Using Synchronised Tag Clouds for Browsing Data Collections

Alexandre de Spindler, Stefania Leone, Michael Nebeling, Matthias Geel, and Moira C. Norrie

Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{despindler,leone,nebeling,geel,norrie}@inf.ethz.ch

Abstract. Tag clouds have become a popular means of visualising and browsing data, especially in Web 2.0 applications. We show how they can be used to provide flexible and intuitive interfaces to web search services over data collections by using multiple synchronised tag clouds to browse that data. A data collection can have alternative tag clouds and a tag cloud alternative visualisations, with the choice of tag cloud and visualisation at any time controlled by a combination of user selection, developer specification and default system behaviour. A search interface is defined by an augmented data model that specifies the viewer classes, their associated tag clouds and the visualisations of these tag clouds. We demonstrate the approach by describing how we implemented a web application to browse data related to researchers and their publications.

Keywords: search service, tag clouds, data browsing, data visualisation.

1 Introduction

Tag clouds and faceted browsing have been used to address the challenge of providing users with intuitive interfaces to web search services. They offer visualisations of data collections that allow users to construct search queries through simple data selection. While faceted browsing allows complex search queries over a data collection to be constructed in a multi-step refinement process, tag clouds typically support only simple selections. However, the advantage of tag clouds is their capability to represent multiple features of a data collection within a single visualisation.

In this paper, we show how we have extended the use of tag clouds to allow the formulation of complex search queries by developing a browser that can offer multiple synchronised tag clouds to visualise the data stored in one or more data collections. By supporting alternative tag cloud representations for selected data collections within a database as well as alternative visualisations, we are able to combine features of tag clouds and faceted browsing.

The application developer can configure the browser through an extension of the data modelling language that is used to specify the view model of the database. We present an extension of SQL used to define the view model and the process of generating a browser from this model. To demonstrate the approach, we describe how a web application to browse data about researchers and their publications has been implemented.

We begin in Sect. 2 with a more detailed discussion of the background to this work and related research before going on to describe our approach in Sect. 3. Sect. 4 then introduces the SQL extension used to define the view model and shows how it can be used to specify a browser for a particular web search service. Sect. 5 provides details of the system architecture and the process of generating a browser from the view model definition. Implementation details are then given in Sect. 6. We discuss the contributions of our work in Sect. 7, and concluding remarks are given in Sect. 8.

2 Background

Tag clouds have become an extremely popular way of providing visual summaries of data collections and are nowadays used in many Web 2.0 sites to provide a basic search service based on user-generated tags. For example, both Flickr[1] and Del.icio.us[2] provide search services based on collaborative tagging. Although very simple, tag clouds can be used to support search, browsing and recognition as well as forming and presenting impressions [1,9]. In previous work, we have shown that tag clouds can also be used as the basis for a generic database browser [6].

The presentation and layout of tags can be controlled so that features such as the font size, type and colour can be used to give some measure of the importance of a given tag, while the positioning of tags may be based on pure aesthetics, alphabetical sorting or some form of relationship between tags. Studies have experimented with such features and their impact on users, concluding that font size, font weight and intensity are the most important features [7,1]. A study on search performance [8] found that topic-based layouts produced better results than random arrangements, but alphabetic layouts were best.

More recently tag clouds have been proposed as a means of summarising and refining the results of keyword searches over structured as well as unstructured data [5,3,4]. In [5], tag clouds are used to summarise query results of the PubMed biomedical literature database based on words extracted from the abstracts returned by a query. The term *data cloud* is used in [3,4] to refer to their particular adaptation of tag clouds for summarising keyword search results. Data clouds were implemented as part of CourseRank, an application that enables students to search for classes, give comments and ratings, and also organise their classes into a personalised schedule. The developer of a data cloud application specifies how application entities can be composed from the relations in the database in order that keyword search can be applied to entities rather than simple attributes or tuples. The keyword search is based on a traditional information retrieval

[1] `http://www.flickr.com`
[2] `http://www.delicious.com`

approach where entities are considered as documents and attribute values as weighted terms.

At the same time, the use of faceted browsers for web search services has become widespread. Faceted browsing allows items in a data collection to be filtered based on the selection of values of one or more properties of these items. For example, the Flamenco image browser [12] provides facets such as shape, colour, location and date to search and browse image collections. Unlike a simple hierarchical scheme, faceted browsing gives users the ability to find items based on more than one dimension. Another example is yelp.com, a local search and reviewing platform, where users can browse for information of interest using a mix of keyword search and filtering. A user might initially search on keyword and then refine the result collection based on multiple facets which are then refined with every filter selection.

Faceted browsers come in various flavours and have been extended with various features. Facets are usually visualised as ordered lists of possible values, where each value is followed by the number of items associated with that value. Non-directional browsers such as Flamenco [12] and the faceted browser for DBLP[3] offer multiple such lists, from which users can select values as part of the filtering process. In the case of Flamenco, the selection of a value in one facet filters the values of all other facets, but DBLP does not offer such synchronisation of facets. Directional browsers, such as Apple's iTunes, have a specific order of facets, most often represented as columns, where the browsing process goes from left to right. The selection of a facet value in one column, triggers a filtering action on facet values of all subsequent columns. In [11], they extend the column representation of facets with the concept of backward highlighting, where the selection of an item or facet value highlights all possible facet values of precedent columns associated with the current selection and that could have led to that selection. In [10], the information presentation is extended with so called *elastic lists*[4] that visualise the weight cardinality of the facet values. Facets are presented in the form of an ordered list where the size of a facet value indicates the cardinality of information items associated with that value.

Both faceted browsing and tag clouds simplify search processes for users, but have limitations in terms of how they are usually used. Often they only support searches over one particular data collection such as products in the case of online stores or publications in the case of DBLP. While tag clouds offer richer visualisation in terms of being able to encode different properties of a data collection in a single visualisation, clearly only limited information can be visualised at one time and usually they support only very simple selection processes. We propose an approach that combines the features of facets and tag clouds, and extends their use to support more general search services over multiple data collections. Our approach has been inspired by [2], where they provide a search tool to summarise, browse and compare search results over clinical trial data that combines faceted browsing with tag cloud visualisation.

[3] `http://dblp.l3s.de`

[4] `http://well-formed-data.net/experiments/elastic_lists`

3 Approach

Before introducing our view model that developers can use to specify a browser interface for a particular application, we will present an overview of the publications application that we developed in order to explain the main ideas behind the approach. In contrast to our previous work where we aimed to realise a generic database browser based on tag clouds [6], we now focus on providing developers with a framework that enables them to provide users with simple data browsers tailored to a particular application domain.

Fig. 1 shows the initial screenshot of the application when started. The interface comprises three tag clouds—one showing ranges of author names, one showing keywords associated with publications and one showing the names of conferences where papers have been published. In all three cases, the size of the tag is relative to the number of associated publications. Each tag cloud is labelled with the name of the data collection that it visualises, followed by the properties used for the tag cloud visualisation.

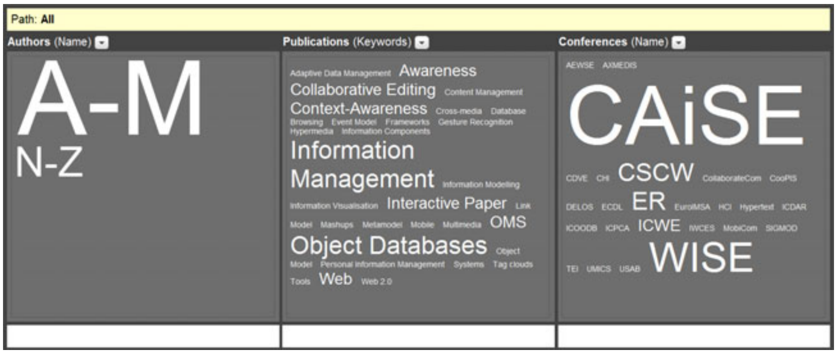

Fig. 1. Initial browser view

In contrast to the conference names and publication keywords, the authors are not represented by their names but by ranges of names instead. This happens if the number of tags shown in a tag cloud exceeds what can be displayed in a browser. In the example, *Authors* initially contains tags for alphabetical ranges "A..M" and "N..Z", where the size is relative to the number of publications aggregated for the respective range of authors. When a user selects one of the ranges, the author names contained in the range are displayed. The intervals used for the ranges can be controlled by the application developer who can specify a threshold limiting the number of tags shown in a tag cloud.

The three tag clouds are synchronised and users can filter data by clicking on a tag in any of the three clouds and the effect will be that values are filtered accordingly in all three clouds. So a user could, for example, click on the tag *CAiSE* in *Conferences* and the *Authors* cloud would then show only the names of people who have authored at least one CAiSE paper, the *Publications* cloud would show only the keywords associated with CAiSE papers and the *Conferences* cloud would show only *CAiSE*.

Presenting single items in one of the tag clouds is useful in terms of showing the context in which the other tag clouds should be interpreted. However, once a tag has been selected there are no further selections possible in that cloud and this is one of the limitations of using tag clouds that we alluded to before when we stated that they can only support simple, single step searches. To support further filtering of *Conferences* based on other properties, an application developer could specify alternative tag clouds for that data collection and these would be available through a dropdown menu. As illustrated in Fig. 2, we also offer a tag cloud for *Conferences* showing the year of conference as well as one showing the title instead of keywords for *Publications*.

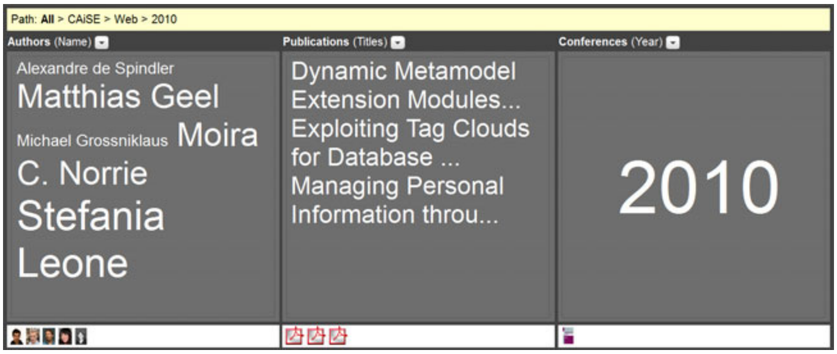
Path: All > CAiSE > Web > 2010
Authors (Name)
Publications (Titles)
Conferences (Year)
Alexandre de Spindler
Matthias Geel
Michael Grossniklaus
Moira C. Norrie
Stefania Leone
Dynamic Metamodel Extension Modules...
Exploiting Tag Clouds for Database ...
Managing Personal Information throu...
2010

Fig. 2. Browsing example

For users to keep track of the selections they have made while browsing the database, we show the navigation path in terms of the tags that have been selected from the tag clouds at the top of the browser. In the example of Fig. 2, the user has first selected the *CAiSE* conference and the keyword *Web* before switching to an alternative view for *Conferences* to select the year *2010*. The user has then also switched the *Publications* view to show the titles instead of the keywords of the resulting three publications. Users can use this kind of breadcrumb navigation to go one or more steps back in the sequence of tag selections or click on *All* to return to the initial browser view shown in Fig. 1.

For each tag cloud, the items resulting from the filtering process are visualised in the designated areas beneath the respective tag cloud if the number of results does not exceed a certain threshold also defined by the application developer. A user can view these items by simply clicking on them. In the case of the *Authors* tag cloud, the author information of a specific author can be accessed. For the *Publications* cloud, publications can be accessed as PDF files and, in the case of the *Conferences* cloud, the conference proceedings can be viewed.

We offer different visualisations for tag clouds so that the developer may choose one appropriate to the information to be displayed and even the task at hand. For example, if the titles of a collection of publications are to be displayed, then it is more appropriate to display these as an ordered list rather than as the sorts of tag clouds one typically sees where several tags are displayed in a single line. We therefore provide a set of basic visualisation types, illustrated in Figure 3, from which a developer may choose. We will describe each of these working from left to right.

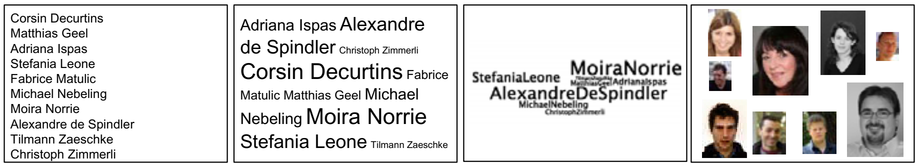

Fig. 3. Visualisation modes for authors

The first visualisation is a simple list of author names, sorted alphabetically by surname and with no variable visual features. The second visualisation is a tag cloud where the tags are aligned horizontally, sorted alphabetically by forename, and their size indicates the number of publications of the author. We refer to this as a line-based visualisation. The third visualisation is similar to ones produced by tools such as Wordle[5] where an advanced algorithm is used to align tags with aesthetics in mind and it can also be used to visualise various forms of relationships between tags. We refer to this as a spiral visualisation since the tag cloud is formed working from a central point and then positioning tags around that point while moving outwards. The fourth visualisation shows that we also support non-textual tags such as images. Any of the three basic visualisation types—list, line-based and spiral—can be used with both textual and non-textual tags.

4 Model and Specification

Our approach builds on the model of a browser that can define multiple synchronised tag clouds to visualise the data stored in one or more data collections. The application developer can configure such a *browser* as a search interface through an augmented data model that specifies the *viewer* classes, their associated *views* in the form of tag clouds and the *visualisations* of these tag clouds. Figure 4 illustrates this concept and how it extends the data model stored in a relational database with a view model based on the shared concept of views. Note that we use the relational model and later SQL, since the majority of web sites build on relational database systems, such as MySQL, for the storage and retrieval of data. However, we note critically that our approach is based on general database principles, such as data tuples and views, and is therefore not tied to a particular database system or modelling language.

[5] `http://www.wordle.net`

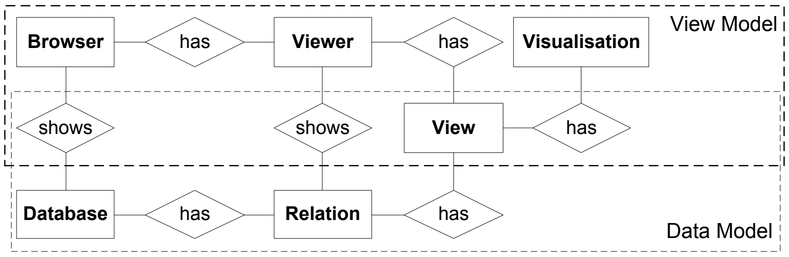

Fig. 4. Extension of the data model with a view model

iN a relational database, the concept of a browser translates to one or more viewer classes used to visualise the tuples stored in a relation. The concept of views is shared by both the data and view model so that viewer classes can associate them with different visualisations, such as simple vertical and horizontal line-based as well as spiral tag clouds. By building directly on the database to specify the view model, many aspects of the visualisations presented in the previous section can be derived directly from the data. For instance, the attribute types of data tuples decide how tags are formatted and displayed, e.g. as text or images, and the number of occurrences of a tag within a data collection determines its size in the visualisation.

To give a concrete example of how a given data model can be augmented to define a search interface using a combination of faceted browsing and tag clouds, Fig. 5 shows a simple domain model for the management of conferences, publications and authors. We will show how the browser and viewer classes corresponding to the interface illustrated in Figures 1 and 2 were defined. The domain model translates to the following relational schema.

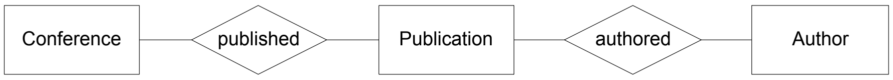

Fig. 5. Example of a simple domain model

Authors (<u>id</u>, first_name, last_name, image)
Publications (<u>id</u>, title, keywords, abstract, image)
Conferences (<u>id</u>, name, year, image)
Authored (<u>author_id, publication_id</u>)
Published (<u>publication_id, conference_id</u>)

The relational schema uses a separate relation to store the tuples, not only for each entity defined in the model above, but also for the foreign key relationships, i.e. Authored and Published, between authors and publications or publications and conferences, respectively. It also covers attributes, such as image, which are primarily used for visualisation when showing the results. In our example from the previous section, the image attribute for authors is used to show a photo, while publications and conference proceedings are represented by a general PDF icon or thumbnail of the front page. Based on this relational schema, Listing 1 now defines the necessary browser and viewer classes using an enhanced version of SQL, which we will describe in more detail later.

```
CREATE VIEWER VR_AUTH (
    "Name"
      (SELECT CONCAT(a.first_name, " ", a.last_name) AS tag, COUNT(ap.
          publication_id) AS count, a.id, ap.author_id FROM Authors AS a,
      Authored AS ap WHERE a.id = ap.author_id GROUP BY ap.publication_id
          RANGE 15)
      LINE MULTISELECT,
);

CREATE VIEWER VR_PUB (
    "Titles"
      (SELECT title AS tag FROM Publications LIMIT 30)
      LIST,
    "Keywords"
      (SELECT SPLIT(keywords) AS tag FROM Publications LIMIT 100 ORDER BY tag
          ASC)
    LINE,
);

CREATE VIEWER VR_CONF (
  "Name"
    (SELECT c.name AS tag, COUNT(p.id) AS count, c.id, p.conference_id FROM
        Conferences AS c, Publications AS p WHERE c.id = p.conference_id
        GROUP BY p.id ORDER BY c.name ASC)
    LINE,
  "Year"
    (SELECT c.year AS tag, COUNT(c.year) AS count, c.id, p.conference_id FROM
         Conferences AS c, Publications AS p  WHERE c.id = p.conference_id
        GROUP BY c.year ORDER BY c.year DESC)
    SPIRAL,
);

CREATE BROWSER B_PUBLICATIONS (
  "Publications" VR_PUB,
  "Authors" VR_AUTH,
  "Conferences" VR_CONF,
);
```

Listing 1. Example browser in SQL

The first viewer class, VR_AUTH, defines a single view over all authors with each tag built using SQL's standard CONCAT function to combine the first and last names. The tag size is calculated using an SQL count over the authored publications. This view is then associated with a horizontal, line-based tag cloud visualisation that will use count to make the size of an author's name dependent on the number of publications that they have. The viewer also allows for multiple selections of authors so that publications authored or co-authored by selected authors will be shown. Additionally, we use ranges in the case that more than 15 authors are displayed in the cloud. The next viewer, VR_PUB, associates a view over all titles of the publications with a vertical list visualisation. An alternative view of the publications is defined as the top 100 keywords in a line-based tag cloud. Here we use a non-standard SQL function SPLIT that we have defined to parse a comma-separated VARCHAR value and return the set of tokens. VR_CONF defines a primary view for conferences by name, where the size of the tag is relative to the number of publications in that conference. A variation here is to use an advanced, spiral tag cloud visualisation to show the publications by year starting from the latest conference, where the size of the tag is relative to the number of publications in the year. Finally, the browser B_PUBLICATIONS defines the search interface with the three viewer classes.

```
CREATE VIEW <view_name> (tag, [count, <other_columns>]) AS (SELECT <column>
    AS tag[, COUNT(<column>) AS count, <other_columns>] FROM <table_1>[, <
    other_tables>] [GROUP BY <column>] [ORDER BY <column> ASC|DESC] [LIMIT <
    number>] [RANGE <number>]);

CREATE VIEWER <viewer_name> (
    "View Name" <view_name>|<inner_view_definition> LIST|LINE|SPIRAL|<
        other_visualisations> [MULTISELECT],
    [<other_views>]
);

CREATE BROWSER <browser_name> (
    "Viewer Name" <viewer_name>,
    [<other_viewers>]
);
```

Listing 2. Extended SQL to specify browsers and viewer classes based on views

Listing 2 gives more details of the extended SQL syntax used above to define the view model. In SQL, views are essentially named SELECT statements that represent stored queries in the form of a virtual table composed of the respective result sets. We build on this concept of views to enable different visualisations of the data. Note that views can either be defined as an inner view as part of the viewer class or referenced by name, which enables re-use and combinations of views. For the proposed visualisations, the developer is required to specify a reserved column tag that will represent the tuples used to display the tags in the tag cloud visualisation. If the tags are to be displayed in different sizes according to certain criteria, then a second reserved column count is required that can build on SQL's COUNT aggregator function to count the occurrences of different values for a given column. In that case, also the GROUP BY statement is required to group the result set by the aggregated values. Note that other SQL statements such as ORDER BY and LIMIT can be used to sort tags in ascending or descending order as well as to limit the amount of tags displayed. Additionally, we define the RANGE statement to display the range of values rather than all retrieved values if the number of tags returned for the query exceeds the specified number. This can be helpful to navigate through large amounts of tags within a single view, e.g. by first showing ranges A..E, F..J and so forth, and, upon selection, showing the names of the respective subset of authors. Such ranges can be built by a custom SQL function that we defined to first sort all retrieved tags and then divide them into categories. For example, in the case of type VARCHAR, ranges could be built from only the first letters of all tags. Finally, the set of values used to display and size the tags typically comes from different columns and not necessarily from the same entity, e.g. to use a larger font for authors the more publications they have. While other_columns will then be required for joining associated relations, they will be ignored by the default tag cloud visualisations. On the other hand, additional columns offer a simple way of allowing for extensions and refinements. For example, new reserved columns, such as color, could be introduced to extend the proposed visualisations and visually group the result set by a specified range of colours.

In addition to this augmentation of view definitions, we further extend SQL with VIEWER and BROWSER definitions, respectively. A viewer class defines a set of alternative views, each of which is associated with a name displayed for the user to switch between visualisations, and a combination of parameters LIST, LINE or SPIRAL and MULTISELECT. The first three determine which of the visualisations shown in Fig. 3 will be used, where LIST represents the vertical alignment of tags, LINE a horizontal, line-based visualisation and SPIRAL the advanced tag cloud visualisation. Again, other visualisations could be supported by introducing new parameters that represent the respective visualisations. If the optional parameter MULTISELECT is provided, then the associated visualisation must allow for multiple selection of tags. With multiple selection, a combination of conjunctive queries between and disjunctive queries within views can be supported. Finally, a BROWSER defines a set of viewer classes and also provides a display name for each of them.

By using these augmentations of SQL, we can build on established database concepts and directly benefit from the rich support for SQL expressions and functions such as COUNT. Moreover, the caching strategies and high performance of query execution in many database management systems, such as MySQL, makes it optimal for web search interfaces. The way in which the final presentation of the tag clouds is generated as well as how the synchronisation between views on selection of a particular tag works are discussed in the next section.

5 Framework

Having described our language extension, we now present a framework that can process such browser specifications and generate a browser interface to search and browse specific data collections. The framework is shown in Fig. 6 in terms of its main components and their interactions. The browser defined in terms of an extended SQL specification is provided as input to the framework ①. The framework processes this specification as follows. First, a document template representing the browser's web interface is generated ②. This document contains one designated placeholder for each viewer, in which the tag clouds will be inserted at application runtime. Second, a browser-specific SQL view manager is created ③ based on the viewer specifications that will create the SQL views specified for each viewer in the database. At run-time, the SQL view manager queries these views ④ to retrieve the tags and their sizes which then provide the necessary input for the tag cloud generator that is called to create the tag clouds and the associated visualisations ⑤. In a final step, these generated tag clouds are inserted into the placeholders of the document template ⑥, which yields the final browser interface presented to the user.

When a user selects a tag in one of the tag clouds, the current view associated with that tag cloud as well as the associated views of all other tag clouds are

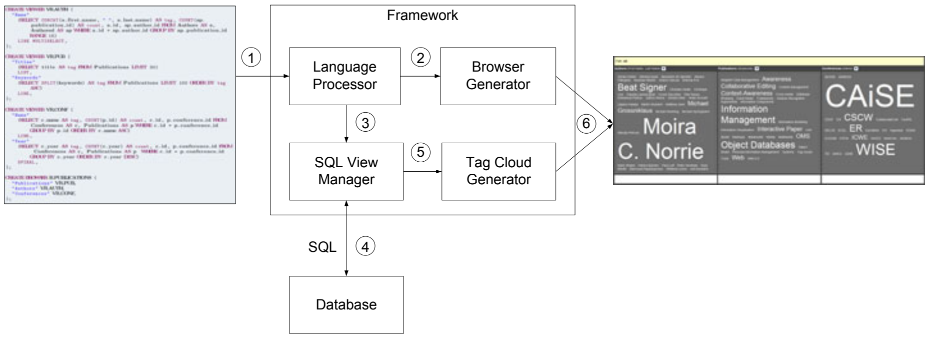

Fig. 6. Framework architecture and workflow

synchronised according to the selection. For this purpose, the view manager first restricts the current view by temporarily extending the WHERE clause in order to reflect the user selection. This updated view is then used as the starting point for the PROPAGATE-UPDATE function shown in Fig. 7, which implements an algorithm propagating the tag selection to associated views in order to keep them synchronised.

```
PROPAGATE-UPDATE($View$)
1  $N \leftarrow \emptyset$
2  $N \leftarrow$ GET-ASSOCIATED-VIEWS($View$)
3  **for** $\forall$ n $\in N$
4  **do** ALTER-VIEW($View$,n)
5      PROPAGATE-UPDATE(n)
```

Fig. 7. Update Propagation Algorithm

For the view passed as the argument, the set N of all associated views are retrieved using the GET-ASSOCIATED-VIEWS function. For every view $n \in N$, the view creation statement is extended by a join operation with respect to the view argument. Such extensions are carried out by the ALTER-VIEW function. Then, the PROPAGATE-UPDATE function is invoked recursively, in order to propagate the selection to all views related to the one currently processed. Note that, if multiple tags are selected subsequently, the algorithm is executed for each affected entity.

As a result, the selection of a tag is propagated along the relationships among database relations and, therefore, all related tag clouds are synchronised. For example, if a conference tag is selected, the view associated with the tag cloud is extended in order to filter the selected conference. Next, all associated views are determined, which, in our publication browser example, would be the publications and authors views. Then, the publications view would be filtered for those publications that were published in the selected conference. Finally, the authors view is extended in order to retrieve only those authors who have a publication at the selected conference.

In general, the sequence of views to be extended is determined by starting with the view in which the tag selection occurred and then following the relationships in a breadth-first manner. Note that this propagation algorithm was designed to work with data models that can be represented as connected and acyclic graphs. However, if there were cycles, endless loops are avoided because the framework keeps track of the views already extended. If there is a viewer showing a database relation not connected to any other, this viewer is independent and therefore cannot be synchronised.

As users continue selecting tags, the cumulated selections are propagated individually and in the same order as they were made by the users. Finally, if users make multiple disjunctive selections at once, the WHERE clause of the respective view is extended with all selection criteria combined in a disjunctive manner. Similarly, if a tag representing a range is selected, all values contained in this range are taken as disjunctive selection criteria.

6 Implementation

We now present how the framework was implemented in the form of a web application. We used a standard Client/Server setup consisting of HTML, CSS and JavaScript on the client side and PHP and MySQL on the server side. The web application provides an administration page where developers can input and execute a browser specification in extended SQL to generate a new browser that is then available from a new URL. Users may then interact with the browser as described in the previous sections.

Figure 8 shows the PHP classes which are involved in the creation of a browser as well as in processing user interactions at runtime. The specification of a browser in extended SQL is handled by the method `generateBrowser` declared in the `LanguageProcessor` class as follows. First, the information required to generate the HTML document template is extracted, which includes the number of viewers, their names and contained views. This information is passed on to the `generate` method defined in the `BrowserGenerator` class in order to create the client-side browser interface. It consists of the top bar for the breadcrumb navigation, the viewers with their names, the views and the dropdown menus for the selection of alternative views, the placeholders for the tag clouds and the bottom bar for the result sets.

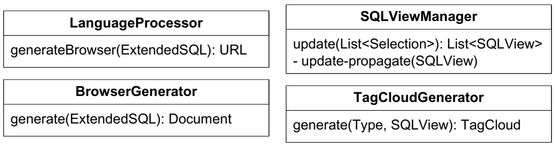
LanguageProcessor
generateBrowser(ExtendedSQL): URL
SQLViewManager
update(List<Selection>): List<SQLView>
- update-propagate(SQLView)
BrowserGenerator
generate(ExtendedSQL): Document
TagCloudGenerator
generate(Type, SQLView): TagCloud

Fig. 8. Classes implementing the framework

Second, the SQL view definitions are extracted from the extended SQL. The respective views are created in the database and their names are stored in a separate database relation from where they can be accessed at application runtime. Finally, a new folder is created on the server, containing the generated browser interface and a PHP script `index.php` responsible for processing user tag selections and returning viewer contents where tag clouds reflecting user selections are dynamically updated at runtime. The URL returned by the `generateBrowser` method in the `LanguageProcessor` class points to this folder. The generated browser interface can then be tailored and styled according to specific application requirements.

The initial tag clouds presented to the users consist of tags that are HTML links. These links point to the `index.php` file created for the current browser, and the selection to be carried out when a particular link is chosen by the user is appended as a query string. The following example URL is the target of a link associated to the *CAiSE* tag in the *Conferences* viewer.

```
index.php?Conferences=CAiSE
```

Such a request is processed on the server side by the `SQLViewManager` class. Its method `update` takes the selection contained in the query string as a parameter and performs the update propagation algorithm described in the previous section. As a result of this update propagation, all extended SQL views reflecting the user selection are created. For each of these extended SQL views, the `generate` method in the `TagCloudGenerator` class creates an updated tag cloud which is merged with the document template and returned to the client. The URLs in the links of these updated tag clouds contain the previous selection in the query string as well as the subsequent selection they represent. For example, the URL of a link associated to the author Matthias Geel would be written as follows.

```
index.php?Conferences=CAiSE&Authors=Matthias%20Geel
```

For each tag selection specified by a user, the URLs of the links in the updated tag clouds are extended in order to contain all previous selections as well as the one to be carried out if the link was followed.

Similarly, the breadcrumb navigation consists of links pointing to the URLs previously requested. Due to the fact that our implementation follows a stateless approach, the implementation of the breadcrumb navigation is a simple manner of creating URLs including the respective query strings.

In order to support multiple disjunctive tag selections at once, the user can switch to a *multi select* mode. In this mode, the selection of a tag does not immediately initiate a request to the server. Instead, a search button is added to the browser interface which triggers the request to the server when the user is finished selecting tags.

7 Discussion

We have presented a general framework that supports the configuration of search interfaces for browsing and querying data collections using multiple synchronised tag clouds. We have illustrated its use based on the example of browsing a publication collection. Such interfaces could support a web search service either of a single research group's publications or over an entire digital library—simply by adapting the specification. While there are faceted search interfaces to publication collections, such as DBLP, our approach is much more flexible, since it not only supports searching for publications, but users can also shift their search focus to other entities of interest, such as authors or conferences. Furthermore, the selection of the visualised entities, their relationship and alternative tag cloud visualisations are configurable based on a combination of user selection, developer specification and default system behaviour.

Our approach is not dissimilar to the one taken by [2], where they provide a domain-specific tool for searching semi-structured clinical trial data where a set of predefined categories are represented using tag clouds. As with standard faceted browsing, users can start with a keyword search and the number of relevant documents are returned as a list, which can be further refined using the tag clouds. The selection of a tag in one dimension triggers the synchronisation of the tag clouds representing all other dimensions, as well as the filtering of the search result. While our approach could be seen as a generalisation of their work as we propose an augmented data model and a framework that supports the configuration of search interfaces for a domain of choice, it is also important to highlight the differences. Their interface consists of a set of predefined facets represented as a tag cloud, while we offer configurability at the interface level through dropdown menus that allow the selection of other tag cloud representations of the same entity. Furthermore, their data model corresponds to a typical data model underlying faceted browsing that is often based on star or multi-dimensional schemas, while our synchronised tag clouds do not evolve around a particular pivot entity. This means that there is no central entity that all other dimensions depend upon. In addition, with our approach, the tag size can be configured to represent dependencies to other entities of interest or simply the occurrence of a specific term, while with their approach the tag size always refers to the number of occurrences of a term in relation to the entity of interest, which in their case is clinical trial data. However, there are also some restrictions to the database schemas we support. The schema has to be a connected acyclic graph in order for our propagation algorithm to calculate the tags for each viewer correctly. While with cyclic structures, the propagation algorithm simply uses a shortest path approach, we could extend our framework so that a developer could configure the algorithm to achieve a different behaviour, if desired.

We note that our current implementation follows a stateless approach. This has some implications on system performance. Users can always choose to navigate to a breadcrumb, which is a bookmark to an individual search and allows a user to continue from there. With our current approach, these queries are executed again, invoking the propagation algorithm to adapt all adjacent tag clouds, while with a stateful approach these views could simply be cached. However, such an approach would be memory-intensive since it requires these views to be materialised.

8 Conclusions

We have presented an approach for browsing and searching data collections based on an extended data model that supports the configuration of a synchronised tag cloud browser for a domain of choice and we have illustrated its use through a publication browser. The generation of the browser is automated and its configuration is a mix of developer configuration using the extended SQL syntax, system default behaviour and user selection. We are also planning a user study to compare our approach to regular web search interfaces as well as faceted browsers.

References

1. Bateman, S., Gutwin, C., Nacenta, M.: Seeing Things in the Clouds: The Effect of Visual Features on Tag Cloud Selections. In: Proc. ACM Conf. on Hypertext and Hypermedia, HT 2008, pp. 193–202 (2008)
2. Hernandez, M.E., Falconer, S.M., Storey, M.A., Carini, S., Sim, I.: Synchronized Tag Clouds for Exploring Semi-Structured Clinical Trial Data. In: Proc. Conf. of the Center for Advanced Studies on Collaborative Research (CASCON 2008), pp. 42–56 (2008)
3. Koutrika, G., Zadeh, Z.M., Garcia-Molina, H.: Data Clouds: Summarizing Keyword Search Results over Structured Data. In: Proc. Intl. Conf. on Extending Database Technology (EDBT 2009), pp. 391–402 (2009)
4. Koutrika, G., Zadeh, Z.M., Garcia-Molina, H.: CourseCloud: Summarizing and Refining Keyword Searches over Structured Data. In: Proc. Intl. Conf. on Extending Database Technology (EDBT 2009), pp. 1132–1135 (2009)
5. Kuo, B.Y.L., Hentrich, T., Good, B.M., Wilkinson, M.D.: Tag Clouds for Summarizing Web Search Results. In: Proc. Intl. Conf. on World Wide Web (WWW 2007), pp. 1203–1204 (2007)
6. Leone, S., Geel, M., Müller, C., Norrie, M.C.: Exploiting tag clouds for database browsing and querying. In: Information Systems Evolution. LNBIP, vol. 72, pp. 15–28 (2011)
7. Rivadeneira, A.W., Gruen, D.M., Muller, M.J., Millen, D.R.: Getting our Head in the Clouds: Toward Evaluation Studies of Tag Clouds. In: Proc. Intl. Conf. on Human Factors in Computing Systems (CHI 2007), pp. 995–998 (2007)
8. Schrammel, J., Leitner, M., Tscheligi, M.: Semantically Structured Tag Clouds: An Empirical Evaluation of Clustered Presentation Approaches. In: Proc. Intl. Conf. on Human Factors in Computing Systems (CHI 2009), pp. 2037–2040 (2009)
9. de Spindler, A., Leone, S., Geel, M., Norrie, M.C.: Using Tag Clouds to Promote Community Awareness in Research Environments. In: Luo, Y. (ed.) CDVE 2010. LNCS, vol. 6240, pp. 3–10. Springer, Heidelberg (2010)
10. Stefaner, M., Muller, B.: Elastic Lists for Facet Browsers. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 217–221. Springer, Heidelberg (2007)
11. Wilson, M.L., André, P., Schraefel, m.c.: Backward Highlighting: Enhancing Faceted Search. In: Proc. ACM Symposium on User Interface Software and Technology (UIST 2008), pp. 235–238 (2008)
12. Yee, K.P., Swearingen, K., Li, K., Hearst, M.: Faceted Metadata for Image Search and Browsing. In: Proc. ACM Intl. Conf. on Human-Computer Interaction (CHI 2003), pp. 401–408 (2003)