UiB | Dept. of Information Science and Media Studies
UiTø | Dept. of Computer Science
NTNU | Dept. of Computer and Information Science
Telenor | Telenor R&I

CAIM
CONTEXT-AWARE IMAGE MANAGEMENT

VISI2

Combining CBIR Search with Text and GPS Location Criteria

Anders Rørvik

Sept.2, 2008

CAIM-UiB
TR #6

Table of Contents

1. Introduction

The task for the author this summer was to extend the functionality of the VISI (Vortex Image Search Interface) which is available at http://bulmeurt.uib.no:8500/caim/VISI. Previously VISI only had support for CBIR (Content-Based Image Retrieval), however it now also has support for TBIR (Text-Based Image Retrieval), as well as a combination of CBIR/TBIR and CBIR/GPS. It is also possible to perform a stand alone GPS search.
VISI2 is implemented in a proprietary tag-based language called Coldfusion, developed by Adobe. Furthermore VISI2 relies on an Oracle 10g database serving as a back end for data storage and offering CBIR functionality. The new version of VISI also makes use of a full text search engine called Verity, which allows building and maintaining indexes based on stored text.

2. Verity

Verity is the name of the company who developed the full-text search technology used by Coldfusion, thus full-text searching in Coldfusion is often referred to as «Verity searching». Examples of other full-text search engines can for instance be Google, Altavista and Yahoo. A full text search engine must include one or more collections which can be searched. A Verity collection is simply a data repository / data structure for the data we want to store.

The following is an example of how a Verity collection can be created.

```
<cftry>
<cfcollection action="create" collection="gps" path="/opt/coldfusion8/verity/collections">
<cfcatch type="any">
       <cflocation url="../error.cfm?type=collectionerror" addtoken="no">
</cfcatch>
</cftry>
```

Note that an exception is cast if the collection already exits, and thus an entry for this exception has been added to the VISI2 error handler. It is also possible to create Verity collections through the Coldfusion administrator interface.

2.1 Populating Verity Collections

Furthermore the collection needs to be populated with data. Coldfusion supports 3 different methods for constructing/populating an index.

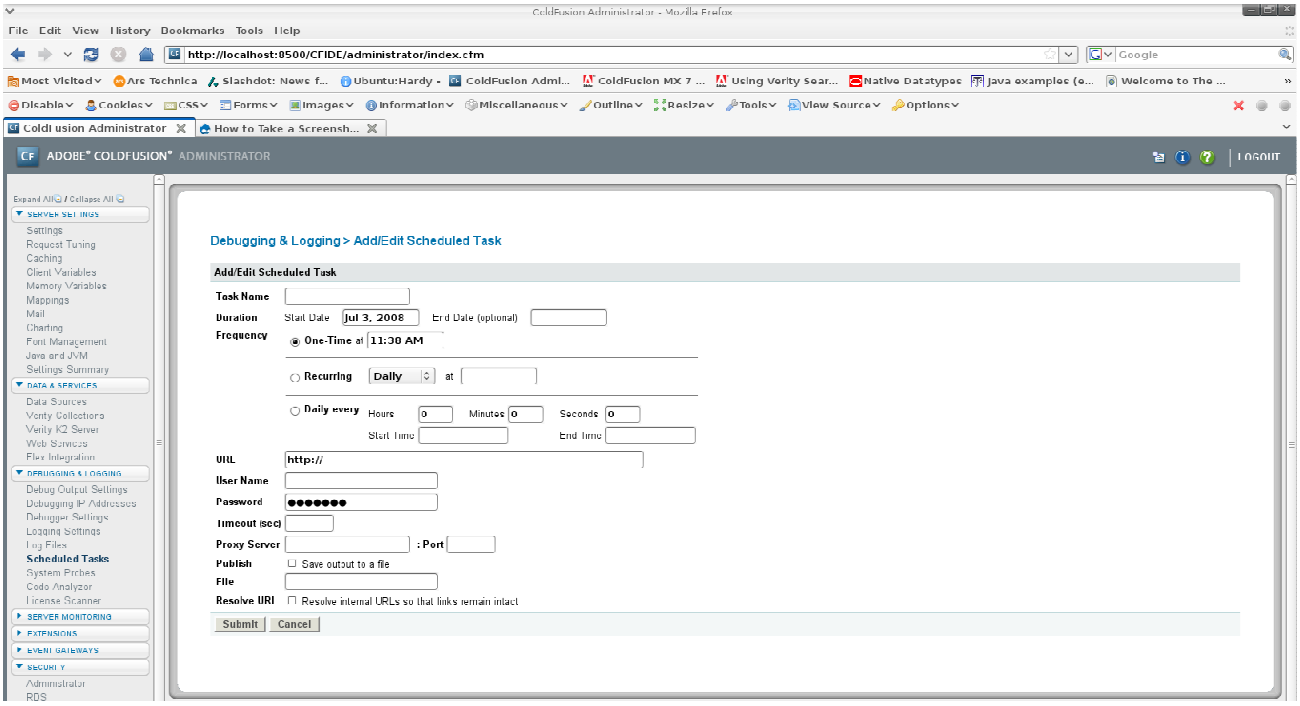
1) Search against all files in a given directory
2) The Verity Spider feature – A collection can be populated by pointing Verity to a website where it will follow all the links on the respective page (and the links the next page leads to).
3) The third approach is based on indexing query result sets.

The latter approach is utilized in VISI2. The metadata about objects and pictures is stored in an Oracle database, and populating the index is done by issuing a query, retrieving all the metadata and putting it into collection as shown below. The code indentation is not the best due to the dimensions of the table. Please view code examples in a proper editor for increased readability. This is merely a reference.

```
<cfquery name="populateIndex" datasource="#Application.db_odbc#">

SELECT DEREF(object).id as objectid, DEREF(object).name as objectname,
       DEREF(object).material as objectmaterial,DEREF(image).id as imageid,
       DEREF(image).caption as imagecaption
FROM contains

</cfquery>
<!--- Updates the collection with the above query results
key=primary key column of the data source table
title=specifies a query column name
body=columns that you want to search for the index
type=if set to custom, this attribute specifies the columns that you want to index. If set to file
or path, this is a column that contains either a directory path and filename, or a directory path
that contains the documents to be indexed.

The custom attributes have to be refererred to as customN... if not, coldfusion will not accept
them. More information can be found
here: http://livedocs.adobe.com/coldfusion/6/CFML_Reference/Tags-pt3.htm

--->

<cfindex
              query="populateIndex"
              collection="keyword"
              action="Update"
              type="Custom"
              key="objectid"
              title="name"
              body="objectid,imageid,objectname,objectmaterial,imagecaption"
              custom1="objectid"
              custom2="objectname"
              language="bokmal"
              />
```

2.2 Optimizing Verity Collections

The Verity collections can, and should be optimized as they grow. It is common to optimize these during low-peak periods (ie. Night time). This task can be accomplished by using the feature scheduling tool in the Coldfusion administrative interface.



Optimizing a collection does not require a lot of effort. Actually it only requires one line of code.

```
<cfcollection action="optimize" collection="gps">
```

To optimize at «low peak» hours, simply set the task to recurring, and the time of the task in the schedule interface. Furthermore the template that contains the above line of code must also be specified. Note: Optimizing a big collection can take a long time, the timeout variable should therefore be set to a high value.

2.3 Searching a Verity Collection

It is the author's opinion that doing searches against a Verity collection is far easier than expressing the same queries using SQL directly to the database. Verity also assigns a score to the matches of a query, and thus a ranked result is returned. The follow snippet of code illustrates how a search is performed on a Verity collection. The “Lcase” function simply converts a string to lowercase. If you give mixed case input to Verity it will make a case-sensitive search. #Form.keywords# retrieves the POST data from the search form.

```
<cfsearch
              collection="keywords"
              name="search"
              criteria="#LCase(Form.keywords)#"
              maxrows="20"
>
```

It is also possible to query an index based on the GPS coordinates of objects.

2.4 Charsets & Language packs in Verity

The author experienced problems when attempting to index words containing Norwegian characters like «øåæ». It turns out that unless you specify a language when creating a collection, it defaults to English. However Coldfusion Server only ships with the «English» and the «English Advanced» language packs, though it is possible to download an extended language pack from Adobe[1]. This package also enables you to run a standalone Verity Server.

> Note that the operators «AND, OR, NOT» must be translated to the respective language you are using, according to Adobe

2.5 Collections used by VISI2

During the development of VISI2 it was necessary to index two types of metadata. One of the types involved various GPS data concerning the objects in the OBJECT table. It was therefore natural to put this metadata in a separate collection. Furthermore it was necessary to index object names and image captions, it seemed like a natural choice to put this metadata in a separate Verity collection, thus leaving VISI2 with the following collections: “gps” and “keywords”.

3.0 TBIR

A functional TBIR(Text Based Image Retrieval) search was one of the main goals for the CAIM summer project. Verity enables a user to do a full text search from Coldfusion, and therefore this solution was selected. A Verity search can be performed in two ways, explicit or simple. Doing this explicitly implies that the programmer has to perform stemming etc. himself. Explicit searches might produce better results than the simple approach, however the simple search is sufficient for what needs to be accomplished in VISI2. See section 2.3 for an example of how a simple search against a Verity collection can be done. Search terms need to be expressed in a certain syntax, an URL[2] that points to the Adobe documentation for this functionality is supplied through the VISI2 interface.

1 http://www.adobe.com/support/coldfusion/downloads.html
2 http://livedocs.adobe.com/coldfusion/6.1/htmldocs/indexsa3.htm#wp1212738

4.0 CBIR/TBIR

One of the main goals for the summer project was to implement CBIR/TBIR functionality in VISI2. It was assumed that such a combination would lead to better recall/precision values. Judging by the preliminary results this definitely appears to be the case.

The CBIR part of the search is performed with the functionality that exists in the “ORDSYS” package created by Oracle. A signature is generated for the seed image, which in turn is specified and uploaded by the user of VISI2. The seed image is then matched against all images within the image table. Scores are generated and inserted into a temporary table called CBIRTBIR_TEMP. The following PL/SQL procedure is a modified version of the one written by Bjørge Næss in 2007.

```
/* Some code has been left out, please see Appendix C for further reference */

        BEGIN
                -- Importing seed image
                IF NOT IMPORT_SEED(p_Url, p_Filename, v_SeedImageId) THEN
                RAISE e_SeedImportFail;
            END IF;
                v_Weights := 'color = "'||p_Color||'", texture = "'||p_Texture||'", shape = "'||p_Texture||'",
                location = "'||p_Spatial||'"';
                p_Results := CBIR_TBIR_IMAGE_SEARCH(v_SeedImageId, v_Weights, p_Threshold);

                EXCEPTION
                  WHEN e_SeedImportFail THEN
                   p_Results := '-2';
                  WHEN OTHERS THEN
                   p_Results :='-3';
END CBIR_TBIR;
```

As you can see the p_Results variable is filled with the results of the function call to CBIR_TBIR_IMAGE_SEARCH. This is also a modified version of a function written by Næss in 2007.

```
/* Some code has been left out, please see Appendix C for further reference */
      LOOP
         INSERT INTO CBIRTBIR_TEMP VALUES(iterator.ID,iterator.Score,0,0);
         IF (v_ReturnString IS NULL) THEN v_ReturnString := iterator.ID;
         ELSE v_ReturnString := v_ReturnString ||','|| iterator.ID;
         END IF;
      END LOOP;
      RETURN v_ReturnString;
  END;
```

The first sentence in the LOOP inserts a row with the picture id, as well as the generated score into the CBIRTBIR_TEMP table. The two latter values point to the Verity score and the total score which in turn will be used by a final procedure to generate the total score for the CBIR/TBIR search.

The Verity search engine is part of the Coldfusion environment, therefore the Verity search will take place in a Coldfusion template. This is done by using the <cfsearch> tag, which receives input from the HTML form in VISI2. A similar example of this can be found in section 2.3 «Searching a Verity Collection».

The author decided to index metadata from the table “contains”. This table keeps track of objects that appear in one or more images. Note that the objects within this table are actually references, however gathering metadata from this table makes sure that the image actually contains one or more objects, unless a user error has occurred. At the present time the ids of the objects that match the query are stored in an array, which in turn is used to fetch the images depicting their respective objects.

As Verity does not generate a score for the image ids retrieved, we use the score generated by the initial search. An array is constructed to hold the ids of the images found. Note that there is a primary key constraint on the id field in the CBIRTBIR_TEMP table, therefore an exception will be cast if one tries to insert duplicate values. However, this is handled by the catch block, which performs an update on the appropriate row.

```
<!--- Updates the temporary table with the Verity scores --->
<cfloop index="Counter" from=1 to="#ArrayLen(allimagesofobjects)#">
              <cftry>
              <cfquery name="update" datasource="#Application.db_odbc#">
              INSERT INTO CBIRTBIR_TEMP VALUES(#allimagesofobjects[Counter]#,0,#search.score#,0)
              </cfquery>
              <cfcatch type="Database">
                     <cfquery name="update2" datasource="#Application.db_odbc#">
                            UPDATE CBIRTBIR_TEMP SET verityscore=#search.score# WHERE
id=#allimagesofobjects[Counter]#
                     </cfquery>
              </cfcatch>
              </cftry>
</cfloop>
```

Finally, the results are ready for processing, this is done by calling the PROCESS_SCORES procedure which iterates over the CBIRTBIR_TEMP table and calculates the total score column for all rows. Please note that Verity returns scores between 0 and 1 where 1 is the best, and Oracle between 0 and 100 where 1 is the best, hence we calculate (1-(verityscore))*100 to get this in “Oracle format”.

```
/* Some code has been left out, please see Appendix C for further reference */
 for iterator IN (
 --- Fetches the values that we calculate a total score from ---
          select oraclescore, verityscore,id
          from CBIRTBIR_TEMP
 order by v_totalscore
 )
 LOOP
 v_totalscore:=(((1-(iterator.verityscore))*100) + (iterator.oraclescore)) / 2;
 UPDATE CBIRTBIR_temp set totalscore=v_totalscore where id=iterator.id;
 END LOOP;
END PROCESS_CBIRTBIR_SCORES;
```

A sorted result can then be retrieved for the template in charge of displaying the images: «searchresultcbirtbir.cfm». The lower the score, the better the result.

```
<cfquery name="fetchprocessedresults" datasource="#Application.db_odbc#">
        SELECT id,totalscore
        FROM CBIRTBIR_TEMP
        ORDER BY totalscore ASC
</cfquery>
```

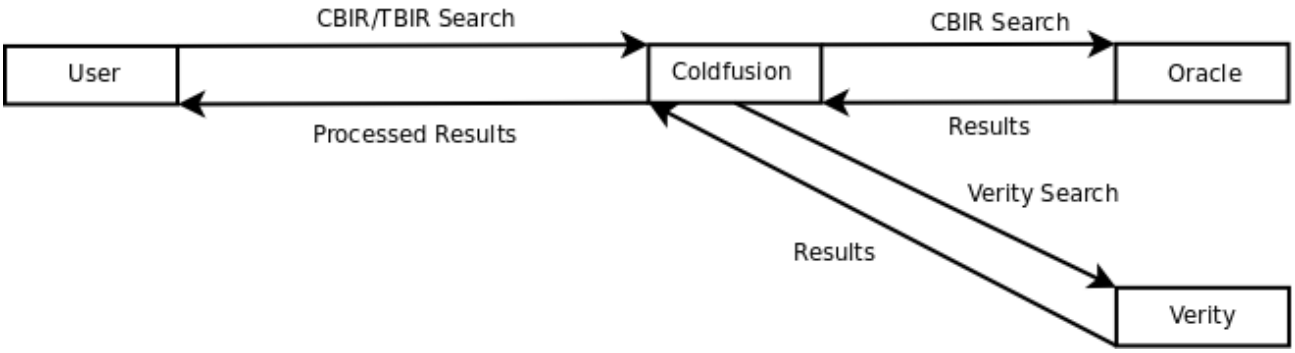
The result of the query is assigned to a session variable for processing on the result page.

```
<cfset Session.imageids = ListToArray(ValueList(fetchprocessedresults.id,","))>
```

Furthermore it is necessary to find the object id of the image with the lowest score. This id is used to display the necessary metadata on the result page. This is done by issuing the queries below.

```
<!--- This query fetches the imageid with the lowest score, this is used to retrieve the objectid
--->

<cfquery name="fetchimageidwithlowestscore" datasource="#Application.db_odbc#">
        SELECT a.id as id
        FROM CBIRTBIR_TEMP a
        WHERE totalscore = (SELECT min(b.totalscore)
        FROM CBIRTBIR_TEMP b)
</cfquery>

<!--- Fetches the objectid corresponding to an imageid --->

<cfquery name="fetchobjectid" datasource="#Application.db_odbc#">
        SELECT DEREF(object).id as id
        FROM CONTAINS
        WHERE DEREF(image).id=#fetchimageidwithlowestscore.id#
</cfquery>
```

4.1 Flow chart



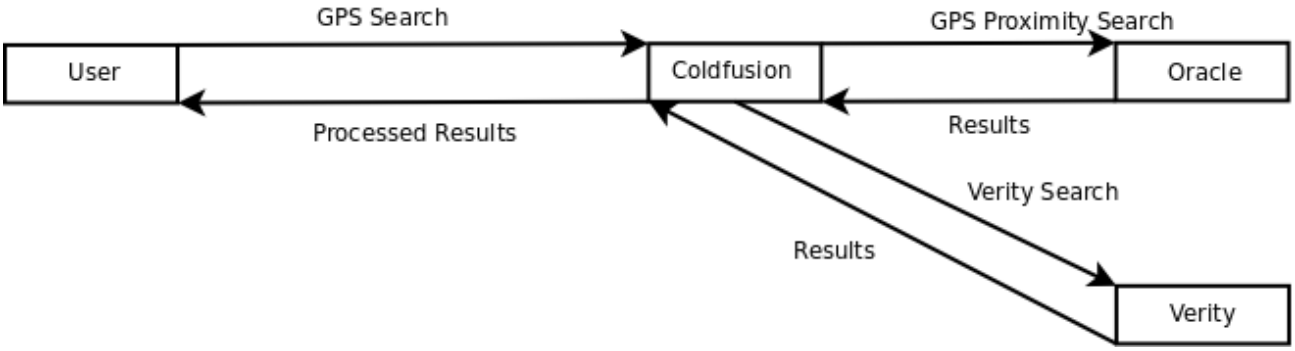
5.0 GPS Search

VISI2 also features image searching based on GPS coordinates. All objects added in the “BergenBy” database hold a set of GPS coordinates which are index in Verity. However, these coordinates only represent the actual object, that is, not the area where one usually would expect a tourist to take photographs from.

The GPS search functionality in VISI2 preliminary searches the Verity GPS index for objects that match the input coordinates. However as stated above, this will most likely fail. Nevertheless a user can input wildcards in a set of GPS coordinates, and still get a match from Verity. Note that the Oracle search will fail if this is the case, though VISI2 can do “unprecise” searches. This is addressed by VISI2 by accepting an additional parameter, “distance”. This parameter is in turn passed on to a PL/SQL procedure which makes use of Oracle's API to discover adjacent locations based on a set of coordinates and a distance(currently measured in meters). An example of such a procedure can be found in Appendix C or section 5 (just below). Finally a procedure is called to calculate the total score.

```
/* Some code has been left out, please see Appendix C for further reference */
  for iterator IN (
  --- Fetches the values that we calculate a total score from ---
            select oraclescore, verityscore,id
            from CBIRGPS_TEMP
  order by v_totalscore
  )
  LOOP
  v_totalscore:=(((1-(iterator.verityscore))*100) + (iterator.oraclescore)) / 2;
  UPDATE CBIRGPS_TEMP set totalscore=v_totalscore where id=iterator.id;
  END LOOP;
END PROCESS_CBIRGPS_SCORES;
```

5.1 Flow chart



6.0 CBIR/GPS Search

Lars-Jacob uttered a request to combine CBIR search and GPS search into a unison search to improve precision/recall. Furthermore, another useful feature with respect to the combination of CBIR/GPS search, could be filtering of the images found with CBIR based on a set of GPS coordinates and the distance parameter. The latter has not been implemented, but this functionality has been added to the future development section.

The CBIR/GPS search was implemented by using a temporary table to hold the result data, as well as several PL/SQL procedures. First of all a modified version of the original CBIR procedure in VISI is used. The procedure carries out a normal CBIR search, but also inserts the ids of the images and their respective scores into a temporary table. Furthermore a GPS search based on exact coordinates is

performed against the Verity index. However it is unlikely that such a search will produce any actual results, nevertheless if matching results are found, they are also inserted into the temporary table. Do note that one can input wild cards into the GPS coordinates, however, this will cause the search described below to fail as wild cards are not valid input to Oracle in this contex.

Finally, a third search option is utilized. A set of GPS coordinates are sent to a procedure which in turns finds all images within a given proximity. This is basically the same procedure as described in 4.0

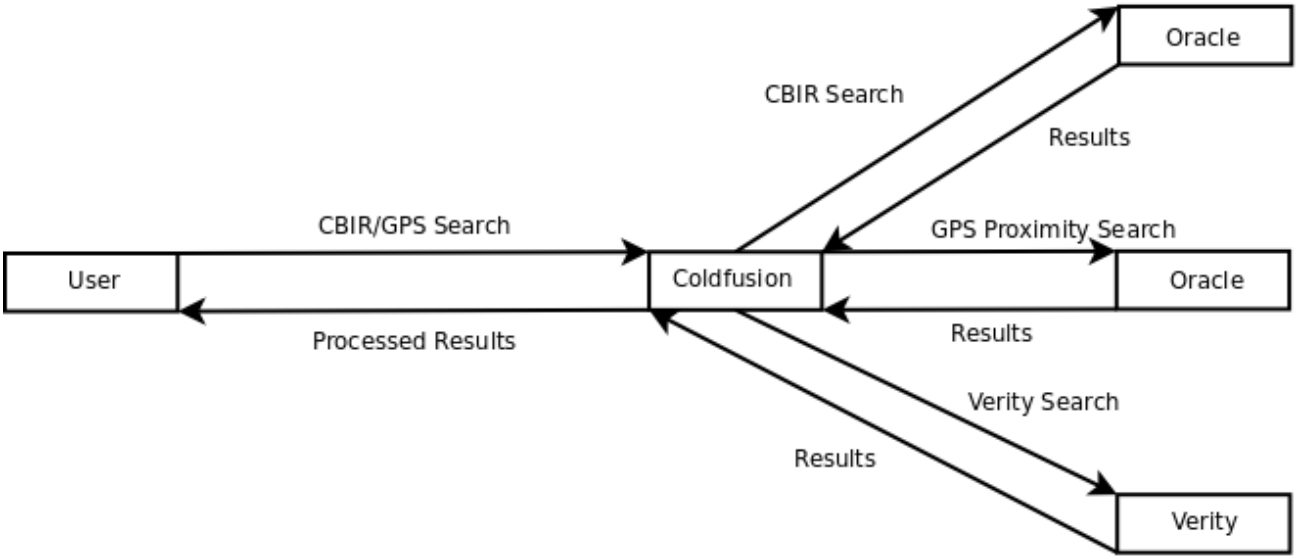
```
/* Some code has been left out, please see Appendix C for further reference */
  BEGIN
      FOR iterator IN (
       SELECT
        o.ID,
        ROUND(
          SDO_GEOM.SDO_DISTANCE(
            MDSYS.SDO_GEOMETRY(
              2001,
              8307,
              mdsys.sdo_point_type(
                p_Latitude,
                p_Longitude,
                NULL),
              NULL,
              NULL),
            o.Coordinates,
            20,
            'unit=M'),
          2) DISTANCE
      FROM image o
      ORDER BY DISTANCE ASC
      )

      LOOP
        IF(iterator.DISTANCE < p_Distance)
        THEN
          IF (p_Results IS NULL) THEN p_Results := iterator.ID;
         ELSE p_Results := p_Results ||','|| iterator.ID;
          END IF;
        END IF;
      END LOOP;
END CBIRGPS_PROXIMITY_SEARCH;
```

The matching image ids are returned to Coldfusion for further processing. This was done due to problems with the exception handling in PL/SQL. As far as the author understood it a cursor's scope is not available within an exception block, thus making it very hard to perform the insert/update table statements efficiently. The actual updating of the temporary table is done with this piece of code:

```
<!--- Attempts to find more pictures within the given perimeter --->

<cfstoredproc datasource="#Application.db_odbc#"
procedure="VISI_BERGENBY.CBIRGPS_PROXIMITY_SEARCH">
       <cfprocparam type="in" value="#Form.latitude#" cfsqltype="cf_sql_number">
       <cfprocparam type="in" value="#Form.longitude#" cfsqltype="cf_sql_number">
       <cfprocparam type="in" value="#Form.distance#" cfsqltype="cf_sql_number">
       <cfprocparam type="out" variable="images"         cfsqltype="cf_sql_varchar">
</cfstoredproc>

<!--- Fetches and processes the output from the CBIRGPS_IMAGE_SEARCG --->
<cfset gpsareasearch = ListToArray(#images#, ",")>

<cfdump var="#gpsareasearch#">

<cfloop index=Counter from=1 to="#ArrayLen(gpsareasearch)#">
       <cftry>
                     <cfquery name="update" datasource="#Application.db_odbc#">
                            INSERT INTO CBIRGPS_TEMP
VALUES(#gpsareasearch[Counter]#,0,#score#,0,0)
                     </cfquery>
              <cfcatch type="Database">
                     <cfquery name="update2" datasource="#Application.db_odbc#">
                            UPDATE CBIRGPS_TEMP SET verityscore=#score# WHERE
id=#gpsareasearch[Counter]#
                     </cfquery>
              </cfcatch>
       </cftry>
</cfloop>
```

Since the proximity search in Oracle does not generate a score, we use the score generated by Verity instead. However it may be the case that it is not possible to find a match in the Verity index, thus score is assigned a value of 0 if this is not the case.

6.1 Flow Chart



7.0 Future Development

As for future development it could be wise to refactor the current code base further. The author has not had much time to do code optimization. The author also considers it a problem that VISI2 is bound to a piece of proprietary software from Adobe. It could perhaps be wise to consider a FLOSS (Free/Libre Open Source Software) solution instead.

Another issue that should be looked into is indexing of the texts/keywords associated with objects and images. The texts and keywords are stored in nested tables, thus one row per text/keyword appears to be sent to Verity, this appears to leave an incomplete index, that is, not all keywords are available. A solution could be to put all the keywords for one object in a column, however implementing this would require changes that would be too time consuming now at the end of the project. A different solution could perhaps be to keep an unique index for both texts and keywords.

Furthermore it could be useful to fetch the GPS coordinates of a picture once a user decides to use this as a seed image, the respective coordinates in the “GPS Search” section, as well as the “CBIR GPS” section could then be updated.

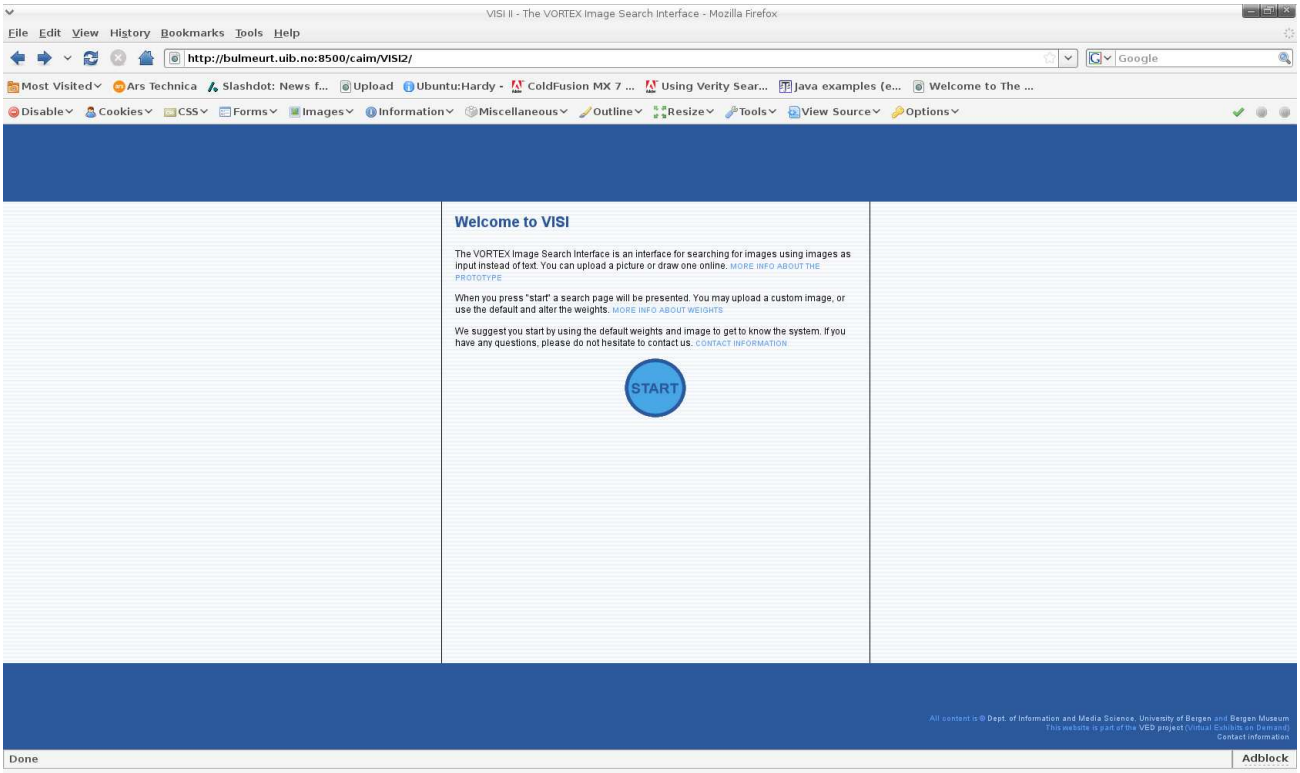
It could also be very interesting to experiment with the weighting used in the various PL/SQL procedures, and observe how this affects the result sets with regard to precision/recall.

Finally, another useful feature with respect to the combination of CBIR/GPS search could be filtering of the images found with CBIR based on a set of GPS coordinates and the distance parameter.

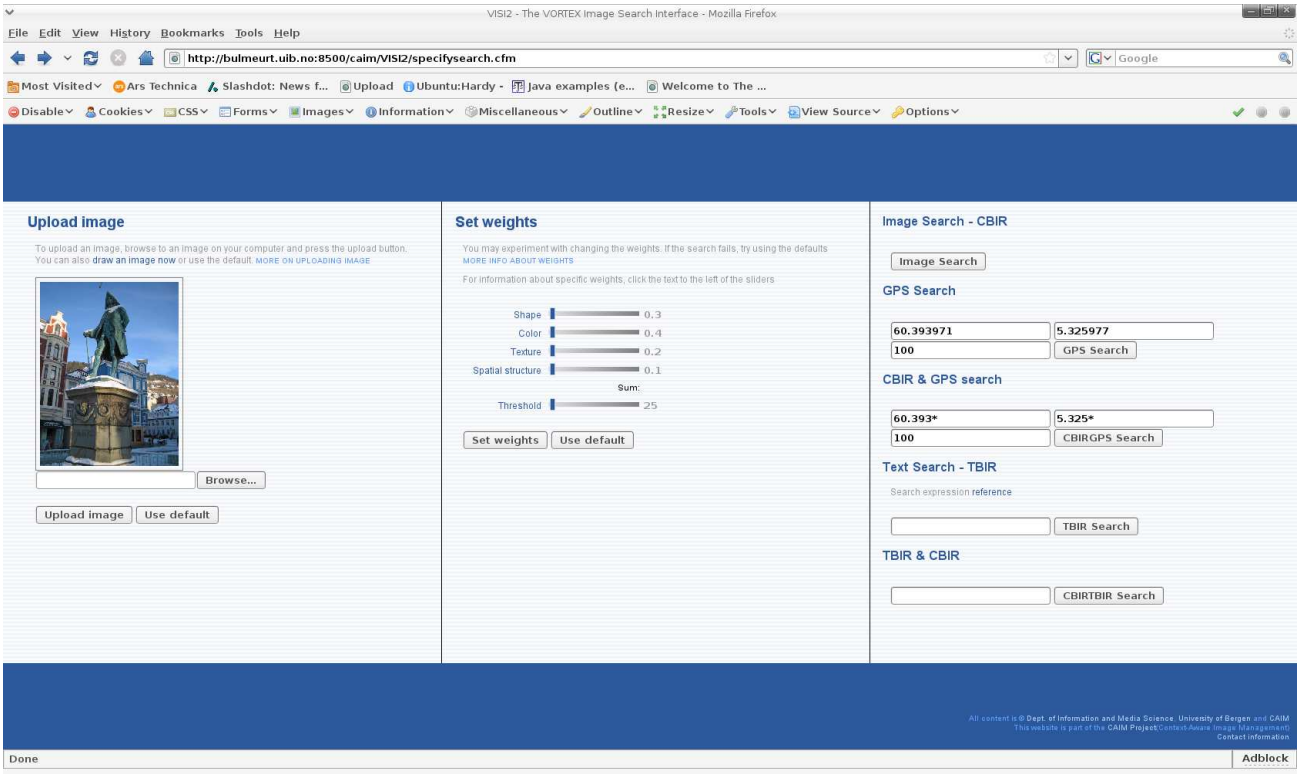
8.0 User Manual

To use the VISI2 prototype simply visit http://bulmeurt.uib.no:8500/caim/VISI2

As a normal user this is the first page that you will see.



To reach the search interface simply click on the “Start” button.



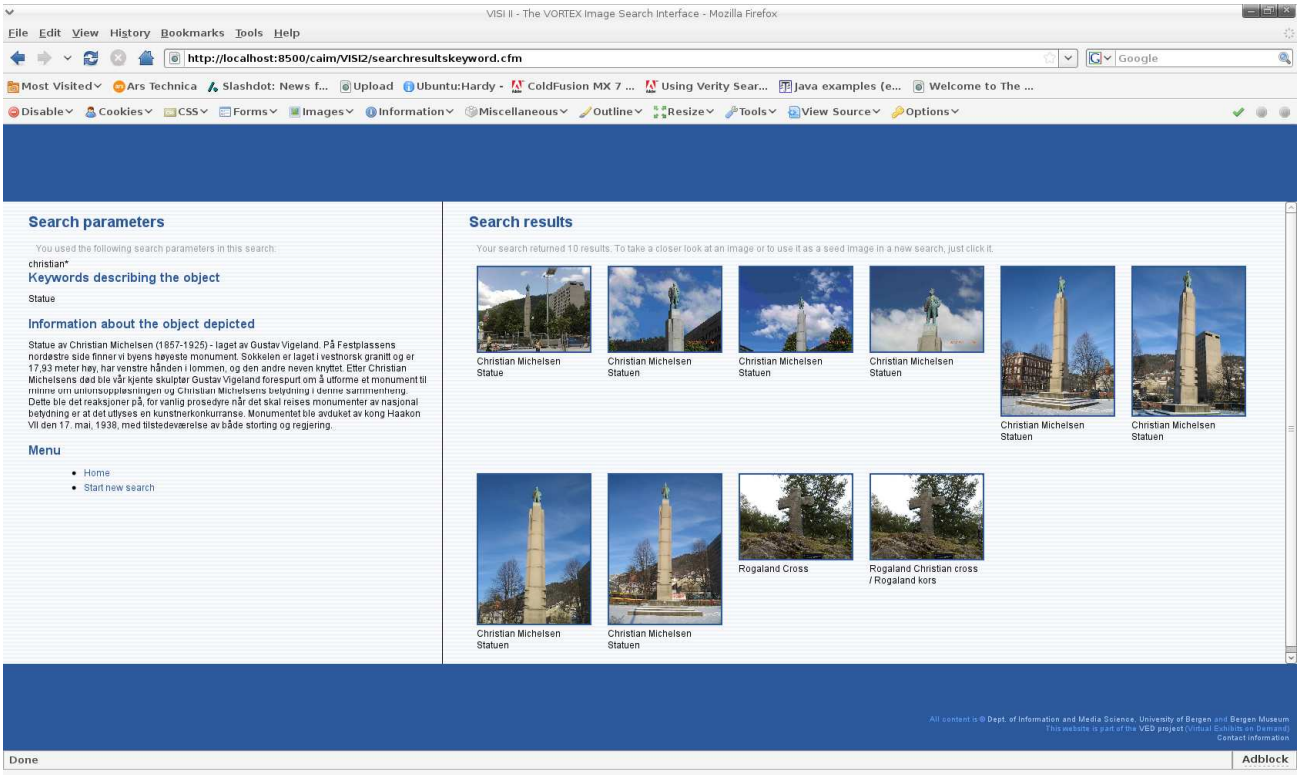
Functionality for GPS search has been implemented in VISI2. The example coordinates as shown above match the default picture ('Børsbygningen'). All objects as well as all images in the database should be stored with a set of GPS coordinates. The Verity index only keeps track of the exact GPS coordinates of all objects. However it is unlikely that a user would send exact coordinates as input to the search interface. Therefore VISI 2 makes use of Oracle's functionality to search for other sets of coordinates within a given proximity. The bottom value of “100” specifies this proximity, given in meters. It is also possible to provide wild cards to the set of coordinates. This will make the Oracle part of the search fail, however it is possible to find matches in the Verity GPS index.

A search method featuring CBIR(Content Based Image Retrieval) and GPS search combined is also available. The GPS part works exactly as described above. With respect to the CBIR part of the search, weights as depicted in the middle of the interface can be used, as well as a user specified seed image.

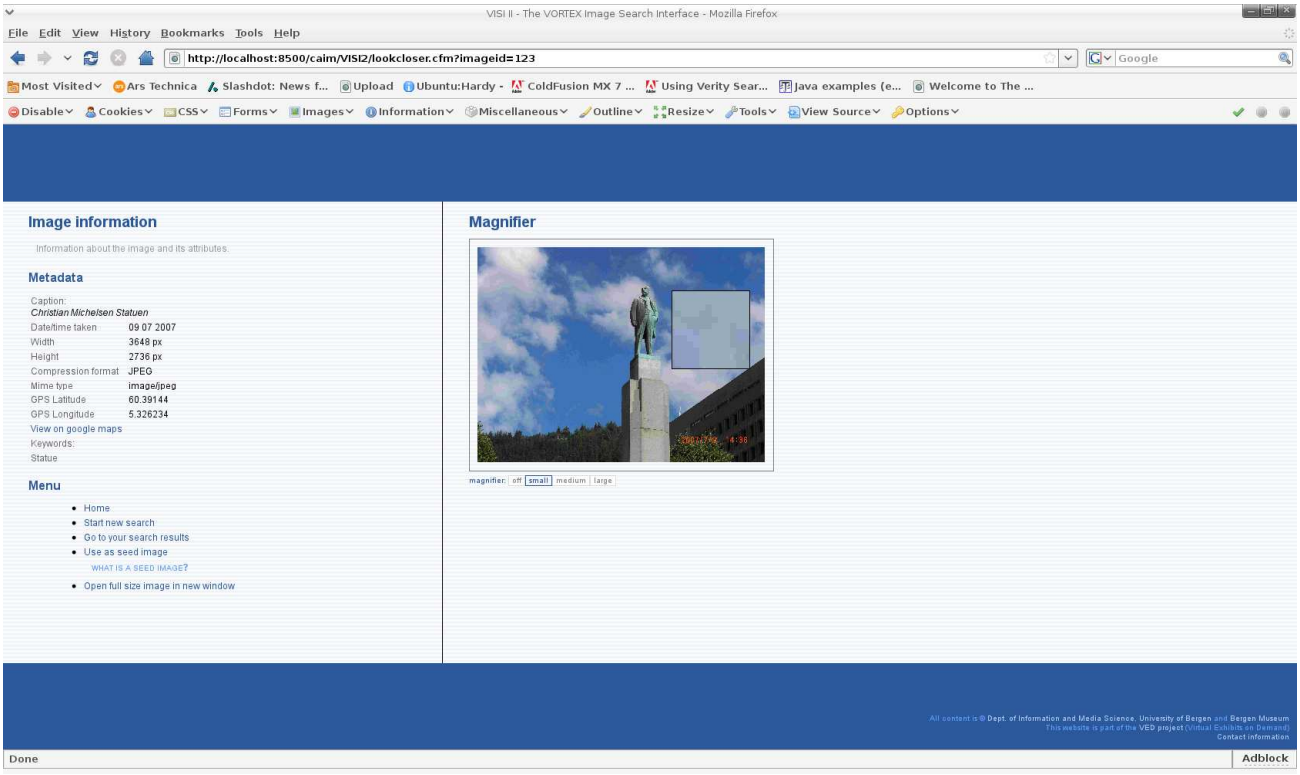
VISI 2 is working together with a full text search engine called Verity to be able to offer TBIR (Text Based Image Retrieval). However there are rules as to how a user should express searches. For example to search for pictures of 'Domkirken' or 'Johanneskirken', one could simply input the string “Domkirken,Johanneskirken” to the keyword search field. An URL to the search expression reference

to Verity is provided in the interface.

It is also possible to use TBIR and CBIR together. A user may input keywords to the search field on the left side of the interface in Verity syntax. Furthermore weights can be specified. However, please do note that the sliders may not work properly in all browsers. The author did not have enough time to look into this thoroughly. The user may choose to upload a new picture, use a picture from a result set fetched in a previous search. There is also a default picture available.



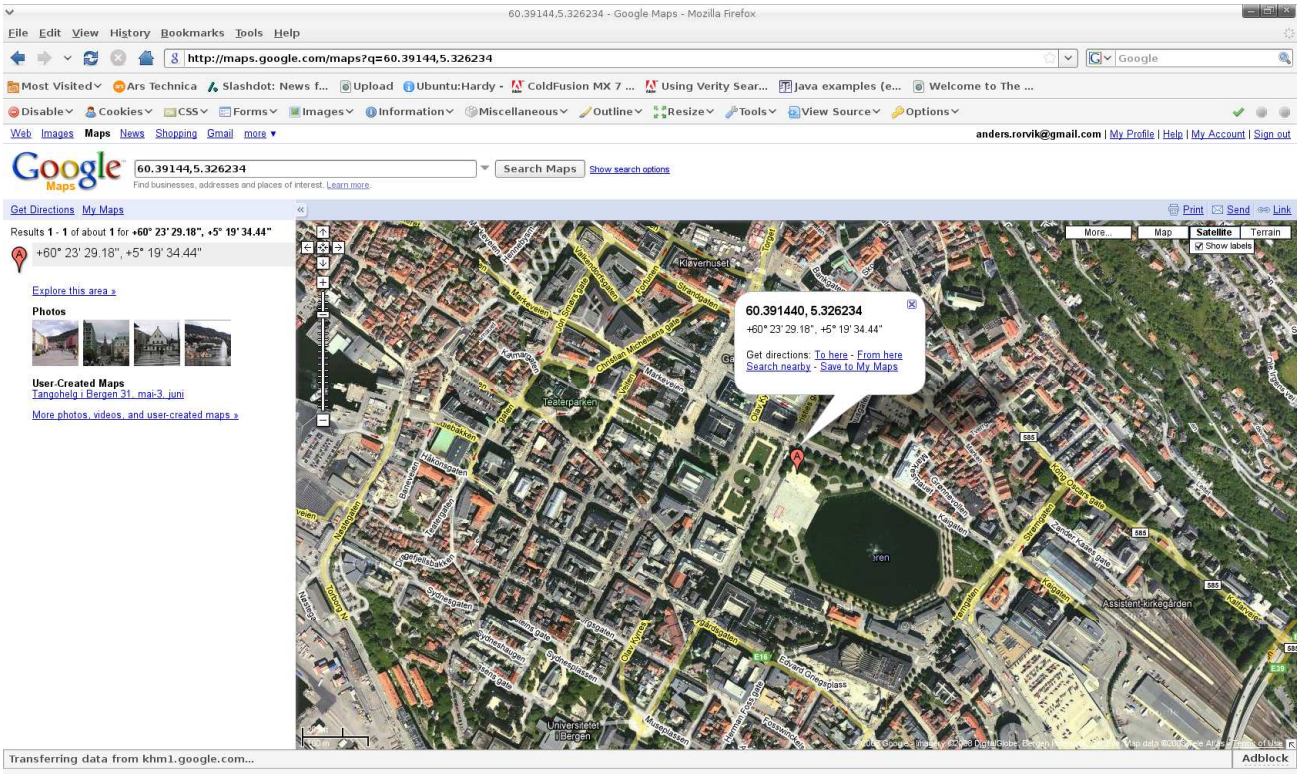
The box on the left side of the picture contains various metadata about the primary object found by the search. The primary object is selected based on the image id with the best score. This is done due to space considerations. The content on the right side of the interface is as one can see a set of images in a grid. The images are described by their respective captions, and upon clicking on an image the following interface will be presented to the user:



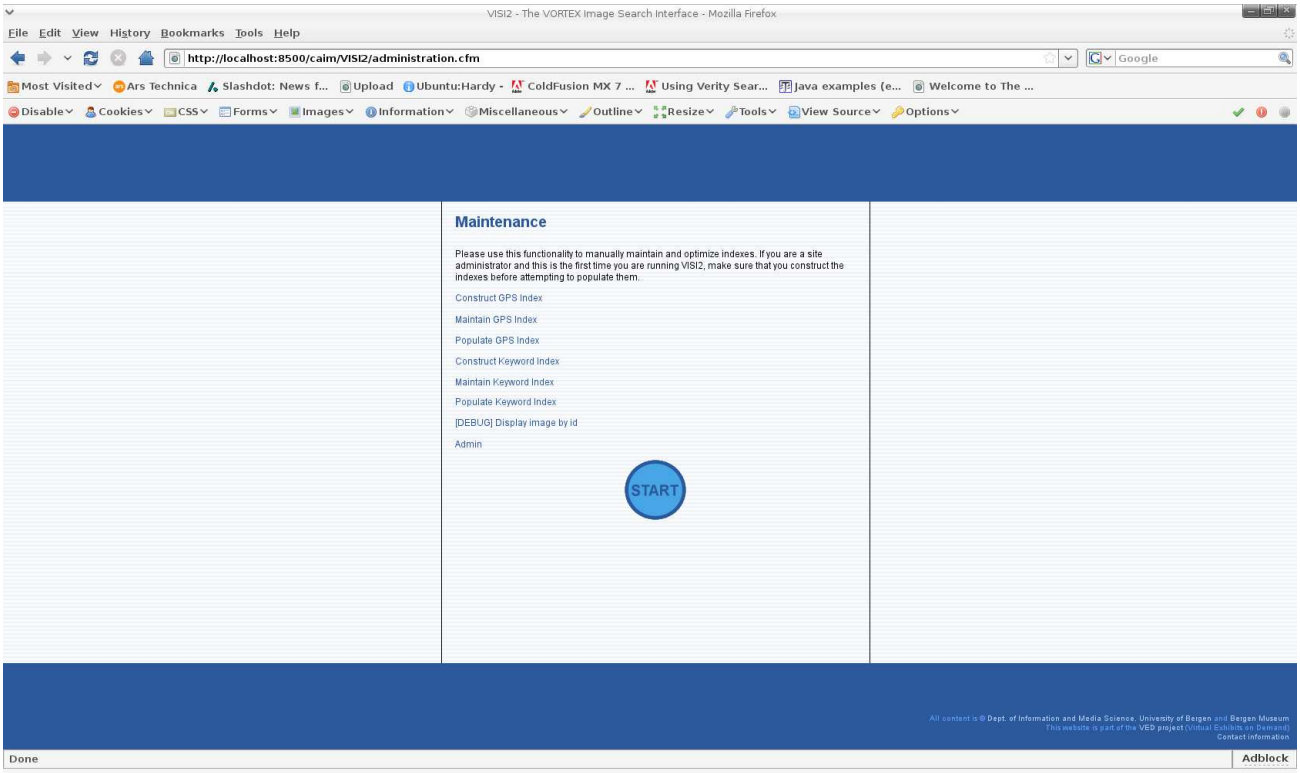
Again, the left box contains various metadata concerning the particular photo. The square on the right side of the image is a magnifying tool that can be made smaller or larger. If desired it can also be turned off. If the user wishes so, it is also possible to view the full size image. Simply click on the designated link in the menu.

The image the user is viewing can also be used as a “seed image” instead of the default one. This can be achieved by clicking the link “use as seed image”.

It is also possible for a user to view the location of where the image was taken on Google maps. Simply click on the “View on google maps” button, and the location of where the image was taken should be pin pointed on the map. This functionality is depicted below.


60.39144,5.326234 - Google Maps - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://maps.google.com/maps?q=60.39144,5.326234
Most Visited Ars Technica Slashdot: News f... Upload Ubuntu:Hardy - ColdFusion MX 7 ... Using Verity Sear... Java examples (e... Welcome to The ...
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Web Images Maps News Shopping Gmail more
anders.rorvik@gmail.com | My Profile | Help | My Account | Sign out
Google Maps
60.39144,5.326234
Search Maps
Show search options
Find businesses, addresses and places of interest. Learn more.
Get Directions My Maps
Results 1 - 1 of about 1 for +60° 23' 29.18", +5° 19' 34.44"
+60° 23' 29.18", +5° 19' 34.44"
Explore this area »
Photos
User-Created Maps
Tangohelg i Bergen 31. mai-3. juni
More photos, videos, and user-created maps »
Print Send Link
More... Map Satellite Terrain
Show labels
60.391440, 5.326234
+60° 23' 29.18", +5° 19' 34.44"
Get directions: To here - From here
Search nearby - Save to My Maps
Klaverhuset
Teaterparken
Universitetet i Bergen
Assistent-kirkegården
Transferring data from khm1.google.com...
Adblock

9.0 VISI2 Administration Interface



The maintenance page can be reached at the following URL:
http://bulmeurt.uib.no:8500/caim/VISI2/administration.cfm

Here it is possible to maintain/populate as well as optimize the indexes currently being used by VISI2. The indexes are automatically updated once a day, but sometimes it might be desirable to update them manually. Simply press the “populate button” for this action to occur. The administration interface also contains an URL pointing to the administrative interface of the Coldfusion server. A debug feature used in the development process “display image by id” is also available.

Appendix A – Readme

The intention of this section is to clarify where the source code for VISI2 can be found. The main development took place on a "private" Coldfusion server running on the author's laptop. However the author has written a shell script to sync VISI2 to Bulmeurt per wish. If you are running GNU/Linux or a different *NIX based operating system, and have SCP installed, the following script is very simple, and useful. The author is aware of plugins that can save files via ssh embedded in editors like Eclipse, however there were numerous problems with these solutions which will not be discussed further here.

```
#!/bin/bash
scp -r /opt/coldfusion8/wwwroot/caim/VISI2
username@bulmeurt.uib.no:/opt/coldfusionmx7/wwwroot/caim
```

Save this script in your user's bin directory. The file name does not matter. However, make sure that the script above is put in the file as a single line, or problems will occur. The path to the source code on Bulmeurt is /opt/coldfusionmx7/wwwroot/caim , and the directory where VISI2 resides is called VISI2, thus making the absolute path /opt/coldfusionmx7/wwwroot/caim /VISI2.

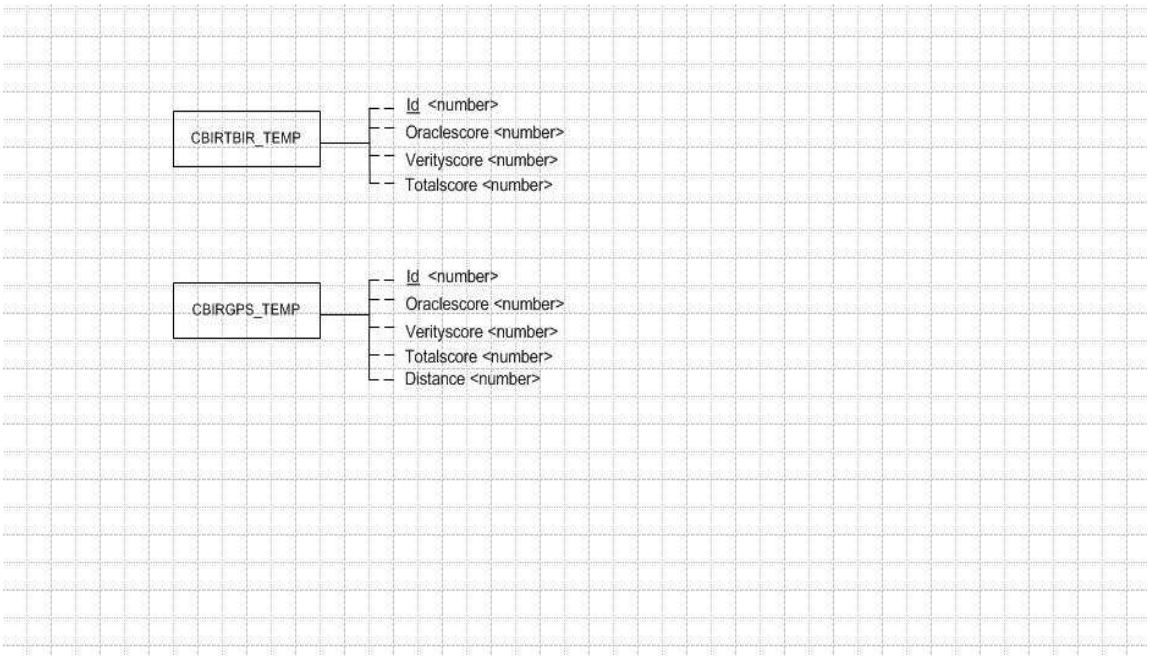
Appendix B – Program code

The source code has not been included in the document due to space considerations The source code for VISI2 is available on Bulmeurt[3] at this location: "/opt/coldfusionmx7/wwwroot/caim /VISI2". It is also available on the CD distributed with the final report.

3 Bulmeurt is a GNU/Linux server currently used by CAIM

Appendix C – DB data

Some of the functionality in VISI2 relies on two "temporary" tables used to gather results and process results from various searches. The tables above are modeled in SSM.



The VISI 2 prototype also makes use of an Oracle package containing various procedures and functions. The main difference between procedures and functions in Oracle is that the former can return several values.

The VISI_BERGENBY package signature can be observed below:

```
CREATE OR REPLACE PACKAGE         "VISI_BERGENBY"  AS

  PROCEDURE SEARCH (
    p_Url          IN VARCHAR2,
    p_Filename      IN VARCHAR2,
    p_Shape         IN VARCHAR2,
    p_Color         IN VARCHAR2,
    p_Texture       IN VARCHAR2,
    p_Spatial       IN VARCHAR2,
    p_Threshold     IN VARCHAR2,
    p_Results       OUT NOCOPY VARCHAR2
  );

  PROCEDURE CBIR_TBIR (
```

```
  p_Url IN VARCHAR2,
  p_Filename IN VARCHAR2,
  p_Shape IN VARCHAR2,
  p_Color IN VARCHAR2,
  p_Texture IN VARCHAR2,
  p_Spatial IN VARCHAR2,
  p_Threshold IN VARCHAR2,
  p_Results OUT NOCOPY VARCHAR2
 );

 PROCEDURE CBIR_GPS (
  p_Url IN VARCHAR2,
  p_Filename IN VARCHAR2,
  p_Shape IN VARCHAR2,
  p_Color IN VARCHAR2,
  p_Texture IN VARCHAR2,
  p_Spatial IN VARCHAR2,
  p_Threshold IN VARCHAR2,
  p_Results OUT NOCOPY VARCHAR2
 );

PROCEDURE PROCESS_CBIRTBIR_SCORES;

PROCEDURE PROCESS_CBIRGPS_SCORES;

 PROCEDURE CBIR_GPS_SEARCH(
  p_Latitude       IN NUMBER,
  p_Longitude    IN NUMBER,
  p_Distance      IN NUMBER,
  p_Verityscore IN NUMBER
);


FUNCTION IMAGE_SEARCH(
  p_SeedId    NUMBER,
  p_Weights   VARCHAR2,
  p_Threshold NUMBER
) RETURN VARCHAR2;

FUNCTION CBIR_TBIR_IMAGE_SEARCH(
  p_SeedId    NUMBER,
  p_Weights   VARCHAR2,
  p_Threshold NUMBER
) RETURN VARCHAR2;

 FUNCTION CBIR_GPS_IMAGE_SEARCH(
  p_SeedId    NUMBER,
  p_Weights   VARCHAR2,
  p_Threshold NUMBER
) RETURN VARCHAR2;

FUNCTION IMPORT_SEED(
  p_Url          IN VARCHAR2,
  p_Filename      IN VARCHAR2,
  p_SeedImageId   IN NUMBER
```

```
  ) RETURN BOOLEAN;


  PROCEDURE GPS_IMAGE_SEARCH(
    p_Latitude       IN NUMBER,
    p_Longitude    IN NUMBER,
    p_Distance       IN NUMBER,
    p_Verityscore IN NUMBER
  );

  PROCEDURE CBIRGPS_PROXIMITY_SEARCH(
    p_Latitude       IN NUMBER,
    p_Longitude    IN NUMBER,
    p_Distance       IN NUMBER,
    p_Results    OUT NOCOPY VARCHAR2

  );

END VISI_BERGENBY;
```

The actual package body can be viewed here:

```
CREATE OR REPLACE PACKAGE BODY VISI_BERGENBY AS

  PROCEDURE SEARCH (
     p_Url           IN VARCHAR2,
     p_Filename      IN VARCHAR2,
     p_Shape         IN VARCHAR2,
     p_Color         IN VARCHAR2,
     p_Texture       IN VARCHAR2,
     p_Spatial       IN VARCHAR2,
     p_Threshold     IN VARCHAR2,
     p_Results       OUT NOCOPY VARCHAR2
   )
   AS
     v_SeedImageId       NUMBER := 1;
     v_Weights           VARCHAR2(1000);
     e_NoResults         EXCEPTION;
     e_SeedImportFail    EXCEPTION;

   BEGIN

     -- Importing seed image

     IF NOT IMPORT_SEED(p_Url, p_Filename, v_SeedImageId) THEN
       RAISE e_SeedImportFail;
     END IF;

     v_Weights := 'color = "'||p_Color||'", texture = "'||p_Texture||'", shape = "'||p_Texture||'", location = "'||p_Spatial||'"';

     p_Results := IMAGE_SEARCH(v_SeedImageId, v_Weights, p_Threshold);

   EXCEPTION
```

```
    WHEN e_SeedImportFail THEN
      p_Results := '-2';
    WHEN OTHERS THEN
      p_Results := '-3';

 END SEARCH;

 PROCEDURE CBIR_TBIR (
          p_Url IN VARCHAR2,
          p_Filename IN VARCHAR2,
          p_Shape IN VARCHAR2,
          p_Color IN VARCHAR2,
          p_Texture IN VARCHAR2,
          p_Spatial IN VARCHAR2,
          p_Threshold IN VARCHAR2,
          p_Results OUT NOCOPY VARCHAR2
          )

          AS
          v_SeedImageId NUMBER :=1;
          v_Weights VARCHAR2(1000);
          e_NoResults EXCEPTION;
          e_SeedImportFail EXCEPTION;

          BEGIN
                    -- Importing seed image
                    IF NOT IMPORT_SEED(p_Url, p_Filename, v_SeedImageId) THEN
                    RAISE e_SeedImportFail;
                END IF;

                    v_Weights := 'color = "'||p_Color||'", texture = "'||p_Texture||'", shape = "'||p_Texture||'", location =
"'||p_Spatial||'"';

                    p_Results := CBIR_TBIR_IMAGE_SEARCH(v_SeedImageId, v_Weights, p_Threshold);

                    EXCEPTION
                      WHEN e_SeedImportFail THEN
                       p_Results := '-2';
                      WHEN OTHERS THEN
                       p_Results :='-3';
END CBIR_TBIR;


PROCEDURE CBIR_GPS (
          p_Url IN VARCHAR2,
          p_Filename IN VARCHAR2,
          p_Shape IN VARCHAR2,
          p_Color IN VARCHAR2,
          p_Texture IN VARCHAR2,
          p_Spatial IN VARCHAR2,
          p_Threshold IN VARCHAR2,
          p_Results OUT NOCOPY VARCHAR2
          )

          AS
```

```
        v_SeedImageId NUMBER :=1;
        v_Weights VARCHAR2(1000);
        e_NoResults EXCEPTION;
        e_SeedImportFail EXCEPTION;

        BEGIN
                -- Importing seed image
                IF NOT IMPORT_SEED(p_Url, p_Filename, v_SeedImageId) THEN
                RAISE e_SeedImportFail;
            END IF;

                v_Weights := 'color = "'||p_Color||'", texture = "'||p_Texture||'", shape = "'||p_Texture||'", location =
"'||p_Spatial||'"';

                p_Results := CBIR_GPS_IMAGE_SEARCH(v_SeedImageId, v_Weights, p_Threshold);

                EXCEPTION
                  WHEN e_SeedImportFail THEN
                   p_Results := '-2';
                  WHEN OTHERS THEN
                   p_Results :='-3';
END CBIR_GPS;



PROCEDURE PROCESS_CBIRTBIR_SCORES
 AS
 v_totalscore NUMBER:=0;

 BEGIN

 for iterator IN (
 --- Fetches the values that we calculate a total score from ---
        select oraclescore, verityscore,id
        from CBIRTBIR_TEMP
 order by v_totalscore
 )
 LOOP
 v_totalscore:=(((1-(iterator.verityscore))*100) + (iterator.oraclescore)) / 2;
 UPDATE CBIRTBIR_temp set totalscore=v_totalscore where id=iterator.id;
 END LOOP;
END PROCESS_CBIRTBIR_SCORES;


PROCEDURE PROCESS_CBIRGPS_SCORES
 AS
 v_totalscore NUMBER:=0;

 BEGIN

 for iterator IN (
 --- Fetches the values that we calculate a total score from ---
        select oraclescore, verityscore,id
        from CBIRGPS_TEMP
 order by v_totalscore
```

```
  )
  LOOP
  v_totalscore:=(((1-(iterator.verityscore))*100) + (iterator.oraclescore)) / 2;
  UPDATE CBIRGPS_TEMP set totalscore=v_totalscore where id=iterator.id;
  END LOOP;
END PROCESS_CBIRGPS_SCORES;


END PROCESS_CBIRGPS_SCORES;


PROCEDURE CBIR_GPS_SEARCH(
    p_Latitude       IN NUMBER,
    p_Longitude    IN NUMBER,
    p_Distance       IN NUMBER,
    p_Verityscore IN NUMBER
  )

  IS
  e_queryError EXCEPTION;

  BEGIN
       FOR iterator IN (
        SELECT
         o.ID,
         ROUND(
           SDO_GEOM.SDO_DISTANCE(
             MDSYS.SDO_GEOMETRY(
               2001,
               8307,
               mdsys.sdo_point_type(
                 p_Latitude,
                 p_Longitude,
                 NULL),
               NULL,
               NULL),
             o.Coordinates,
             20,
             'unit=M'),
           2) DISTANCE
       FROM image o
       ORDER BY DISTANCE ASC
       )

       LOOP
         IF(iterator.DISTANCE < p_Distance)
         THEN
           INSERT INTO CBIRGPS_TEMP VALUES(iterator.ID,0,p_Verityscore,iterator.DISTANCE,0);
         END IF;
       END LOOP;
END CBIR_GPS_SEARCH;

  FUNCTION IMAGE_SEARCH(
     p_SeedId    NUMBER,
     p_Weights   VARCHAR2,
```

```
    p_Threshold NUMBER
  ) RETURN VARCHAR2
     IS
       v_SearchimageSig    ORDSYS.ORDImageSignature; -- SÃ¸kebildets signatur
       v_ReturnString      VARCHAR2(1000);
   BEGIN

     --Henter ut den bildesignaturen som man skal sammenligne med
     SELECT i.imageSignature
       INTO v_SearchimageSig
       FROM IMAGE i WHERE i.id=p_SeedId;

     FOR iterator IN (
       SELECT i.ID, ORDSYS.IMGScore(1) Score FROM IMAGE i WHERE
            ORDSYS.IMGSimilar(
              i.ImageSignature,
              v_SearchimageSig,
              p_Weights,
              p_Threshold,
              1
            ) = 1
         AND i.id != p_SeedId
         ORDER BY Score
       )
     LOOP
        IF (v_ReturnString IS NULL) THEN v_ReturnString := iterator.ID;
        ELSE v_ReturnString := v_ReturnString ||','|| iterator.ID;
        END IF;
     END LOOP;
     RETURN v_ReturnString;
END;


 FUNCTION CBIR_TBIR_IMAGE_SEARCH(
   p_SeedId    NUMBER,
   p_Weights   VARCHAR2,
   p_Threshold NUMBER
 ) RETURN VARCHAR2
     IS
       v_SearchimageSig    ORDSYS.ORDImageSignature; -- SÃ¸kebildets signatur
       v_ReturnString      VARCHAR2(1000);
   BEGIN

     --Henter ut den bildesignaturen som man skal sammenligne med
     SELECT i.imageSignature
       INTO v_SearchimageSig
       FROM IMAGE i WHERE i.id=p_SeedId;

     FOR iterator IN (
       SELECT i.ID, ORDSYS.IMGScore(1) Score FROM IMAGE i WHERE
            ORDSYS.IMGSimilar(
              i.ImageSignature,
              v_SearchimageSig,
              p_Weights,
              p_Threshold,
```

```
            1
          ) = 1
        AND i.id != p_SeedId
        ORDER BY Score
      )
    LOOP
      INSERT INTO CBIRTBIR_TEMP VALUES(iterator.ID,iterator.Score,0,0);
      IF (v_ReturnString IS NULL) THEN v_ReturnString := iterator.ID;
      ELSE v_ReturnString := v_ReturnString ||','|| iterator.ID;
      END IF;
    END LOOP;
    RETURN v_ReturnString;
END;

FUNCTION CBIR_GPS_IMAGE_SEARCH(
  p_SeedId    NUMBER,
  p_Weights   VARCHAR2,
  p_Threshold NUMBER
 ) RETURN VARCHAR2
    IS
      v_SearchimageSig    ORDSYS.ORDImageSignature; -- SÃ¸kebildets signatur
      v_ReturnString      VARCHAR2(1000);
  BEGIN

    --Henter ut den bildesignaturen som man skal sammenligne med
    SELECT i.imageSignature
     INTO v_SearchimageSig
     FROM IMAGE i WHERE i.id=p_SeedId;

    FOR iterator IN (
     SELECT i.ID, ORDSYS.IMGScore(1) Score FROM IMAGE i WHERE
          ORDSYS.IMGSimilar(
            i.ImageSignature,
            v_SearchimageSig,
            p_Weights,
            p_Threshold,
            1
          ) = 1
        AND i.id != p_SeedId
        ORDER BY Score
      )
    LOOP
      INSERT INTO CBIRGPS_TEMP VALUES(iterator.ID,iterator.Score,0,0,0);
      IF (v_ReturnString IS NULL) THEN v_ReturnString := iterator.ID;
      ELSE v_ReturnString := v_ReturnString ||','|| iterator.ID;
      END IF;
    END LOOP;
    RETURN v_ReturnString;
END;

FUNCTION IMPORT_SEED(
 p_Url          IN VARCHAR2,
 p_Filename     IN VARCHAR2,
 p_SeedImageId  IN NUMBER
 )
```

```
  RETURN        BOOLEAN
IS
  v_Image ordsys.ordimage;
  v_ImageSig ordsys.ordimagesignature;
  v_CTX RAW(4000) := null;

  v_MetaData    XMLSequenceType;
  v_MetaDataRoot VARCHAR2(40);
BEGIN

  DELETE FROM IMAGE WHERE ID = p_SeedImageId;

  INSERT INTO IMAGE VALUES(
                    p_SeedImageId,
                    null,
                    null,
                    null,
                    ordsys.ordimage.init(),
                    ordsys.ordimagesignature.init(),
                    null,
                    null,
                    null,
                    null,
                    null
                    );

  -- Locking table for insert
  DBMS_OUTPUT.put_line('Locking table for insert');

  SELECT Image, imageSignature INTO v_Image, v_ImageSig
    FROM IMAGE WHERE ID = p_SeedImageId FOR UPDATE;

  -- Import the image
  DBMS_OUTPUT.put_line('Importing the image: ' || p_URL || '/' || p_Filename);

  v_Image.setsource('HTTP', p_URL, p_Filename);
  v_Image.import(v_CTX);
  v_ImageSig.generatesignature(v_Image);
  v_Image.setproperties;

  -- Update the image table
  UPDATE IMAGE SET Image = v_Image, imageSignature = v_ImageSig WHERE ID = p_SeedImageId;

  DBMS_OUTPUT.put_line('Image inserted');

  RETURN TRUE;

  EXCEPTION
       WHEN OTHERS THEN
         dbms_output.put_line('Error inserting image: (' || SQLERRM || ').');

         RETURN FALSE;
END IMPORT_SEED;

PROCEDURE GPS_IMAGE_SEARCH(
```

```
   p_Latitude       IN NUMBER,
   p_Longitude    IN NUMBER,
   p_Distance      IN NUMBER,
   p_Verityscore IN NUMBER
 )

 IS
 e_queryError EXCEPTION;

 BEGIN
      FOR iterator IN (
       SELECT
        o.ID,
        ROUND(
          SDO_GEOM.SDO_DISTANCE(
            MDSYS.SDO_GEOMETRY(
              2001,
              8307,
              mdsys.sdo_point_type(
                p_Latitude,
                p_Longitude,
                NULL),
              NULL,
              NULL),
            o.Coordinates,
            20,
            'unit=M'),
          2) DISTANCE
      FROM image o
      ORDER BY DISTANCE ASC
      )

      LOOP
       IF(iterator.DISTANCE < p_Distance)
       THEN
         INSERT INTO CBIRGPS_TEMP VALUES(iterator.ID,0,0,iterator.DISTANCE,0);
       END IF;
      END LOOP;
END GPS_IMAGE_SEARCH;

PROCEDURE CBIRGPS_PROXIMITY_SEARCH(
   p_Latitude       IN NUMBER,
   p_Longitude    IN NUMBER,
   p_Distance      IN NUMBER,
   p_Results   OUT NOCOPY VARCHAR2

 )
  IS
 e_queryError EXCEPTION;

 BEGIN
      FOR iterator IN (
       SELECT
        o.ID,
        ROUND(
```

```
          SDO_GEOM.SDO_DISTANCE(
            MDSYS.SDO_GEOMETRY(
              2001,
              8307,
              mdsys.sdo_point_type(
                p_Latitude,
                p_Longitude,
                NULL),
              NULL,
              NULL),
            o.Coordinates,
            20,
            'unit=M'),
          2) DISTANCE
        FROM image o
        ORDER BY DISTANCE ASC
        )

        LOOP
          IF(iterator.DISTANCE < p_Distance)
          THEN
            IF (p_Results IS NULL) THEN p_Results := iterator.ID;
           ELSE p_Results := p_Results ||','|| iterator.ID;
            END IF;
          END IF;
        END LOOP;
END CBIRGPS_PROXIMITY_SEARCH;


END VISI_BERGENBY;
```

The DDL for the two temporary tables can be found below:

```
CREATE TABLE CBIRTBIR_TEMP(
id NUMBER,
oraclescore NUMBER,
verityscore NUMBER,
totalscore NUMBER,
primary key(id)
);

CREATE TABLE CBIRGPS_TEMP (
id NUMBER,
oraclescore NUMBER,
verityscore NUMBER,
distance NUMBER,
totalscore NUMBER,
primary key(id)
);
```