necessary to maintain that session and the *local knowledge store* in the user's own mind, and more generally to manage the whole Q&A session.

6. PROTOTYPING THE SMART WEB

Some technical issues in the current Web architecture can make complex browsing sessions to become tedious and error-prone. As a prototype, we developed *DataO*, a smart Web browser that extends the capabilities of a modern web browser (Firefox / XulRunner) by running it inside a Java platform environment. This prototype is a nice playground to override any component of the Web browser with our own implementation (in any language available on the Java platform), or plug any Java-compatible software library in the browser internals.

6.1 Capturing User Intention

6.1.1 The user as the context

A session in *DataO* begins with your login, meaning that the browser will also load your profile. A profile is an ontology describing your interests, your pending sessions, your favorite mail repository, your calendar, your contacts, etc. That entire context is the first source of information for the *local knowledge store* and is used to initiate a *smart session*.

6.1.2 The question as the context

Defining the precise question to be solved in the current browsing session is the mandatory preliminary step to achieve. *DataO* provides rich GUIs to capture the user's question (i.e. its intention): graph views, automatic proposals, questions repositories browsing, extending previous questions, keyword-concept searches, etc.

To keep control of the whole browsing session, *DataO* does not delegate the question capturing to remote services. It uses the remote sources (the Web of Data, external web services) only to fill its *local knowledge store* and processes the *smart session* on its own.

6.2 Processing the user intention

6.2.1 External semantic discovery services

In its current form, *DataO* manages local data as ontologies and RDF graphs. These flexible formats allow fast prototyping and very agile data management and inference. External data sources required to extend the *local knowledge store* are identified thanks to external semantic discovery services such as SIndice[1], Watson[2] or SPARQLPedia[3].

6.2.2 Internal Semantic processor

DataO manages import, aggregation and alignment of remote semantic data thanks to an internal semantic processor that implements the W3C semantic stack. This stack proposes additional features on the browser: local knowledge store synchronizing with remote data services, local inference of the knowledge store, web workflows discovery/aggregation based on inference of the local knowledge base, local workflow engine, semantic-based HTML template aggregation, etc.

6.3 Presenting the final answer to the user

6.3.1 Advanced HTML widgets

The integration of Firefox/XulRuner inside the Java platform allows to closely monitor interactions between HTML widgets and the user. For example, the HTML textarea widget can be used in conjunction with a *Controlled English* plug-in to provide additional features, such as contextual drop-down lists to auto-tag free text with semantic metadata.

6.3.2 RDF-HTML Template repository

The end-user interface in *DataO* mimics the Web browser (HTML, clickable links, URLs) and maps those concepts to the semantic processor. The web pages presented to the user are façade to interact with the *local knowledge store*. They are built on the fly by the semantic processor and the local workflow engine which are driven by the context (user profile, completeness of the *local knowledge store*, details of the web workflow, etc).

7. CONCLUSION

In this short paper, we have presented some issues with existing Web browsing implementations, especially Web browsers. And we propose a prototype of a next-generation Web browser, *DataO*, which provides additional features on the client-side to overcome those issues.

DataO is a agile web client platform, based on the integration of Firefox/XulRunner in the Java platform. It is a work in progress. At the moment, we concentrate on adding semantic features to *DataO*. But other different R&D approaches to improve the web client in different manner could be applied too (for example, NLP, P2P, mobile code and agents, etc).

For more information about the *DataO* smart browser that we are developing, feel free to visit the website of the project:

http://datao.sourceforge.net

---

[1]: http://sindice.com
[2]: http://watson.kmi.open.ac.uk
[3]: http://sparqlpedia.org

The Web as a Content Management System

Patrick Sinclair, Nicholas Humfrey, Yves Raimond, Michael Smethurst, Tom Scott

BBC Audio and Music Interactive
Henry Wood House, 3-6 Langham Place,
W1B 3DF, London, UK
+442077654640

{firstname.lastname}@bbc.co.uk

ABSTRACT

In this paper, we describe the BBC Music Beta, providing a comprehensive guide to music content across the BBC. We publish a persistent web identifier for each resource in our music domain, which serves as an aggregation point for all information about it. We describe a promising approach in building web sites, by re-using structured data available elsewhere on the Web --- the Web becomes our Content Management System. We therefore ensure that the BBC Music Beta is a truly Semantic Web site, re-using data from a variety of places and publishing its data in a variety of formats.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Data sharing

General Terms

Management, Design, Human Factors.

Keywords

Semantic web, Linked Data, Music, API

1. INTRODUCTION

The aim of the BBC Music Beta [3] is to provide a comprehensive guide to music content across the BBC, linking information about an artist to those BBC programmes that have played them. The first step is to provide detailed information about artists who appear on BBC programmes, have had an album reviewed on the BBC Music site or have been covered on BBC News.

Rather than maintain our own source of music information, we are using existing, open repositories: MusicBrainz [4], an open source community-maintained database of music information, and Wikipedia [8].

This paper describes how by using these repositories we are using the web as a content management system, effectively enabling anyone to indirectly contribute to the BBC Music Site, while also helping to create links between existing URIs. We also describe how MusicBrainz has allowed us to automatically link one of our richest assets, BBC News stories, to individual artists using the hyperlinks, and how this is in turn encouraging a richer web.

2. BBC MUSIC BETA

The BBC Music Beta is a major rewrite of the existing site, in redeveloping the service we set out with a clear objective to design and building the service around the primary objects within the music domain and integrate those with the other BBC domains our audience is interested in, namely programmes, events and users.

The primary music objects are: artists, releases (and their reviews) and labels, for each of these we are working to provide persistent URIs. To ensure that we minimise the number of web identifiers we have reused those provided for Musicbrainz. Our hope in doing this is that we make it easier for others to link to and reuse the data we are publishing.

Once these persistent web identifiers for the different types of resources in our domain are set up, the most difficult step towards a Semantic Web site is achieved. Different representations for each of these resources can suit different types of user agents, from traditional web browsers to more sophisticated user agents making use of structured web data. Our HTML documents are designed for the earlier, whereas our RDF documents are designed for the latter. We automatically deliver the representation suiting a particular user agent's need using HTTP content negotiation.

For example, for an artist on the BBC Music Beta, we consider the following web resources:

The artist itself, where `:guid` represents it's MusicBrainz identifier (e.g. `a3cb23fc-acd3-4ce0-8f36-1e5aa6a18432`):
`http://www.bbc.co.uk/music/artists/:guid#artist`

A document about the artist:
`http://www.bbc.co.uk/music/artists/:guid`

Several content-negotiated versions of that document, e.g.
`http://www.bbc.co.uk/music/artists/:guid.html` and
`http://www.bbc.co.uk/music/artists/:guid.rdf`

These web identifiers ensure we follow the principles outlined in [9], whilst not requiring any HTTP redirects, which would significantly increase the traffic on our website. Both the HTML and the RDF documents provide links to further web identifiers, e.g. programmes in which an artist has been featured, a corresponding artist in DBpedia [1] or an identifier for the "music artist" concept in the Music Ontology [6].

In addition to those primary pages we also identified a large number of additional resources -- fragments of a page if you will -- that are also assigned their own URI. Each document, each page, is then composed by transcluding the relevant set of resources into the document. For example, the URL for an artist at: `http://www.bbc.co.uk/music/artists/:guid`

is in turn composed of the following resources:

```
http://www.bbc.co.uk/music/artists/:guid/news
http://www.bbc.co.uk/music/artists/:guid/reviews
http://www.bbc.co.uk/music/artists/:guid/links
http://www.bbc.co.uk/music/artists/:guid/labels
```


WWW 2009, April 20-24, 2009, Madrid, Spain.

This approach has the advantage that different representation can transclude a different set of resources i.e. the URIs remain the same while the set that makes up any given representation may vary (as will the document format). For example the mobile (XHTML MP) representation excludes the wikipedia biography whereas the desktop (XHTML) does include this information. We of course recognise that this approach may be considered controversial since the approach is not allowed within the HTTP spec. However, we believe that it is an acceptable compromise since it does improve the user experience while retaining one URI per concept.

2.1 Web as a Content Management System

On the BBC Music Beta, there are three sources of information: MusicBrainz, Wikipedia and the BBC. MusicBrainz is used as the backbone of the site, providing data such as artists' releases, relationships with other artists and links to external websites. Wikipedia is used for artists' biographies. The BBC provides additional material, such as an image, album reviews, details about which programmes have played that artist and links to featured content elsewhere on the BBC site.

To obtain data from MusicBrainz, we are using their replication mechanism [5]. This consists of SQL change event packets delivered hourly over FTP, which we download and apply to our own local MusicBrainz database. We then use the Wikipedia article link for each artist provided by MusicBrainz to fetch the Wikipedia article synopsis that forms the artist biography. To keep the Wikipedia text up to date, we have developed a system that tracks the Wikimedia recent changes IRC channel [7] that is updated whenever an article is created, edited or deleted. When we track an edit to an article linked to from MusicBrainz, the system downloads that article and stores it in our local database, allowing us to keep the Wikipedia biographies up to date in real time.

The use of MusicBrainz and Wikipedia to provide the underlying data for the site has allowed us to cover a much wider range of artists than would otherwise be possible - it is beyond our resources to maintain a biography for every artist heard on the BBC. It is also ensures that the data is kept up to date and doesn't go stale. For instance, when an artist dies their profile is updated within a few hours by the community and reflected on our site.

The BBC is also an active member of both the MusicBrainz and Wikipedia communities, at the time of writing the BBC's music team has contributed over 2,800 edits to MusicBrainz. Many of these edits include content about new or specialist music but a significant number of edits includes updating links to other sites. The addition of these links is important because it helps the web aggregate more resources around those concepts. Thus the contributions of BBC staff means that not only does the BBC benefit, so does the quality and quantity of data at those services, but possibly more importantly the web at large benefits because it is more coherent, more linked and so more easy to navigate.

2.2 Automatically Linking Artists and News

On many of the news stories published on BBC News journalists add related Internet links. If a story covers a music artist, it might link out to their home page, their MySpace site or even a Wikipedia article. In MusicBrainz, artists can have several URLs associated to them. By simply cross-referencing each link on a news story with the URLs in MusicBrainz, when we find a match we can confidently say that the news story relates to the artist associated with that URL.

For example, a news story covering Madonna links to her homepage at `http://www.madonna.com/`. When we look up the `http://www.madonna.com/` in MusicBrainz we find it associated with the MusicBrainz artist Madonna, allowing us to link to that news item from Madonna's artist profile. By tracking an RSS feed and checking the links for each story we can generate a news feed with RSS for any artist.

This simple technique is completely automatic and will perform across any of the 400,000 artists in MusicBrainz. As it is based on matching links added by BBC editors we can be very confident that a news item will be associated with the correct artist. We also believe that it adds value to the web, as BBC editors will be encouraged to add useful links from news stories so that these can be aggregated on artist profile pages. Editors from BBC Music will play a more active part in maintaining artist links in MusicBrainz instead of manually associating artists with news items, improving the data quality in MusicBrainz.

We would like to extend this technique on other music news sites besides the BBC but it requires that these sites link out to external sites, not just to their own pages. As discussed by Tim O'Reilly [2], there is a current trend for sites to prefer linking to themselves. Perhaps our approach to linking news stories with artists will encourage such sites to start linking out again.

3. SUMMARY

BBC Music Beta uses open source, community maintained MusicBrainz and Wikipedia projects to publish comprehensive information about the music heard on the BBC. We are exploiting techniques powered by the web links stored in MusicBrainz to aggregate content around artists, encouraging contributions by BBC staff to MusicBrainz and Wikipedia. The reuse of the existing MusicBrainz identifiers for our persistent URIs makes it easier for others to link to and reuse the data we are publishing.

4. ACKNOWLEDGMENTS

Our thanks to everyone at Audio and Music Interactive, in particular the Music Discovery and Music Interactive teams.

5. REFERENCES

[1] S. Auer and J. Lehmann, “What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content”, The Semantic Web: Research and Applications, pages 503-517, 2007, http://www.eswc2007.org/pdf/eswc07-auer.pdf

[2] Tim O’Reilly, “Is Linking to Yourself the Future of the Web?”,http://radar.oreilly.com/2008/08/is-linking-to-yourself-the-future-of-the-web.html

[3] BBC Music Beta: http://www.bbc.co.uk/music/beta

[4] MusicBrainz: http://www.musicbrainz.org

[5] MusicBrainz replication mechanism: http://musicbrainz.org/doc/ReplicationMechanics

[6] Music Ontology: http://musicontology.com

[7] Wikmedia Recent Changes IRC channel: http://meta.wikimedia.org/wiki/IRC_channels

[8] Wikipedia: http://www.wikipedia.org

[9] W3C, “Architecture of the World Wide Web, Volume One”,http://www.w3.org/TR/webarch

A web-based rights management system for developing trusted value networks

Víctor Torres
Universitat Pompeu Fabra
Carrer Roc Boronat 138
08018 Barcelona (Spain)
+34 935422586
victor.torres@upf.edu

Jaime Delgado, Xavier Maroñas, Silvia Llorente
Universitat Politècnica de Catalunya
Campus Nord. Carrer Jordi Girona, 1-3
08034 Barcelona (Spain)
+34 934011644, +34 934016783
jaime.delgado@ac.upc.edu, xmaronas@ac.upc.edu, silviall@ac.upc.edu

Marc Gauvin
NetPortedItems S.L.
Calle Sol Naciente 10
03016 Alicante (Spain)
+34 96 515 0664
mgauvin@digitalmediavalues.com

ABSTRACT

We present an innovative architecture that enables the digital representation of original works and derivatives while implementing Digital Rights Management (DRM) features. The architecture's main focus is on promoting trust within the multimedia content value networks rather than solely on content access and protection control. The system combines different features common in DRM systems such as licensing, content protection, authorization and reporting together with innovative concepts, such as the linkage of original and derived content and the definition of potential rights. The transmission of reporting requests across the content value network combined with the possibility for authors to preserve rights over derivative works enables the system to distribute income amongst all the actors involved in different steps of the creation and distribution chain. The implementation consists of a web application which interacts with different external services plus a desktop user application used to render protected content. It is currently publicly accessible for evaluation.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software - *Distributed systems.* H.3.5 [**Information Storage and Retrieval**]: Online Information Services - *Web-based services.* K.4.1 [**Computers and Society**]: Public Policy Issues - *Intellectual property rights, Privacy.* K.4.4 [**Computers and Society**]: Electronic Commerce - *Intellectual property, Security.*

General Terms

Algorithms, Management, Design, Security, Standardization, Legal Aspects.

Keywords

Content management, multimedia content protection, digital rights management, event reporting, user-generated content.

1. INTRODUCTION

The aim of this paper is to present a web-based system for the registration of original and derived digital content providing traditional Digital Rights Management (DRM) functionality as well as some innovative features. The proposed system, called IPOS-DS (Intellectual Property Operations System - Digital Shadow) [1][2], is about managing user creations and the information they own and provides users the necessary technology for being able to easily spread user-generated content in a trusted and protected manner.

For this purpose, IPOS-DS has proposed and implemented in a single web-based platform a set of innovative features, which are not present in existing DRM systems, and which relies on the relationships and entities that are being standardized in the MPEG-21 Multimedia Value Chain Ontology (MVCO) [3].

WWW 2009, April 20-24, 2009, Madrid, Spain.

2. IPOS-DS KEY CONCEPTS

IPOS-DS implements a set of relevant features and concepts, which are summarized next:

Content value network. This concept refers to the different steps of content creation, determining how the digital content evolves from representation of the creator's original work to become digital content representing other types of intellectual property entities that can be consumed by an end user.

Content Format, Lineage and Ownership. The content representation format adopted to represent objects is the Digital Media Project (DMP) Content Information (DCI) [4]. The fact that object representations are digitally signed provides a means to prove content ownership. On the other hand, the presence of a link from any derived object towards its ancestor enables the possibility to trace the whole content lineage, ensuring attribution.

Potential Rights. Potential rights correspond to those the object's author would be willing to grant to any user, including derived objects, along the content value chain.

Content Usage Monitoring. IPOS-DS follows the MPEG-21 Event Reporting [5] approach to embed event report requests in the registered objects, but enhancing it by transmitting the request along the whole content value network. In this way, all the involved actors are informed about its usage.

Benefit from the success of derivatives. IPOS-DS gives authors the possibility to benefit from the success of derived content by adding a new condition to determine the percentage of rights and incomes that the object's author preserves over any object derived from theirs.

Directed Rights. IPOS-DS enables the possibility to offer some rights for being acquired to some restricted set of users. This

feature is needed when the author wants only some selected users to be able to acquire the rights they are offering.

Really Simple Syndication. IPOS-DS provides a dynamic RSS 2.0 feed which contains information about the latest registered objects in the system so that they can be traced by any feed reader or aggregator.

3. ARCHITECTURE

IPOS-DS is a service-oriented architecture that consists of a main web application, accessible through a web browser, which interacts with different DRM components that are implemented as web services (see Figure 1). It also includes a desktop user application which renders protected content (see Figure 4). Next we summarize the main features it provides.

3.1 Applications

3.1.1 Web Application

From the web application the user can access almost all the system functionality. Figure 3 provides a simplified view of how the web application is divided into sections.

Registration of new Objects. In this section the user can register any kind of object. The web application provides a form (see Figure 3) where the user can fill the metadata fields, define the potential rights and attach a resource if needed.

Potential Rights Modification. This option is available for own objects and enables the user to modify rights that are offered for being acquired by the general public or specific users.

Search amongst own Objects. In this section the user can search by any of the metadata fields of the objects they registered.

Global Object retrieval and download. The user can perform a global search amongst all the objects registered in the system by any user. For the listed results (see Figure 2), several options are available, as e.g. view or download the object's XML, navigate towards the object's ancestor, if available, and acquire a license.

License acquisition. This option is accessible from the results obtained in the global object retrieval. When a user selects this option, they are redirected to a web page were they can select the rights and conditions amongst the different options the original author made available. Once the user selects the right and conditions they are interested in, a specific license is generated for that user, following the MPEG-21 Rights Expression Language (REL) [6].

View acquired objects. Once a user has acquired some licenses that enable them to exercise a right over an object, as reported in license acquisition, they can consult all of them in a specific section of the web application. This section also enables the user to register derived objects from those for which he owns a license that grants them the corresponding derivation right.

Search and view Reports. The user can search amongst all the reports that are directed to them.

Personal data management. The user is able to modify their personal data and default language for the web application.

User Groups and Contacts. The user can define their own contacts and contacts groups used for issuing directed rights.

3.1.2 Desktop User Application

The IPOS-DS desktop user application (see Figure 4) should be seen as a simple player devised to demonstrate how the resources can be rendered while ensuring the enforcement of the corresponding rights and conditions. The main functionalities in the player are:

Load Object. The player opens the object and displays the metadata to the user.

Download Content. Download the encrypted resource associated to the object. The player enables authenticated and authorized users to download the resource associated to the loaded object.

Decrypt and render the resource. It is done only if the user is authorized to. If authorized, the player will get the encryption key that can be used to decrypt the resource and render it.

3.2 Services

The IPOS-DS main web application interacts with different web services that implement the intelligence of the system, which are summarized next:

User Authentication and Registration Service. This component acts as a single-sign-on access point.

Content Registration Service. This component is responsible for registering the objects received after processing the registration form data in the main web application.

Content Service. It is an application that depends directly on the IPOS-DS web application used to transfer source files.

Objects Search Service. It provides a means for retrieving objects when searching amongst the objects' metadata.

License Service. It deals with the generation and archival of licenses, which convey user's usage rights and conditions.

Authorization Service. It enforces the fulfillment of the rights and conditions expressed in licenses.

Reporting Service. It collects the reports about content usage, while providing searching capabilities amongst the collected reports. Moreover, it determines the payment duties.

4. FUTURE WORK

IPOS-DS is currently accessible from [1] for evaluation. The goal for the next months is to promote its usage amongst different user communities that may be interested in using such a system for spreading their works and creations.

One of the potential groups where the system may be of interest could be the composer's collective, where different users with a trusted relationship use to collaborate to create, arrange and instantiate audio or audiovisual content.

Another goal is the adoption of the IPOS-DS platform by collecting societies in different countries. It could help to spread and ease the management of content generated by the millions of creators, adaptors and instantiators around the world generating content without any collective management or even digital object governance for that matter.

5. ACKNOWLEDGEMENTS

This work has been partly supported by the Spanish administration (Multimedia Content Management Life Cycle project, TEC2008-06692-C02-01).

6. REFERENCES

[1] NetPortedItems. IPOS-DS. http://www.digitalmediavalues.com/

[2] Distributed Multimedia Applications Group. http://dmag.ac.upc.edu

[3] M. Gauvin, J. Delgado et al. Media Value Chain Ontology (Committee Draft). ISO/IEC JTC 1/SC 29/WG 11/N10264

[4] The Digital Media Project. 2007. Approved Document No 2 – Technical Reference: Architecture, Version 3.0. No.1002/GA15.

[5] ISO/IEC 21000-15, Information technology – Multimedia framework (MPEG-21) – Part 15: Event Reporting.

[6] ISO/IEC 21000-5, Information technology – Multimedia framework (MPEG-21) – Part 5: Rights Expression Language.

7. SCREENSHOTS

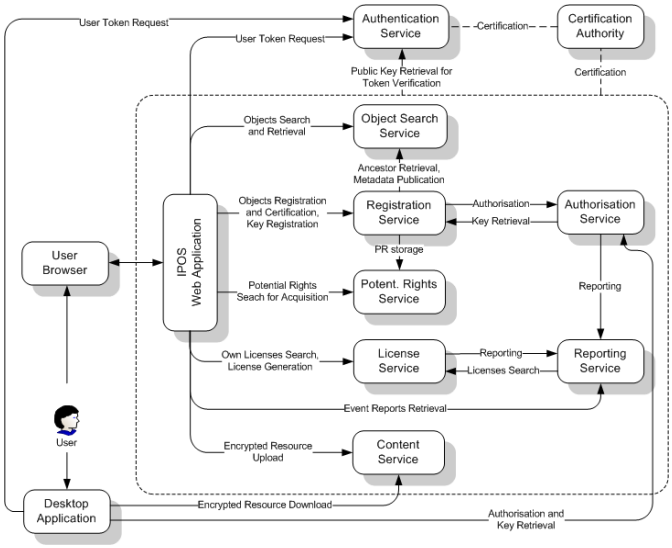


Figure 1. IPOS-DS overall architecture.

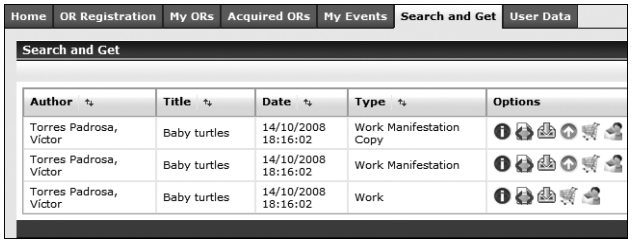


Figure 2. Result of a search by author.

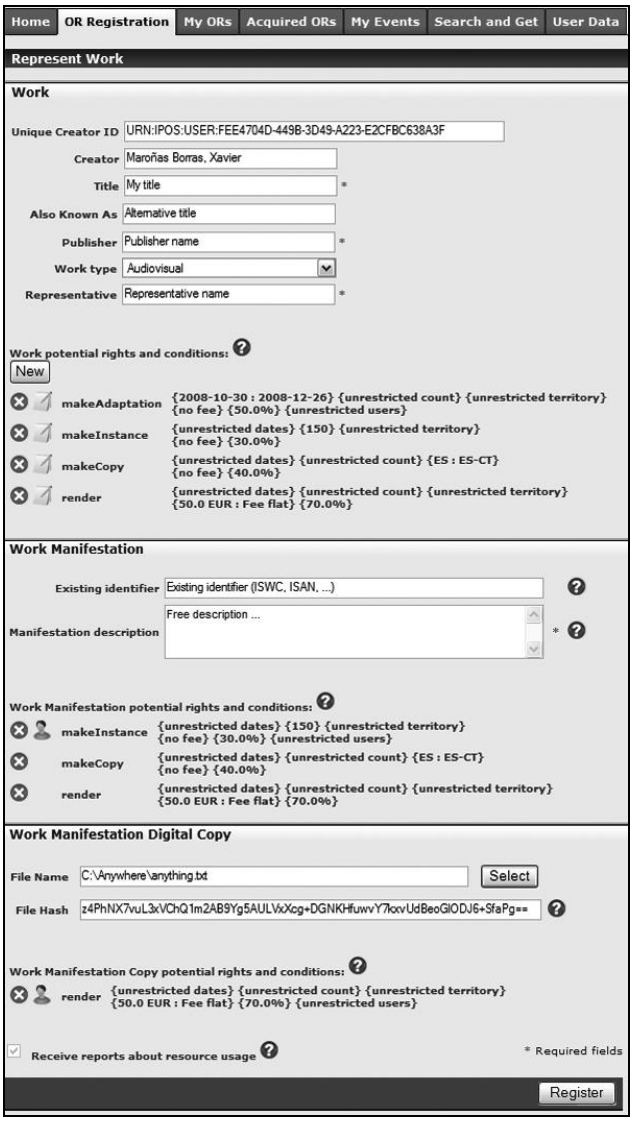


Figure 3. IPOS-DS web application view excerpt.

Figure 4. Desktop IPOS-DS user application.

Combining multi-level audio descriptors via web identification and aggregation

Jun Wang
National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences
wangjun@dsp.ac.cn

Xavier Amatriain
Telefonica Research
xavier@amatriain.net

David Garcia Garzon
Barcelona Media Centre d'Innovació, Universitat Pompeu Fabra
dgarcia@iua.upf.edu

Jinlin Wang
National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences
wangjl@dsp.ac.cn

ABSTRACT
In this paper, we present the *CLAM Aggregator* tool, which offers a convenient way to combine multi-level audio descriptors. A reliable method to identify users' local music collection with the open data resources is included in the tool. In the context of the *CLAM* framework and the *Annotator* application, the *Aggregator* allows users to configure, aggregate and edit music information ranging from low-level frame scales, to segment scales, or to any metadata from the outside world such as semantic web. All these steps are designed in a flexible, graphical, and user-defined way.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous

General Terms
Music Descriptors

Keywords
Annotating Tool, Linked Data, Music Descriptors, Semantic Web, XML

1. INTRODUCTION
Music feature extraction is one of the most crucial procedures in fields of Music Information Retrieval (MIR). The traditional solution is to automatically extract relevant features from the original music signal. Technologies such as Web 2.0 are making the World Wide Web a primary host of a sizeable amount of text-based and semantic information. This is allowing researchers to start to bring up novel ways to use this data for information retrieval tasks. For instance, research teams of Levy [1], Knees [2] and Barrington [3] etc. investigated metadata from the Web, including social tags, pages via Google queries and web-mined documents as an additional source of semantic metadata for MIR. All these works report that the combination of signal-level and web-based features outperforms each of them in isolation.

Nevertheless, web-based information is hardly ever associated with the audio files it refers to. The *CLAM Aggregator* aims at providing a convenient tool for combining multi-level music information. It interlinks all levels of descriptors ranging from the audio-based frame level features, to the web-based song level descriptors. It enables researchers to deal with multivariate data analysis and develop complicated methods such as leveraging the semantic gap.

The most remarkable features of the *CLAM Aggregator* are: (i) Being Developed in the context of the *CLAM* framework [4] and *CLAM Annotator* [5], which allows to extend it in unlimited ways, by embedding user-defined automatic extraction algorithms in it; (ii) Offering reliable identifications to link the audio files with the Linking Open Data [6], avoiding the obstacles of noisy, redundant or false extraction in the existing technologies; (iii) Using standard XML language to represent data. Thus, it works with readable and understandable data files that are easily connected to external applications or databases and can also be easily converted to semantically rich RDF resources and linked back to the Linking Open Data.

2. FEATURES AND COMPONENTS
2.1 Annotator Schemas and Descriptors
The *CLAM Annotator* (see [5]) offers a convenient GUI that allows editing multi-level descriptors. It makes use of different XML files: the *Annotator Schema* files define the list, the type and the value range of different descriptors; the *Descriptors pool* files contain all the actual values for the descriptors.

We can define multi-level descriptors in the Schema. A song-level descriptor is unique within the song scope. It can be of any type. On the other hand, a low-level descriptor has a segment or frame scope and must be of floating point type.

Descriptors may be generated by any third-party extractors by providing a proper Schema. Any extraction algorithm may dump its results in the XML representation format of a *CLAM Descriptors Pool*, without having to worry about formatting issues.

2.2 Aggregation
More and better algorithms for automatic feature extraction are constantly being developed in the MIR field. For such purposes, and especially for researchers to conveniently develop, compare, and analyze these new algorithms, it is important to allow the combination of descriptors in a single Annotator project. The *CLAM Aggregator* offers a dynamic GUI to achieve this. This can be considered as another extractor which also makes use of the fore-mentioned *Schema* files and *Descriptors pool* files.

The *CLAM Aggregator* extracts selected descriptors from the output pool files of different extractors, according to the user configuration in the GUI. An example of the generated configuration file is shown in listing 1. The list of *sources* defines the IDs of different extractors (e.g., "ClamCore", "Chord",

"SemWeb"), configures the *Schema* files, the *Descriptors pool* file suffix, and the extractors respectively. The list of *map* defines the selected attributes of each extractor, and maps the scope::attribute names from the sources to the target aggregated pool files. According to these configurations, the Aggregator's *Schema* file and *Descriptors pool* file are automatically generated.

```
sources = [     ("ClamCore", FileMetadataSource(path=".", schemaFile="CLAM.sc",
                                poolSuffix=".example", extractor="ClamExtractor")),
                ("Chord", FileMetadataSource(path=".", schemaFile="Chords.sc",
                                poolSuffix=".chord", extractor="ChordExtractor")),
                ("SemWeb", FileMetadataSource(path=".", schemaFile="SemWeb.sc",
                                poolSuffix=".webpool", extractor="SemWebExtractor")),
]
map = [         # ('TargetScope::TargetAttribute', 'ID', 'SourceScope::SourceAttribute'),
                ("Song::Danceable", " ClamCore ", "Song::Danceability"),
                ("Song::ChordFrames", "Chord", "Song::Frames"),
                ("ChordFrame::ChordHartePcp", "Chord", "Frame::HartePcp"),
                ("Song::TagJamendo", "SemWeb", "Song::TagJamendo"),
                ("Song::Review", " SemWeb ", "Song::Review"),
                ("Song::Rating", " SemWeb ", "Song::AlbumRating"),
]
```

Listing 1. Configuration File generated from the CLAM Aggregator GUI

2.3 Web Identification

Above, we described the convenience for research teams to embed and integrate their extractors in *CLAM Annotator*, by easily adapting the applications to certain interface. It should be noticed that web-based descriptors are becoming more and more important in MIR fields nowadays. However, the existing extraction solutions like Google query and other search-engine-based methods cannot overcome the issues related to noisy, redundant or false extractions.

Here, we demonstrate a reliable and extendable web-based extractor. The extractor first makes use of GNAT (Yves, et al., [7]) to find corresponding related identifiers of local audio files. GNAT uses audio fingerprinting and available metadata to identify the songs MBID on MusicBrainz, and then outputs RDF statements representing the links between local songs and the remote web identifiers. The proposed graph matching algorithm allows GNAT to be robust even with inaccurate or incomplete local metadata.

Using the MBID and the RDF statements, the extractor crawls through several datasets of Linking Open Data and extracts high-level descriptors such as editorial metadata, user comments, genre, album reviews, or tags. This approach is related to the interlinking of music-related datasets on the Semantic Web [8]. The extraction could be extended in unlimited and flexible ways, to respond to the need of different research tasks and applications.

3. APPLICATION GUI

3.1 Configuration

Extractor names, corresponding scopes and attributes can be shown graphically (see fig. 1). Users can select desired attributes by checking them. A configuration file is then generated. Furthermore, and according to the configuration, an aggregated description Schema is automatically constructed (fig. 2).

3.2 Editing High-level Descriptors

Fig. 3 depicts the GUI for browsing and editing high-level descriptors. Users can enter free text, e.g. human-knowledge descriptors like "extreme sports" to describe the *Usage* attribute of a song.

3.3 Fine Tuning Low-level Descriptors

Part of the powerful GUI of *CLAM Annotator* for fine tuning low-level descriptors is shown in fig. 4. Other applications for tasks such as editing segmentation marks or auralizing annotations are also provided in this framework (see details in [5])

4. CONCLUSIONS AND FUTURE WORK

In the paper we have presented the *CLAM Aggregator*, which can be used to link local music collections with Linking Open Data, and combine multi-level descriptors in a user-configurable way. This tool is available as open source at http://clam-project.org/clam/trunk. The demo and screenshots are on Planet Clam at http://clam-project.org/planet/index.html. More screenshots can be viewed on Wikis at http://clam-project.org/wiki/Development_screenshots.

Future plans include research into the advantages such multi-level combination brings for automatic high-level semantic annotations (e.g. emotions) of music. Secondly, a conversion of the XML-based local audio descriptor files to Semantic Web resources, and linking them to corresponding identifiers on Linking Open Data.

5. ACKNOWLEDGMENTS

Our thanks to CLAM development team for their contributions and the research team of Yves Raimond, etc., for allowing us to adopt tools they had developed. Part of this work has been supported by the Google Summer of Code 2008 project.

6. REFERENCES

[1] M. Levy, M. Sandler. A semantic space for music derived from social tags, In Proceedings of ISMIR 2007.

[2] P. Knees, T. Pohle, M. Schedl and G. Widmer. A Music Search Engine Built upon Audio-based and Web-based Similarity Measures. In Proceedings of SIGIR 2007.

[3] L. Barrington, M. Yazdani, D. Turnbull, G. Lanckriet. Combining feature kernels for semantic music retrieval, In Proceedings of ISMIR 2008.

[4] X. Amatriain. CLAM: A framework for audio and music application development. IEEE Software, vol. 24, no. 1, pp. 82-85, Jan./Feb. 2007, doi:10.1109/MS.2007.8.

[5] X. Amatriain, J. Massaguer, D. Garcia, I. Mosquera. The CLAM Annotator: a cross-platform audio descriptors editing tool. In Proceedings of ISMIR 2005.

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

[7] Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the Semantic Web. In Proceedings of IDOW'08 (Beijing, China, April 2008).

[8] C. Bizer, T. Heath, D. Ayers, Y. Raimond. Interlinking open data on the web. In Demonstrations Track, 4th European Semantic Web Conference, Innsbruck, Austria, 2007.

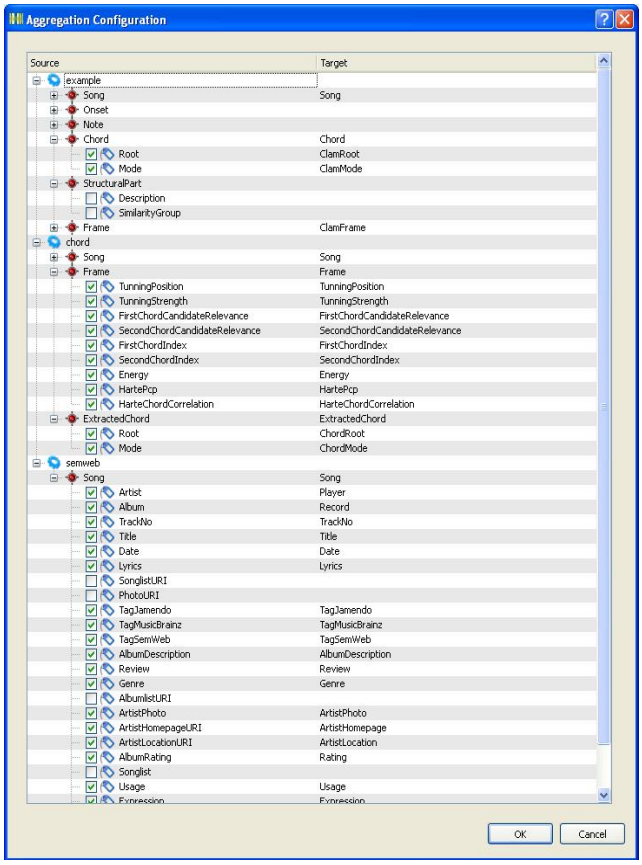


Figure 1. Configuration GUI for aggregation.

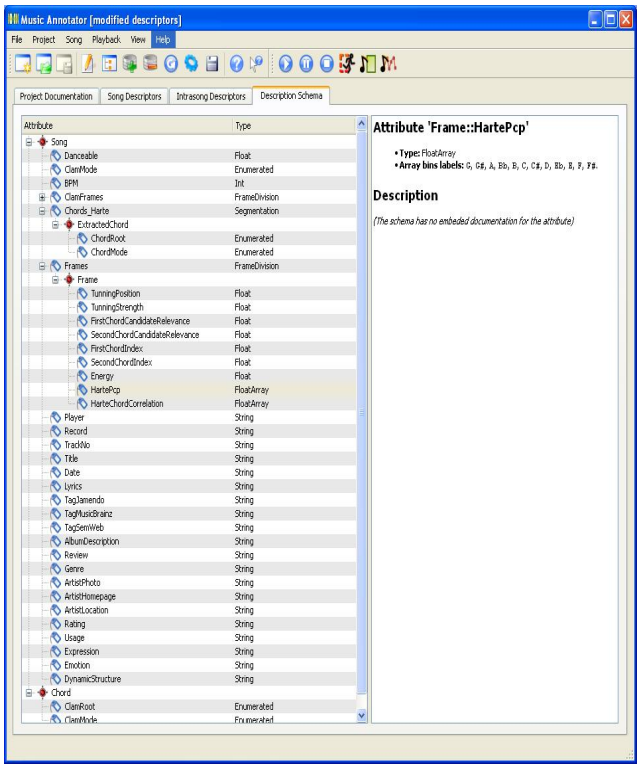


Figure 2. Aggregated description Schema.

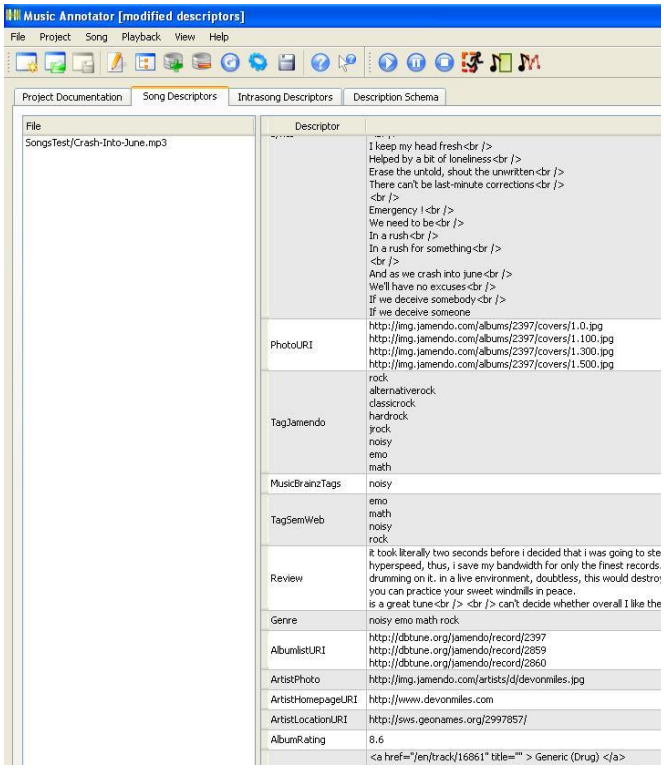


Figure 3. Editing high-level descriptors.

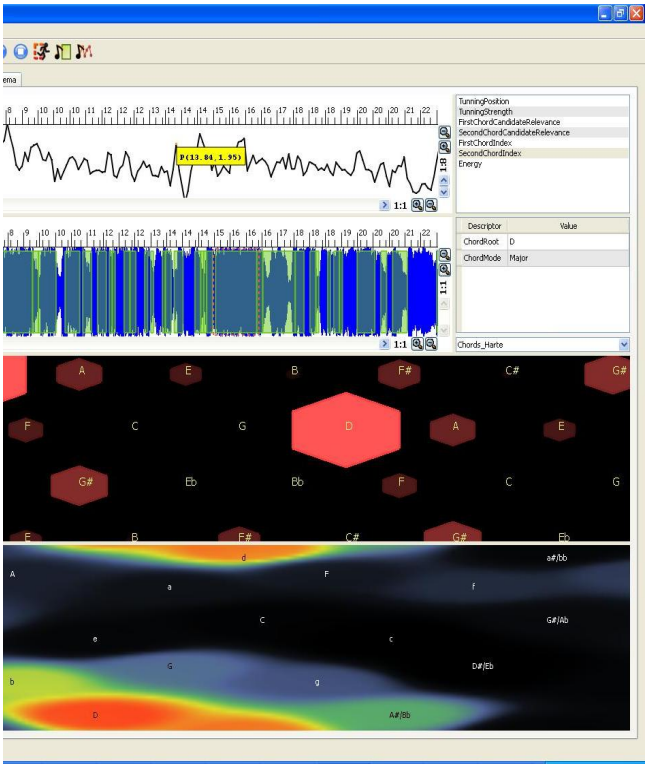


Figure 4. Fine tuning low-level descriptors.

WebNC: efficient sharing of web applications

Laurent Denoue, Scott Carter, John Adcock, Gene Golovchinsky, Andreas Girgensohn
FXPAL
3400 Hillview Ave., building 4
Palo Alto, CA 94304, USA
1(650)842-4821
{denoue,carter,adcock,gene,andreas}@fxpal.com

ABSTRACT
WebNC is a browser plugin that leverages the Document Object Model for efficiently sharing web browser windows or recording web browsing sessions to be replayed later. Unlike existing screen-sharing or screencasting tools, WebNC is specially optimized to handle scrolling within a web page. Rendered pages are captured as image tiles, and transmitted to a central server through http post. Viewers can watch the webcasts in real-time or asynchronously using a standard web browser: WebNC only relies on html and javascript to reproduce the captured web content. Along with the visual content of web pages, WebNC also captures their layout and textual content for later retrieval. The resulting webcasts require very little bandwidth, are viewable on any modern web browser including the iPhone and Android phones, and are searchable by keyword.

Categories and Subject Descriptors
H.5.3 [INFORMATION INTERFACES AND PRESENTATION]: Group and Organization Interfaces, Web-based interaction

Keywords
Webcasting, screencasting, browser plugin, real-time sharing

1. INTRODUCTION
Screencasting and screensharing are useful for real-time or asynchronous collaboration [1]. They were designed to capture desktop windows [2,4], not specifically web pages. But today a lot of user interaction happens inside a web browser window, so we designed WebNC (Figure 1) specifically to capture web applications.

Living inside a browser as a plugin, WebNC has access to the Document Object Model of the pages, including the type of its elements such as text, pictures, video clips, flash movies, applets, and the location of the scrollbars in these elements. By leveraging this data, WebNC can efficiently capture a rendered web page as image tiles along with metadata like viewport size, cursor position and scroll positions, and send them to a central server.

Once captured, these webcasts can then be viewed using a standard web browser without requiring any plugin. WebNC outputs its webcasts as standard HTML and Javascript, making them viewable with most modern web browsers, including smartphones such as the Apple iPhone and Google Android.

WWW 2009, April 20-24, 2009, Madrid, Spain.

Because it uses only outbound HTTP requests for both the plugin and the viewing javascript code, WebNC avoids the need to reconfigure the network firewall. Unlike most video-codec based screen-recorders, WebNC captures webcasts at their native resolution, thus producing high-resolution and readable webcasts.

Unlike existing tools, WebNC also captures textual elements being rendered by the web browser and their location on screen, allowing users to retrieve webcasts by content.

Finally, WebNC solves a real privacy problem that emerges with existing tools: with WebNC, users can share just a single TAB in their browser, and any popup not part of that TAB will not be shared, like this private instant message or email popping up during a web-conference.

Figure 1. On the bottom right, the WebNC icon on reads "WebNC on" and is blue when the user shares a web browser window; otherwise, it reads "WebNC off" and turns black. Session time and bandwidth is shown left of the icon.

2. WebNC SYSTEM DESCRIPTION
2.1 User experience
To start sharing a web page, the user simply clicks on the WebNC icon on the bottom/right corner of the browser window (after having installed the WebNC extension which works on all platforms, including Windows, Mac and Linux). The icon changes color to indicate that it is actively recording. The user is shown a sessionID that can be given to others for them to view the live webcast, or kept for later viewing.

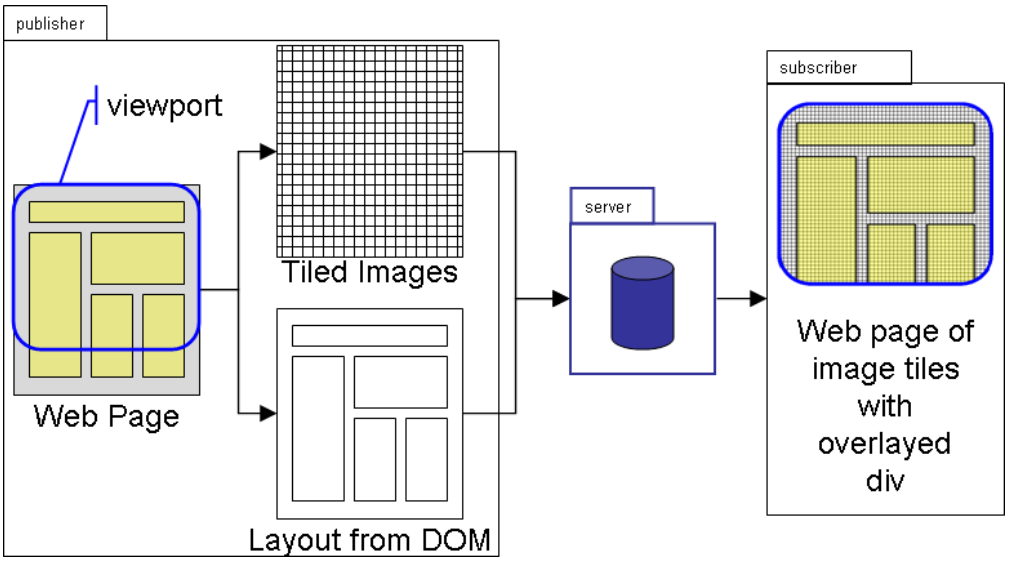


Figure 2: WebNC system overview. The extension (publisher) captures the visual content of the web page as a set of tiled images along with its context (viewport, layout, etc..). The server then acts as a proxy, sending image tiles, layout information, and scroll events to the viewer (subscriber).

By default, WebNC only captures the browser tab that was active when the user initiated the session. If the user switches to another tab in the browser, this new tab will not be captured. The user needs not worry about moving the window outside of the screen boundary as WebNC will still be able to capture these regions (see next section). When the user is done sharing, a simple click on the WebNC icon ends the session.

2.2 Capturing web pages

Living inside the browser, the WebNC plugin has access to the Document Object Model (DOM). It determines the vertical and horizontal positions of the scrollbars and uses them to name the tiles of 256x256 pixels that it captures, starting from the top/left corner of the viewable area on the web page. Tiles are captured using the drawWindow function of the canvas element in Firefox [3]. An equivalent call is available in Internet Explorer. Each tile is named after its position on the page, starting from 0,0 for the top/left tile and moving right and down until it reaches the entire scrollable area, not only the viewable area. WebNC has also access to the hidden pixels, so it can grab entire 256x256 tiles even when some of the pixel rows and columns fall outside of the viewable area.

Every once in a while, WebNC grabs the entire viewable area as a reference bitmap using drawWindow API from the canvas object [3]. It then compresses into PNG or JPG each tile until all tiles in the viewable area are compressed. Images are hashed into a unique signature comprising the MD5 of the URL, bits of the image itself, and the tile number. Tiles that have not been sent already are sent, along with the cursor position, its shape, and the size of the window, scroll position and scrollable area. This data is also sent even when no tile has changed, giving the server the latest cursor position and scrolling information.

2.3 Capturing text

WebNC also queries the nsIAccessibleText object of text elements to extract characters and their bounding boxes. This content is optionally sent along with the tile images and stored on the server. Users can retrieve specific parts of webcasts containing a given keyword. Privacy settings can filter out common entities such as email addresses, social security numbers, phone numbers, and street addresses.

2.4 Viewing webcasts: html and javascript

A standard web browser pointed to the WebNC server is enough to view a session. The page embeds javascript code that uses Ajax polling to retrieve the metadata of tile names and viewport size, scroll position, and cursor data.

The javascript creates a DIV element set to the viewport width and height, and embeds another DIV set to the size of the scrollable area inside the first DIV. It then creates as many IMG elements are required, setting their SCR to the tile values received earlier. Finally, the position of the embedded DIV is set to the scroll positions, giving users the effect of scrolling the viewport to the corresponding position. The mouse position is used to place another IMG absolutely over the first DIV element, with its SRC set to replicate the shape of the cursor.

Because the viewer code receives the data periodically from the server, WebNC uses javascript timers to smoothly transition the locations of the fake cursor as well as the scroll positions. This technique makes for a very smooth and enjoyable viewing experience.

3. EVALUATION and DISCUSSION

Our current system is built as an extension for Firefox, and thus runs on Windows, Mac and Linux. For Windows, we also built a native C++ plug-in to capture objects such as Flash/QuickTime/Applets for which drawing surfaces were not available to the drawWindow function on Window. We built the WebNC server using Java/Tomcat with a simple JSP pages. Data sent from the plugins are stored in memory, keyed under a given sessionid. The viewer code is in pure html and javascript and thus works on all web browsers tested so far, including Firefox,

Internet Explorer, Safari, as well as mobile web browsers based on WebKit as found in iPhone and Android G1 smartphones.

A few users tried WebNC and could easily share their browser window. Viewers were equally able to view webcasts, and were quite impressed with the smooth cursor and scrolling behaviors.

Using WinMacro [5], we recorded a 2.5 minute web browsing session involving navigation among several Wikipedia articles and the Fuji Xerox homepage. In all, 9 unique web pages were visited in the course of the session. The interactions included a fair amount of scrolling. We replayed the session in a full-screen web browser window on a 1024x768 desktop, viewed it remotely with various methods, and measured the network use by capturing the network traffic with WireShark [6]. WebNC used an average of 280 kbps, versus 130 kbps for Microsoft Remote Desktop Protocol (RDP), 521 kbps for UltraVNC with tight encoding and caching enabled and 729 kbps for Microsoft SharedView. We ran a follow-up test of RDP where we disabled the persistent bitmap cache and rebooted the client to insure that the in-memory tile cache was empty. In this second test the average bandwidth used by RDP was measured at 225 kbps.

WebNC is not yet as bandwidth efficient as RDP, though leveling the playing field by starting from an empty image cache as described above narrows the gap significantly. The current implementation of WebNC has several opportunities for optimization. First, sending back only tiles that have changed to the client could save up to 150 kbps. Second, network overhead can further be reduced by changing our polling Ajax to a server-push comet technique (*e.g.*: long polling). It's worth restating that unlike RDP and the others, WebNC does not require software beyond a standard web browser to view shared content.

4. CONCLUSION AND FUTURE WORK

WebNC works and has been shown to produce low bandwidth webcasts that are compliant with web standards for viewing. The ability to retrieve webcasts by content is extremely interesting, and more work is required. It would be possible to allow users to retrieve webcasts based not only on keywords captured during the webcast, but also on element types such as whether the page contained forms, video clips, images, applets, etc. WebNC needs to be optimized to cut out dynamic elements like movie clips and treat these as separate tiles, or sprites.

5. REFERENCES

[1] IA. W. Esenther. Instant co-browsing: Lightweight real-time collaborative Web browsing. In In Proc. of the 11th Int. WWW Conference, May , pages 107--114, Honolulu, Hawaii, 2002.

[2] Microsoft Remote Desktop Protocol *http://en.wikipedia.org/wiki/Remote_Desktop_Protocol*

[3] Mozilla Firefox canvas API *http://developer.mozilla.org/En/Drawing_Graphics_with_Canvas*

[4] Virtual Network Computing *http:// www.realvnc.com*

[5] http://geocities.com/win_macro/winmacro_v1_2.html

[6] http://www.wireshark.org

Author Index