Dynamic Visualizations for Multi-Domain Search Results

Alessandro Bozzon, Luca Cioria, Piero Fraternali, Maristella Matera
Dipartimento di Elettronica e Informazione, Politecnico di Milano
P.zza L. da Vinci, 32 - 20133 - Milano
[name.surname]@polimi.it

ABSTRACT

Search systems are becoming increasingly sophisticated in their capacity of building results that are not mere lists of documents but articulated sets of concepts retrieved from different domains. As the search result sets exhibit more structure, techniques are required to visualize the retrieved objects in a way that facilitates the immediate understanding of their properties and relationships. This paper investigates the use of models to represent both result sets and visualization spaces, and of model-to-model transformations to dynamically suggest an optimized result visualization for multi-domain search.

1. INTRODUCTION

Past years have seen an evolution in the way search engines, and more generally information seeking applications, deliver responses to user's information needs. Mainstream search engines are now capable of recognizing quite a large amount of concepts in the input keywords (people, cities, events, etc.) and provide a customized representation of results, e.g., by including maps, photographs, videos, and concept-dependent data, like tourism information for a retrieved city. A more sophisticated approach to result visualization is however provided by vertical search applications (e.g., Wanderfly - www.wanderfly.com), which exploit domain knowledge to optimize the display of retrieved results.

Also due to the always increasing availability of public Web data sources in different domains, we expect a diffusion of high quality information retrieval systems and an increased interest for search-based applications also in B2B and B2C applications, as a primary means for locating relevant documents and/or combinations of objects of interest. These factors motivate the demand for appropriate development methods, supporting the construction of effective result presentation interfaces [9]. Like in horizontal search engines, one would like to achieve a flexible and dynamic assembly of the result layout: the kind and amount of information should vary depending on the concepts found in the result set. As in vertical search applications, one would like to fine tune the result display to the type of objects retrieved, to optimize the immediate readability of the result page. This may require varying the visualization technique quite radically, e.g., using maps to chart multiple geo-referenced objects, time lines to convey temporal series, and ad hoc widgets for multidimensional data.

This paper addresses the problem of automating the construction of result visualization interfaces for *multi-domain search* tasks, where results are ranked combinations of objects with typed attributes and relationships. In [3] we already clarified our perspective on visualizations for multi-domain search tasks, and discussed how the dynamic construction of search results visualizations can be "reduced" to the identification of a model-to-model mapping between a *data set model* and a *visualization space model*. In this paper we illustrate in details the models and the mapping process that exploits static and dynamic result properties (e.g., data types and attribute value distribution) to dynamically determine the visualization to use for result presentation.

The paper is organized as follows: Section 2 overviews the related work; Section 3 introduces the models of the *result data set* space and the *visualization* space, and describes the mapping rules between such models for choosing the most appropriate visualizations at runtime; Section 4 illustrates the mapping on a running example; Section 5 briefly shows how the described mapping is implemented within an architecture for multi-domain search applications; finally, Section 6 outlines the future work.

2. RELATED WORK

New generation search engines are moving towards the collection and integration of heterogeneous data sources. Kosmix [11] is an example of general-purpose topic discovery engine. It offers one-page information summaries about a topic, retrieved through calls to Web services that extract information from deep Web data sources. The schema of each topic is a complex record type, which not only comprises typed properties of the entity but also associations to other entities representing related topics.

Data visualization has a long standing tradition, which initially focused on the analysis of alternative visualization techniques for various categories of data [4]. Classic works like [10] [12] offered guidance in selecting the most appropriate visualization techniques for different types of data (e.g., 1-, 2-, 3-dimensional data, temporal and multi-dimensional data, and tree and network data). Later works (e.g., [6] [13]) explored the underlying conceptual structure of data-

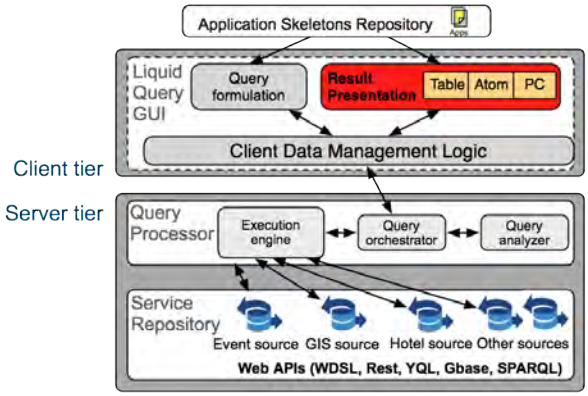

Figure 1: SeCo Architecture

oriented visualizations, highlighting a common framework of data visualization strategies, giving a deeper rationale to the taxonomies of visualization techniques.

Much work has also addressed the automatic generation of presentations. The pioneer approach proposed by Mackinlay [8], and other successive works (e.g., [7] [4]), exploit data characterization and propose rule-based approaches to map data types to visual elements. The common aim of such works is to automatically derive "adequate" visualizations [4], where adequate means *complete*, i.e., the user perceive from it all the information enclosed within the original data, and *correct*, i.e., no other information is perceived.

Our work capitalizes on the results achieved in the research fields above mentioned. As we will illustrate in the following sections, we apply such results to the dynamic visualization of multi-domain search results [3], trying to address the issues that characterize this specific context.

3. DYNAMIC VISUALIZATION PROCESS

The problem of multi-domain search is defined as the computation and presentation of results to queries over multiple Web data sources that return (possibly ranked) lists of objects. A typical multi-domain query, which we will use throughout the paper as running case, is: *Find combinations of hospitals and doctors specialized in the treatment of a given disease, ranked based on the rating of the hospital and on the scientific impact of the doctor.*

Answering multi-domain queries requires a processing architecture like the one implemented in the Search Computing (SeCo) Project [5], and illustrated in Figure 1. The server tier comprises a *Service Repository*, where external data sources can be wrapped and registered using a variety of technologies, and a *Query Processor*, where the orchestrator invokes the analyzer to decompose the query into service calls, and then sends an execution plan to the runtime engine that manages the invocation of services and the assembly of results efficiently. In the client tier, the *Liquid Query Graphical User Interface* (LQ GUI) [2] allows the user to formulate queries instantiating pre-registered search *application skeletons*, that declare the available data sources and the connection paths joining a source to another one.

The currently implemented LQ GUI has a fixed set of data visualizations (table, atom view, and maps).

The work described in this paper aims at equipping the result presentation module highlighted in Figure 1 with the capability of automatically suggesting visualizations to the

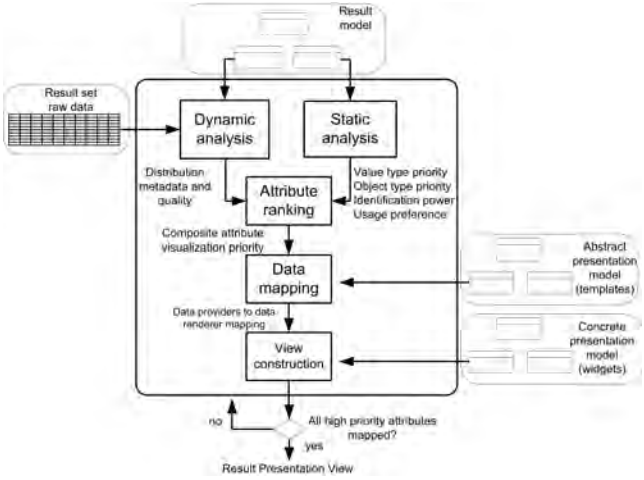

Figure 2: Overview of the visualization process

user, based on the features of the current result set and on the available visualization templates; both aspects are encoded as models. The idea is that when the user submits the initial query, the results are analyzed on the fly and the proper visualization is selected and adapted to the characteristics of the retrieved objects (e.g., since hospitals are geo-referenced, results are displayed on a map). If the user interacts with the query, then the system analyzes the updated result set and suggests alternative visualizations, (e.g., a time-line, if the user chooses to visualize the doctors' dates of availability).

3.1 Overview of the process

The dynamic visualization process is shown in Figure 2. It outputs the definition of the *presentation view* to use for displaying results, starting from a number of inputs specifying relevant characteristics of the result set and the visualization space. In particular, the *result set data* consists of a ranked list of *combinations*, i.e., tuples of *objects* extracted from different data sources and correlated by join conditions. The *result set model* expresses properties of object attributes that can be used for deciding the visualization and also incorporates usage preferences. The *visualization models* expresses the organization of the view, at the abstract and *concrete* level. The abstract level specifies the composition of the view in terms of canonical visualization forms, called *templates*. Examples of templates are cartesian planes, maps, timelines, vertical lists, and temporal animations (cartesian planes + time). The concrete level instantiates templates by identifying the widgets to be actually used to implement the template visualization paradigm.

The goal of the process is to determine the best mapping from *data providers* (attribute values, object instances and combinations of objects) to *data renderers* (axes and visual clues that make up the templates) so that the result set is visualized in a way that best matches the distribution of objects and combinations in the result set, the types of the object attributes, and the preferences about which information to show first.

The output view is decided in consecutive steps. *Dynamic Analysis* collects statistics on result set data that may impact visualization (e.g., range and density of attribute values). In parallel, *Static Analysis* extracts from the result set

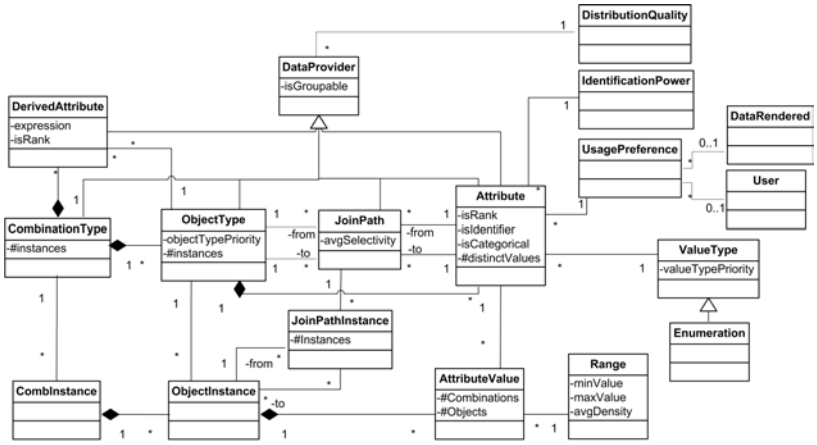

Figure 3: Meta-model of the result set

model visualization priorities of attributes according to the characteristics of their type, their suitability to identify objects, and relative importance of their information content.

As a second step, starting from the abstract visualization model, *Data Mapping* employs heuristics to calculate a matching between the sorted list of attributes and the available visualization *templates*. Each template receives a suitability score and the top-ranked template is selected.

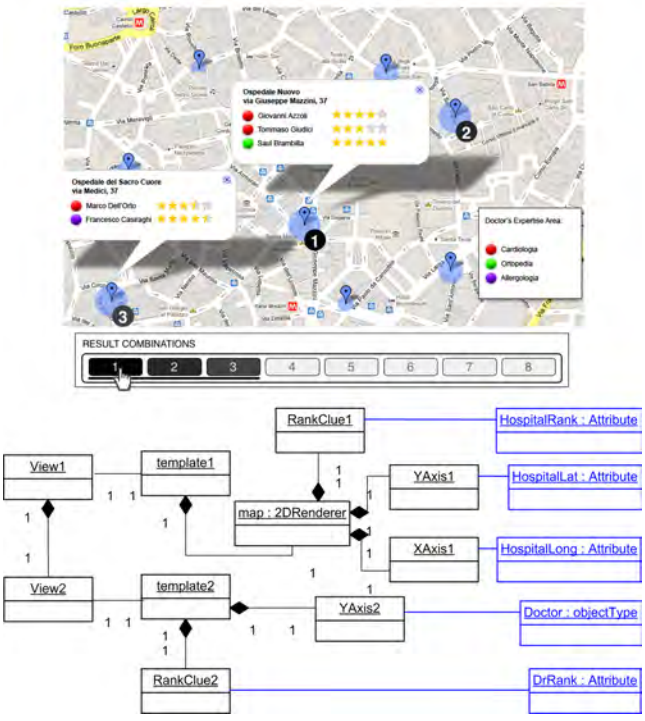

Figure 4: A rendered view (top) and its abstract model (bottom)

Finally, *View Construction* converts the chosen abstract template into a concrete view, by replacing abstract data renderers with concrete widgets (e.g., a map template is concretely implemented as a Google map view with overlaid HTML 5 elements). The process can be recursive: if attributes with priority above a threshold could not be assigned to a template, a sub-view can be created by invoking the same process. Typically, this is done on an object-by-object basis, to create sub-views that can be displayed on demand (e.g., pop-up windows with the details of a doctor not displayed on the map). When all important attributes are mapped, the (possibly nested) view is instantiated and added to the LQ GUI, to be directly rendered or suggested to the user.

3.2 Result set model

The result set model specifies the properties relevant for visualization of combinations, objects and attributes, which conforms to the meta-model described in Figure 3. It represents type-level, instance-level and statistic information.

At the type level, the containment structure of *combinations*, *objects* and *attributes* is represented, with the join paths that correlate object types and the attributes used for joining. *Rank criteria* of objects and combinations are modeled as *derived attributes* consisting of weighed sums of attribute values. Object rank can also be expressed simply by the value of an attribute (marked by the *isRank* Boolean flag). Attributes have a *type*, can be *identifiers* (e.g., primary or secondary key), and may denote categories (marked by the *isCategorical* flag). At the instance level, *combination instances* contain *object instances*, which are tuple of *attribute values*. Also, *join path instances* denote the sets of target object associated to a source object. At the statistic level, the results of dynamic analysis useful for visualization are represented: the implicit *enumeration* type of string attributes (e.g., the doctor's specialties appearing in the result set), the actual *range* of attribute values, the *number of instances* of objects and combinations, the *average selectivity* of attribute values and join paths; *density* statistics also express the number of attribute values per interval in the range. The instances of join paths can be used to support nested object visualization: objects lacking a strong "visual characterization" can be associated with stronger objects, resulting in a nested visualization. Figure 4 shows an example (on top): the placement of hospitals on the map is exploited to show where the associated doctors work, even if doctors are not geo-referenced *per se*.

Finally, the result set model also expresses different kinds of quality values heuristically assigned by the mapping process to the attributes.

The *Distribution Quality* is a numerical measure summarizing the effectiveness of displaying the instances of a data

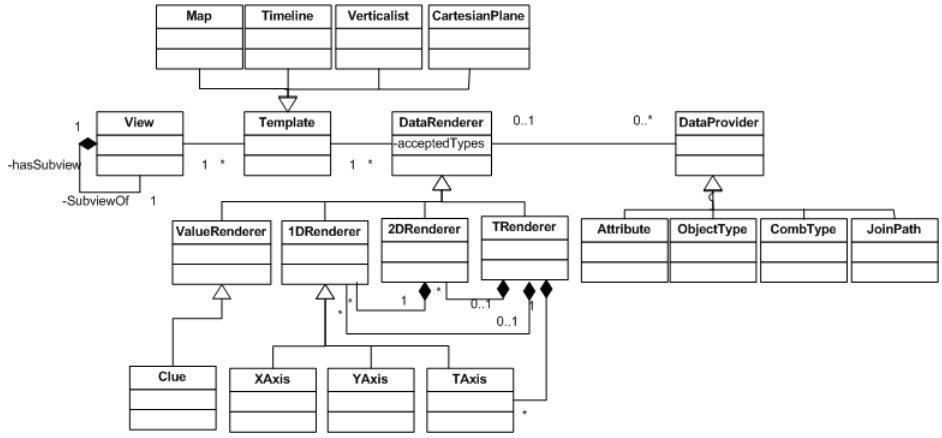

Figure 5: Meta-model of the visualization space

provider (combinations, objects instances, joined objects, and attribute values) based on their distribution statistics. For instance, an attribute with too dense value distribution has low quality. This indicator is computed in two variants: one (*unclustered*) for the case in which all distinct values are rendered; one (*clustered*) for the case in which values are grouped. The latter variant is computed when the density of unclustered values exceeds a threshold, based on a subdivision of the range of values into fixed width intervals.

The *Identification Power* represents the suitability of the attribute to denote meaningfully an object instance. For instance, object's external names have high identification power. The value is derived from the specification of primary or secondary key attributes in the definition of the queried data sources.

The *Object Priority and Attribute Type Priority* represent respectively (i) the relative importance of objects (e.g., hospital first, then doctors) and (ii) a partial order over attribute types boosting those types that have highly communicative power (like geographical coordinates, timestamps). Similar boost is given to rank attributes.

The *Usage Preference* specifies the user-perceived suitability of attributes to be associated to certain visualization dimensions. For instance, latitude and longitude attributes may have high usage preference values as data supplier to Cartesian axes, while attributes correlated to the ranking of objects can be effectively associated to visual clues, as shown in Figure 4, where the size of the circle around an hospital is proportional to its rank position.

3.3 Abstract visualization model

The meta-model of the abstract visualization is described in Figure 5. A *view* contains one or more *templates*, constituted by *abstract data containers*, which can be punctual, monodimensional, bidimentional, or timed. Examples of monodimensional data renderers are horizontal and vertical axes, and temporal axes. The latter are intended as a temporal animation where data values are presented in succession. Example of punctual data containers are visual clues (e.g., size and color). The mapping process associates each data renderer to a data provider, which can be anything containing values: a combination type, an object type, an attribute or a join path.

Figure 4 shows an example of rendered view (top) and of the corresponding abstract model (bottom) for our running query. The view consists of a map template and of a nested subview. The map templates has two axis, for geographical coordinates, and a visual clue dimension, for a numerical attribute. The subview consists of a list templates, with one vertical axis and a visual clue for a numerical attribute. The bottom part of Figure 4 shows the visualization model resulting from the mapping process, and highlights the mapping of data providers to dimensions: the latitude and longitude of hospital objects are associated with the axis of the map template, and the *Hospital* rank to the visual clue. The axis of the list template in the subview is mapped to the object instances of the *Doctor* object type, and the visual clue to the doctor's specialty and rank.

4. MAPPING SEARCH RESULTS TO VISUALIZATIONS

We now illustrate the mapping process on the running example. Table 1 shows a sample of combinations of the two objects *Hospital* and *Doctor* used in the exemplification.

Static analysis extracts a number of metadata annotated into the result data set model, whose values range from 0 to 1, which we report in the following.

Object Priority and Attribute Type Priority. We assume that our running query is meant to facilitate the location of hospitals; therefore, all the Hospital attributes have the highest object priority ($objectPriority = 1$). Also, geographical attributes have the highest attribute type priority ($attributeTypePriority = 1$), since we assume that maps are effective visualizations for objects with geo-referenced attributes.

Identification Power. Hospital.Name, *Hospital.Address* and *Doctor.Name* are assigned with the highest scores, given their suitability to denote objects in the visualization space.

Rank Correlation. Score, *Hospital.Rank* and *Doctor.Rank* have the highest *rankCorrelation*, being them ranking attributes.

Usage Preference. Usage preferences depend on the available templates, which we assume to be: maps, timelines, cartesian planes, and vertical lists, composed of axes and visual clues. *Hospital.Lat* and *Hospital.Long* have the highest preference related to the visual elements `XAsis` and `YAxis` respectively. Other secondary preferences for the `XAxis` element go to *Hospital.Name* and *Doctor.Name*, and for the `YAxis` to *Hospital.Rank*, *Doctor.Expertise* and *Doctor.Rank*.

Table 1: Tabular representation of the result set of the running example.

	Hospital						Doctor			
Score	ID	HospitalName	Address	Lat	Long	Rank	ID	Name	Expertise	Rank
0.923	1	Ospedale Nuovo	Via G. Mazzini, 37	45,46331	9,18796	4.3	1	G. Azzoli	Cardiologia	4
0.901	1	Ospedale Nuovo	Via G. Mazzini, 37	45,46331	9,18796	4.3	4	T. Giudici	Cardiologia	3
0.874	1	Ospedale Nuovo	Via G. Mazzini, 37	45,46331	9,18796	4.3	2	S. Brambilla	Ortopedia	5
0.837	2	Ospedale Sacro Cuore	Via Medici, 37	45,46121	9,1807	4.1	3	M. Dell'Orto	Cardiologia	3.5
0.789	2	Ospedale Sacro Cuore	Via Medici, 37	45,46121	9,1807	4.1	5	F. Casiraghi	Allergologia	4.5
0.676	2	Ospedale Sacro Cuore	Via Medici, 37	45,46121	9,1807	4.1	9	G. Martinenghi	Andrologia	2.5
0.556	3	Clinica D.M.S.	Via Bergamini, 12	45,46195	9,19328	2.7	7	S. Secco	Ortopedia	3

Table 2: Resulting values for the attribute ranking indexes.

	Hospital Attributes						Doctor Attributes			
Visual Elements	ID	Name	Address	Lat	Long	Rank	ID	Name	Expertise	Rank
Xaxis	0	0,37	0,2593	0,833	0,615	0,466	0	0,244	0,2815	0,3426
Yaxis	0	0,259	0,2593	0,611	0,615	0,466	0	0,1333	0,2815	0,3426
Taxis	0	0	0	0	0	0	0	0	0	0
Clue	0	0	0	0	0	0,466	0	0	0,2815	0,3426

These preferences refer for example to visualizations where cartesian spaces render values for the two ranking attributes or statistics about doctors' expertise. Given the absence of temporal data, no preferences are computed for the element `TAxis`. A preference for the `Clue` element is given to the two rank attributes, as they are numeric.

Dynamic analysis refines the static scores with the characteristics of the actual data to be rendered. It starts with determining attribute *ranges*. For each interval attribute (e.g., geo-localization, timestamp and numeric), the range of values is determined and a *resolution* is estimated as the minimum distance between two points so that they do not overlap at rendering time. For example, given the *Hospital.Long* attribute, its variability range is $0,01239$ (about 1 km). Considering the limit of 20 points to be represented on an axis, the resolution value therefore suggests that the minimum distance between points should be 0,002478 (about 50 mt). The analysis then proceeds by scoring each attribute according to the following heuristics.

Categorical Attribute Identification looks for repeating values, with the aim of determining whether attributes denote categories. This is useful in order to identify attributes that can be effectively represented as clues. For example, *Doctor.Expertise* has repeating values - several doctors share a same expertise area. After removing the repeated doctor instances, it is still possible to identify repeated values for this attribute. Its *isCategorical* property will be therefore set to true.

Clustered Distribution Quality aims at recognizing the existence of groups of equal (or very closed w.r.t. the attribute resolution) values. For example, 3 groups are identified for the attribute *Hospital.Name*. Groups are ordered according to their cardinality, and the percentage of combinations falling into the first n groups is considered [1]. For *Hospital.Name* 100% of values fall into the identified groups. The clustered distribution, quantified as 0.4, is determined by correlating the number of groups, the percentage of instances falling in the first n groups (the higher the better), and the variance of groups cardinalities (the lower the better). This last factor allows us to weigh the index value with respect to the presence of unbalanced clusters. In our example, the Clustered Distribution Quality heuristics also applies to the *Doctor.Expertise* attribute. The resulting value (0.6) is slightly higher than the one computed for the Hospital attributes, due to the better distribution of values into clusters, which minimizes the group cardinality variance.

Unclustered Distribution Quality is applied to interval attributes and computes the distances between pairs of ordered values and compares them with the attribute resolution. Thus, this heuristic expresses how well distinct values would distribute in the visualization space. For example, the attributes *Hospital.Lat* and *Hospital.Long* do not feature overlapping values; also, the distances between pairs of values is always greater than the resolution computed for the two attributes during static analysis. The distance standard deviation is also low, meaning that the points distribute homogeneously in the visualization space. Overall, the index is relatively high $(0,72)$. Slightly lower values are achieved for *Hospital.Rank* and *Doctor.Rank* ($0,65$ and $0,59$ respectively), due to a higher variance of their value distances. For the attribute *Doctor.Rank* some values are even overlapped.

Attribute ranking aggregates a weighed average of the static and dynamic analysis indexes, with the aim of ranking the different attributes w.r.t. the different elements of the visual model. The result is shown in Table 2.

Data Mapping starts from the attribute ranking and matches attributes to visual dimensions of available templates, so to produce associations such as the one illustrated in Figure 4. Each mapping is scored based on the strength of the attribute and on the match between attributes and template data renderers; then the top-ranked is selected. Match strength also considers constraints and preferences. For example, GPS attributes are always matched in pairs with the X and Y axes. Also, rank and categorical attributes are preferably associated with the clue dimension. For brevity, we exemplify only the mapping of attributes to the map template for the main view; attributes are also mapped to the timeline, vertical list, and cartesian plane templates, but the match score is lower and thus the map is selected. Similarly, for the subview, we only comment the mapping to the vertical list template, which wins over map, timeline, and cartesian plane.

`XAxis`. It is associated with *Hospital.Long*; this attribute

[1]The number n of groups that can be reasonably rendered in a vis. space is heuristically set to 10 in our experiments.

has a high score for the Object and Attribute Type Priority, and a good score for the Unclustered Distribution Quality.

`YAxis`. Due to the previous choice, it is "necessarily" associated with *Hospital.Lat* (geographical coordinates are matched in pairs).

`Clue`. Since `XAxis` and `YAxis` both refer to attributes of a same object (Hospital), then other attributes of this object have priority as visual clues. *Hospital.Rank* has the highest rank and is therefore associated with `Clue.Size`. The data renderer `Clue.Info`, which is a catchall renderer that can be used with attributes of any type, is associated with *Hospital.Name* and *Hospital.Address*. Since there are no categorical attributes for the Hospital object, `Clue.Color` and `Clue.shape` are not instantiated.

`NestedView`. Since all the attributes of Hospital are mapped and none of Doctor, a nested view is created. The mapping algorithm is applied recursively to a data set reduced to the Doctor instances. The vertical list template gets the best matching score: indeed, the `Yaxis` element gets associated with identifying attribute *Doctor.Name*. `Clue.Color` is matched with *Doctor.Expertise*: indeed, being this a categorical attribute, its values can be effectively rendered through different colors. `Clue.Info` gets associated with the remaining attributes (only *Doctor.Rank*)

During the successive **Concrete View Construction**, abstract renderers of the templates instantiated at the previous phase are replaced by concrete widgets. For example, the map-based template is implemented through a Google Map component, which visualizes the geographical attributes, the size clue (by means of circles of different radius) and the nested template (with pop-ups). The local rank of the doctors is rendered graphically by using the stars clue, a representation style supported by the widget.

Once the concrete widget is rendered, the user can perform some filtering actions that trigger the construction of a new view. For example, given the map-based view represented in Figure 4, users might select a specific hospital, focusing their interest on the specialists, so reducing the result to the only Doctor object type. A new generation process therefore starts, taking into account the Doctor's attributes and applying the steps previously described to such attributes. For example, one top-ranked template would consists of the association of the attribute *Doctor.Expertise* with the `XAxis` element, of the *Doctor.Rank* attribute with the `YAxis`, and of the *Doctor.Name* attribute with the `Clue.Info`.

5. PRELIMINARY IMPLEMENTATION

We have implemented a first prototype adding dynamic and adaptive result visualization to the LQ GUI component shown in Figure 1. At server-side, the *query processor* has been extended with an analysis component implemented in Ruby on Rails, including two modules devoted to the static and dynamic analysis. At the client side, a new *UI-builder* module, implemented in JavaScript, assembles the user interface according to the *visualization template*s and to the result of a *data mapping* operation. The current implementation relies on an un-optimized delegation of the data mapping functions to the server side component. The client side also comprises the concrete presentation widgets responsible of rendering the result set and of capturing the user interaction commands. Each widget is implemented as an HTML and Javascript view component, configured and instantiated at run-time by the *view construction* module, according to the result of the calculated *data mappings*.

6. CONCLUSIONS

In this paper we have presented an approach for dynamically creating the visualization of data sets for multi-domain search applications. The data set and the visualization space are modeled in a platform independent way and heuristic transformation rules map the data set model into an abstract visualization model, which is then made concrete by instantiating abstract data renderers with widgets.

Future work will concentrate on the efficient client-side implementation of the dynamic visualization architecture, on the provision of more advanced visualization templates and concrete widgets, such as those discussed in [1], and on the fine tuning of heuristic mapping rules, also with the help of usability studies.

Acknowledgments

This research is part of the Search Computing (SeCo) project, funded by the European Research Council, under the IDEAS Advanced Grants program.

7. REFERENCES

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, 2011.

[2] A. Bozzon et al. Liquid query: multi-domain exploratory search on the web. In *Proc. of WWW*, pages 161–170, 2010.

[3] A. Bozzon et al. Visualization of multi-domain ranked data. In S. Ceri and M. Brambilla, editors, *SeCO Workshop*, volume 6585 of *LNCS*, pages 53–69. Springer, 2010.

[4] T. Catarci et al. *User-Centered Data Management*. Morgan & Claypool, 2010.

[5] S. Ceri et al. Search computing: Managing complex search queries. *Internet Computing, IEEE*, 14(6):14 –22, 2010.

[6] E. Chi. A taxonomy of visualization techniques using the data state reference model. In *INFOVIS*, pages 69–76, 2000.

[7] E. M. Haber, Y. E. Ioannidis, and M. Livny. Foundations of visual metaphors for schema display. *J. Intell. Inf. Syst.*, 3(3/4):263–298, 1994.

[8] J. D. Mackinlay. *Automatic Design of Graphical Presentations*. PhD thesis, Stanford University, 1986.

[9] D. Newman et al. Visualizing search results and document collections using topic maps. *J. Web Sem.*, 8(2-3):169–175, 2010.

[10] C. North. A taxonomy of information visualization user-interfaces, 1998.

[11] A. Rajaraman. Kosmix: Exploring the deep web using taxonomies and categorization. *PVLDB*, 2(2):1524–1529, 2009.

[12] J. Rodrigues, A. Traina, M. de Oliveira, and C. J. Traina. Reviewing data visualization: an analytical taxonomical study. In *IV*, pages 713–720, 2006.

[13] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.