

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link.

<http://hdl.handle.net/2066/32615>

Please be advised that this information was generated on 2020-09-09 and may be subject to change.

Implementing Vimes - the broker component.

Eric D. Schabell
erics@cs.ru.nl

Bas van Gils
basvg@cs.ru.nl

University of Nijmegen, Computing Science Institute, P.O. Box
9010, 6500 GL Nijmegen, The Netherlands

Abstract

This document will discuss the *Vimes* retrieval architecture broker component from the research project *Profile Based Retrieval Of Networked Information Resources (PRONIR)*. It will provide an overview of the development process from requirements investigations done with use cases, on to the actual design and implementation.

1 Introduction

This document will present a structured look at the *Vimes* retrieval architecture broker development project for the research project *Profile Based Retrieval Of Networked Information Resources (PRONIR)*.

The information retrieval architecture called *Vimes* was briefly described in (Gils et al., 2003b). To facilitate experimentation and validation within the *PRONIR* project the *Vimes* retrieval architecture will have to be implemented.

The *Vimes* retrieval architecture will be implemented in several components. Here we will be presenting the broker component, starting with the results of our requirements investigation using *Use Cases*. These results lead into the section where we will present our design choices. This will finish up with a short discussion of the implementation with the reader being pointed to the current location of the software.

Furthermore, in the rest of this paper the reader is assumed to be familiar with at least (Gils et al., 2003a) and (Gils et al., 2004).

2 Requirements

This section will present the results of our requirements investigation based on *Use Cases*.

2.1 Problem Statement

To implement the *Vimes* retrieval architecture as described in the introduction, a broker component will be needed to mediate between the user, the transfor-

mation component and the search component.

2.2 Statement of work

The realization of the broker component will be considered completed when each and every use case has been implemented. An analysis of the requirements will be made using use cases, which will function as the contract with which we determine completion of the broker component.

2.3 Stakeholders

The following have been identified as stakeholders in this project:

- Bas van Gils - primary researcher who will be validating his research with the *Vimes* retrieval architecture.
- Erik Proper - supervisor for the *PRONIR* research project of which Bas van Gils research is a part of.

2.4 Actors

The following list includes all actors that are the initiation point for a use case:

- User (provides search query requests).
- Searcher (component that inputs search results).
- Transformer (component that inputs transformations).

2.5 Defined use cases

The following table shows a listing of use cases as defined for completing the *Vimes* broker functionality:

- Process user request.
- Send search request.
- Send transform request.
- Process search results.
- Process transform results.
- Process queue.
- Send user results.

2.5.1 Process user request

This use case deals with the incoming data for the users query. It will need to be registered, queued and processed. Furthermore, the eventual results will need to be returned to the *Vimes* user interface component.

Use Case Name:	Process user request
Description:	The broker will provide a mechanism for processing user retrieval requests from the user interface component.
Actors:	User
Preconditions:	<ol style="list-style-type: none">1. Broker is reachable for User.2. Database is reachable for the broker (queue).
Triggers:	User requests a search be completed by submitting a query through the user interface component.
Basic Course of Events:	<ol style="list-style-type: none">1. The User submits a search request through the user interface component.2. The request is queued by the broker.3. The User is notified that the request is accepted.
Exceptions:	
Postconditions:	<ol style="list-style-type: none">1. Request for searching has been accepted and is in the queue.2. User has been notified.

2.5.2 Send search request

The broker will need to interact with the *Vimes* search component. This use case deals with sending user requests on to the search component for processing.

Use Case Name:	Send search request
Description:	The broker will provide a mechanism for sending eventual requests on to the search component.
Actors:	Searcher
Preconditions:	<ol style="list-style-type: none">1. Searcher is reachable for broker.2. Database is reachable for broker (queue).
Triggers:	A queue run (processing the queued search queries).
Basic Course of Events:	<ol style="list-style-type: none">1. Broker has job from the queue that needs to be sent to Searcher.2. Send job to Searcher for processing.3. Job queue is updated to reflect being sent to Searcher.
Exceptions:	
Postconditions:	<ol style="list-style-type: none">1. Job has been sent to the Searcher.2. Job queue has been updated.

2.5.3 Send transform request

The broker will need to interact with the *Vimes* transformation component. This use case details the passing of transformation requests on to the transformation component.

Use Case Name:	Send transform request
Description:	The broker will be able to send transformation requests based on user preferences (form/format).
Actors:	Transformer
Preconditions:	<ol style="list-style-type: none">1. Transformer is reachable for broker.2. Database is reachable for broker (queue).
Triggers:	A queue run.
Basic Course of Events:	<ol style="list-style-type: none">1. Broker has job from queue that needs to be sent to the Transformer.2. Send job to Transformer for processing.3. Job queue is updated to reflect being sent to Transformer.
Exceptions:	
Postconditions:	<ol style="list-style-type: none">1. Job has been sent to the Transformer.2. Job queue has been updated.

2.5.4 Process search results

The broker will need to interact with the *Vimes* search component. This use case will detail the process of processing the users search request results that the search component returns.

Use Case Name:	Process search results
Description:	The broker will provide a mechanism for receiving search results from the search component.
Actors:	Searcher
Preconditions:	<ol style="list-style-type: none">1. Broker component is reachable for Searcher.2. Database is reachable for broker (queue).
Triggers:	Broker receives the results of a search query from the Searcher.
Basic Course of Events:	<ol style="list-style-type: none">1. Broker receives results of a search query job from the Searcher.2. Response is cached if appropriate.3. Response is evaluated to determine if it completes the related job or not.4. Job entry in queue is updated to show new status.
Exceptions:	None.
Postconditions:	<ol style="list-style-type: none">1. Results of a job has been registered in the queue.2. Results of a job can result in updated cache.

2.5.5 Process transform results

The broker will need to interact with the *Vimes* transformation component. This use case handles the processing of transformation results from the transformation component.

Use Case Name:	Receive transform results
Description:	The broker will provide a mechanism for receiving transformation results from the transform component.
Actors:	Transformer
Preconditions:	<ol style="list-style-type: none">1. Broker component is reachable for Transformer.2. Database is reachable for broker (queue).
Triggers:	Broker receives the results of a transformation request from the Transformer.
Basic Course of Events:	<ol style="list-style-type: none">1. Broker receives results of a transformation request from the Transformer.2. Response is evaluated to determine if it completes the related job or not.3. Job entry in queue is updated to show new status.
Exceptions:	None.
Postconditions:	Results of a transformation request has been registered in the queue.

2.5.6 Process queue

This use case will describe the processing of the jobs that are still awaiting some action. These actions can be transformations, search query results or completed results that need to be returned to the user interface component.

Use Case Name:	Process queue
Description:	The user submitted search request jobs are processed after being submitted into the job queue. The broker is responsible for all logic involved with processing the search jobs and for resolving them into finished results to be sent back to the user interface component.
Actors:	Searcher, Transformer, User
Preconditions:	<ol style="list-style-type: none"> 1. Database is reachable for broker (queue). 2. Queue is not empty.
Triggers:	<ol style="list-style-type: none"> 1. Process user request. 2. Process search results. 3. Process transform results.
Basic Course of Events:	<ol style="list-style-type: none"> 1. Broker retrieves job from queue. 2. Broker checks for job dependencies (all completed?). 3. As needed, (dependent) job triggers send search request. 4. As needed, (dependent) job triggers send transform request. 5. As needed, job status in queue updated. 6. Job completed, triggers send user results. 7. Repeat until end of queue reached.
Exceptions:	<ol style="list-style-type: none"> 1. Searcher is unreachable, re-queue job. 2. Transformer is unreachable, re-queue job. 3. User is unreachable, re-queue job.
Postconditions:	Job queue processed, resulting in updated queue.

2.5.7 Send user results

This use case deals with returning the resulting data from a users query. It will need to be returned to the user and the queue cleaned out.

Use Case Name:	Send user results
Description:	The broker will be able to send results of user queries back to the user.
Actors:	Searcher, Transform
Preconditions:	<ol style="list-style-type: none">1. User is reachable for the broker.2. Database is reachable for the broker (queue).
Triggers:	Job reaches completed status in the queue.
Basic Course of Events:	<ol style="list-style-type: none">1. A job in the queue has reached completed status.2. The user search results are returned to the User.3. The request is dequeued by the broker.
Exceptions:	None.
Postconditions:	<ol style="list-style-type: none">1. Requested search result has been returned to User.2. Job (all traces) has been removed from the queue.

2.6 Scenarios

Here you will find each use case description with as many scenarios as needed to quantify the individual use cases.

2.6.1 Process user request

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Process user request
Use Case Steps:	<ol style="list-style-type: none">1. User submits a validated search query to Vimes.2. Data is processed into a request that is queued:<ol style="list-style-type: none">(a) <u>keywords</u>(b) <u>forms</u>(c) <u>formats</u>(d) <u>limits</u>(e) <u>email</u>3. Broker queues request.4. Broker notifies user request has been accepted.
Alternative Path:	<ol style="list-style-type: none">1. Broker notifies user that request has not been accepted, with back button.

2.6.2 Send search request

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Send search request
Use Case Steps:	<ol style="list-style-type: none">1. Broker retrieves a queued request.2. Broker sends request to Search component:<ol style="list-style-type: none">(a) <u>request_id</u>(b) <u>keywords</u>(c) forms(d) formats(e) limits3. Broker annotates request as sent to Search component.4. Broker queues request.
Alternative Path:	<ol style="list-style-type: none">1. Broker is unable to send request to Search component, just re-queue request unannotated.

2.6.3 Send transform request

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Send transform request
Use Case Steps:	<ol style="list-style-type: none">1. Broker retrieves a queued request.2. Broker sends request to Transform component:<ol style="list-style-type: none">(a) <u>request_id</u>(b) <u>results</u>(c) <u>forms</u>(d) <u>formats</u>3. Broker annotates request as sent to Transform component.4. Broker queues request.
Alternative Path:	<ol style="list-style-type: none">1. Broker is unable to send request to Transform component, just re-queue request unannotated.

2.6.4 Process search results

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Process search results
Use Case Steps:	<ol style="list-style-type: none">1. Broker receives a completed search query from the Searcher:<ol style="list-style-type: none">(a) <u>request_id</u>(b) <u>search_results</u>2. Broker caches response.3. Broker annotates request in queue as Search completed.
Alternative Path:	<ol style="list-style-type: none">1. None.

2.6.5 Process transform results

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Process transform results
Use Case Steps:	<ol style="list-style-type: none">1. Broker receives a completed transformation results form the Transformer:<ol style="list-style-type: none">(a) <u>request_id</u>(b) <u>search_results</u>(c) <u>transform_results</u>2. Broker caches response.3. Broker annotates request in queue as Transform completed.
Alternative Path:	<ol style="list-style-type: none">1. None.

2.6.6 Process queue

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Process queue
Use Case Steps:	<ol style="list-style-type: none">1. Broker receives process request queue.2. Broker processes each queued request for status changes.3. Requests processes search results:<ol style="list-style-type: none">(a) Requests back from Searcher but marked for transformations are sent to Transformer.(b) Requests back from Searcher not needing transformations are marked completed.(c) Requests marked completed are sent back with results to User via provided email.4. Requests processes transform results:<ol style="list-style-type: none">(a) Requests back from Transformer are marked as completed.(b) Requests marked completed are sent back with results to User.5. Any completed results are removed from the queue.
Alternative Path:	<ol style="list-style-type: none">1. Any problems related to requests in the queue always results in the request not being altered and left in queue.

2.6.7 Send user results

The following scenario details an example usage of the use case including relevant data.

Use Case Name:	Send user results
Use Case Steps:	<ol style="list-style-type: none">1. Broker retrieves request from queue that has completed.2. Broker sends request results to User:<ol style="list-style-type: none">(a) <u>request_id</u>(b) <u>search_results</u>(c) <u>email</u>3. Broker removes completed request from queue.
Alternative Path:	<ol style="list-style-type: none">1. Should Broker be unable to send completed request results to User, then request remains in completed status in queue.

3 Design

This section will present an overview of our design choices for the *Vimes* broker component.

3.1 Class diagrams

An overview of the used classes is given in an general diagram without any details presented in the classes themselves. Following this, the individual classes will be presented in more detail with attributes and methods being shown.

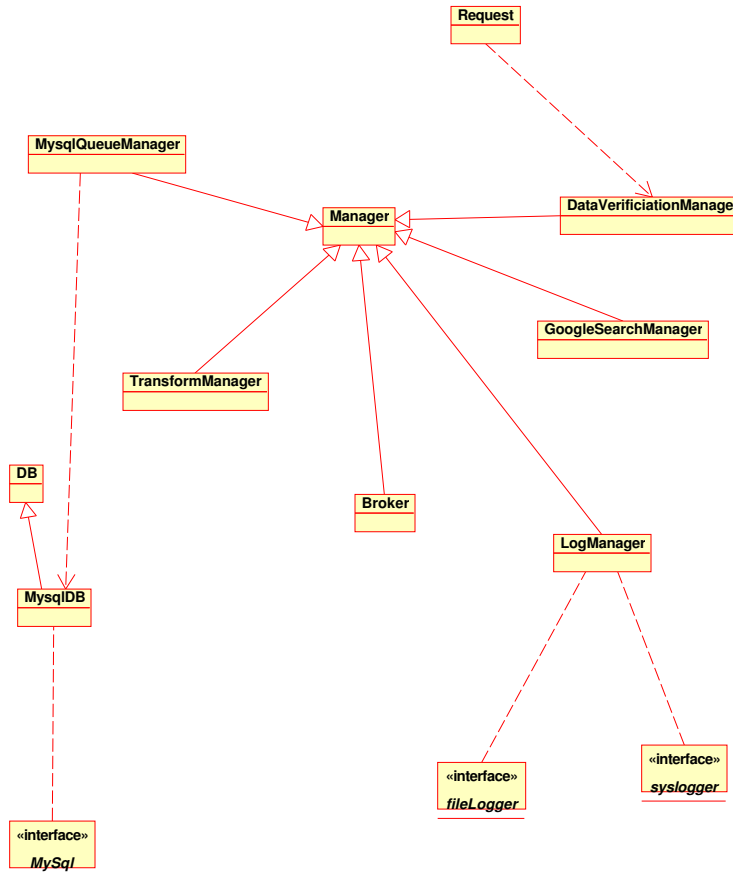


Figure 1: Class diagram overview

3.1.1 Broker

Manager implementation that is responsible for providing services to coordinate all interaction with the *Vimes retrieval architecture* and the User. The Broker will ensure that requests are processed and that results are provided to the User.

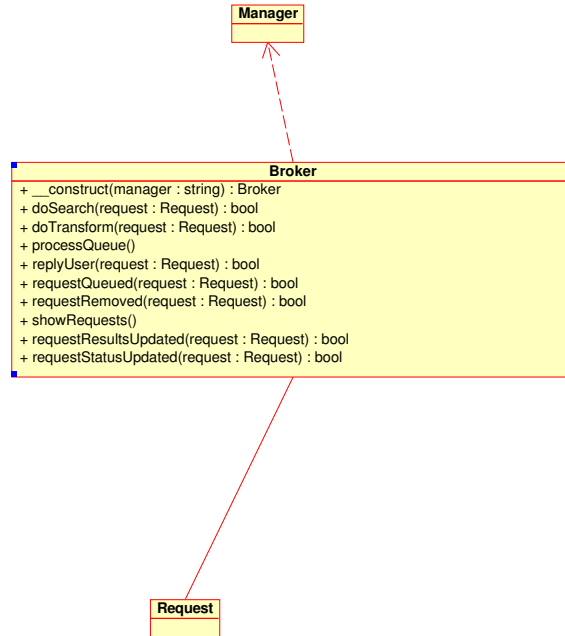


Figure 2: Broker class diagram

4 Implementation

The implementation of the Broker is to be done in PHP (version 5), using object oriented design principles. We have a running prototype with only limited access at:

http://osiris.cs.kun.nl/vimes/vimes_ui/vimes.php

For the complete overview of all generated class documentation we refer you to the online documentation at:

http://osiris.cs.kun.nl/vimes/vimes_classdocs

4.1 Broker implementation

Listing 1: Broker Class

```
1
2 <?
3
4 /**
5  * @author Eric Schabell <erics@cs.ru.nl>
6  * @copyright Copyright 2005, GPL
7  * @package VIMES
8  */
9
10 // const defines.
11 //
12 require_once( "const.inc" );
13
14 /**
15  * Broker class - deals with user requests and makes use of the rest of
16  * the Vimes
17  * framework for searching and transforming retrieval results. This
18  * class
19  * is a sub-class of Manager.
20  * @access public
21  *
22  * @package VIMES
23  * @subpackage Manager
24  */
25 class Broker extends Manager
26 {
27     /**
28     * Constructor - initialize the Broker.
29     * @access public
30     *
31     * @param string Type is Broker.
32     * @return Broker Broker object.
33     */
34     public function __construct( $manager="Broker" )
35     {
36         parent::__construct( $manager );
37     }
38
39     /**
40     * doSearch - sends a request off to the Search component for
41     * processing and updates status of the request. Should the
42     * Broker
43     * be unable to contact the Search component (returns false)
44     * then
45     * the job will remain in the queue and the status will not be
46     * changed.
47     * @access public
48     *
49     * @param Request The request object to be sent.
50     * @return bool True if request search done, otherwise False.
```

```

46 */
47 public function doSearch( $request )
48 {
49     $dataArray = $request->getRequestData();
50
51     // pass request off to the search component.
52     //
53     $gsm = new GoogleSearchManager;
54
55     // set our search info.
56     //
57     $gsm->setKey( "6KDTjCDfy0oGl/n+QC7GZQsveJkQw8bT" );
58     $gsm->setQueryString( $dataArray['keywords'] );
59     $gsm->setMaxResults( $dataArray['limits'] );
60     $gsm->setSafeSearch( TRUE );
61
62     // do the search.
63     //
64     $search_results = $gsm->doSearch();
65     if ( !$search_results )
66     {
67         // errors occurred.
68         //
69         parent::setErrorMsg( $gsm->getError() );
70         return FALSE;
71     }
72     else
73     {
74         // success, set status to search, save results
75         // and update status.
76         //
77         $request->setRequestStatus( REQUEST_SEARCH );
78         $request->setRequestResults( array(
79             $search_results ) );
80
81         if ( ! $this->requestStatusUpdated( $request ) )
82         {
83             parent::setErrorMsg( "Search completed
84             , but unable to set status to
85             searched, leaving in queu marked
86             as queued..." );
87             return FALSE;
88         }
89         elseif ( ! $this->requestResultsUpdated(
90             $request ) )
91         {
92             parent::setErrorMsg( "Search done, but
93             unable to save results, leaving
94             in queue marked as search.." );
95             return FALSE;
96         }
97     }
98     else
99     {
100         $request->setRequestStatus(
101             REQUEST_SEARCHED );
102         if ( ! $this->requestStatusUpdated(
103             $request ) )
104         {
105             parent::setErrorMsg( "Search
106             completed, set results,
107             but unable to set status
108             to final searched status
109             ..." );
110             return FALSE;
111         }
112     }
113 }
114 // search completed, results saved, status on final
115 searched.

```

```

100         //
101         return TRUE;
102     }
103
104     /**
105     * doTransform - sends a request off to the Transform component
106     * for
107     * processing and updates status of the request. Should the
108     * Broker
109     * be unable to contact the Transform component (returns false)
110     * then
111     * the job will remain in the queue and the status will not be
112     * changed.
113     * @access public
114     *
115     * @param Request The request object to be sent.
116     * @return bool True if request transformation done, otherwise
117     *         False.
118     */
119     public function doTransform( $request )
120     {
121         // pass request off to the search component.
122         //
123         $transform = new TransformManager;
124         $transform_results = $transform->doTransform( $request
125             );
126
127         if ( !$transform_results )
128         {
129             // errors occurred.
130             //
131             parent::setErrorMsg( $transform->getError() );
132             return FALSE;
133         }
134         else
135         {
136             // check success, set status to transformed,
137             // update status,
138             // requeue request.
139             //
140             if ( $request->getRequestStatus() ==
141                 REQUEST_TRANSFORM )
142             {
143                 $request->setRequestStatus(
144                     REQUEST_TRANSFORMED );
145
146                 if ( !$this->requestStatusUpdated(
147                     $request ) )
148                 {
149                     parent::setErrorMsg( "
150                         Transform completed, but
151                         unable to queue status to
152                         searched, leaving in queue
153                         with nothing updated..." );
154                     ;
155                     return FALSE;
156                 }
157             }
158             /**
159             // TODO: implement update once we
160             // actually do something... think it
161             // will not be here but in Transform
162             // class.
163             elseif ( !$this->
164                 requestResultsUpdated( $request )
165                 )
166             {
167                 // need to roll back status
168                 // update.
169                 //

```

```

148         $request->setRequestStatus(
149             REQUEST_SEARCHED );
150         if ( ! $this->
151             requestStatusUpdated(
152                 $request ) )
153         {
154             // something wrong,
155             // don't set errorMsg
156             // as we are
157             // interested in what
158             // the method called
159             // has to say
160             // about this error.
161             return FALSE;
162         }
163         parent::setErrorMsg( "
164             Transform unable to save
165             results, rolled back queue
166             to searched status..." );
167         return FALSE;
168     }
169     */
170 }
171 // transform processed, results saved, status final
172 // transformed.
173 //
174 // return TRUE;
175 }
176
177 /**
178 * replyUser - send results to user via email provided.
179 * @access public
180 *
181 * @param Request The request to be sent to user.
182 * @return bool True if sent, otherwise False.
183 */
184 public function replyUser( $request )
185 {
186     // get email.
187     //
188     $dataArray = $request->getRequestData();
189     $email = $dataArray['email'];
190
191     // get results array.
192     //
193     $resultsArray = $request->getRequestResults();
194     $search_result = $resultsArray[0];
195
196     // build email.
197     //
198     $message = "Results from your Vimes Retrieval
199         request:\n\n";
200     $message .= " Request number: " . $request->
201         getRequestId() . "\n";
202     $message .= " Keywords: " . $dataArray['
203         keywords' ] . "\n";
204     $message .= " Forms: " . $dataArray['forms' ]
205         . "\n";
206     $message .= " Formats: " . $dataArray['formats
207         ' ] . "\n";
208     $message .= " Limits: " . $dataArray['limits
209         ' ] . "\n\n";
210     $message .= "
211         ===== ";
212
213     // now add the results elements.
214     //
215     $re = $search_result->getResultElements();
216     foreach($re as $element)

```

```

200     {
201         $message .= "\n\n";
202         $message .= "                Title: " .
203             $element->getTitle() . "\n";
204         $message .= "                URL: " .
205             $element->getURL() . "\n";
206         $message .= "                Snippet: " .
207             $element->getSnippet() . "\n";
208         $message .= "                Summary: " .
209             $element->getSummary() . "\n";
210         $message .= "                Host Name: " .
211             $element->getHostName() . "\n";
212         $message .= "Related Info Present: " .
213             $element->getRelatedInformationPresent() .
214             "\n";
215         $message .= "                Cached Size: " .
216             $element->getCachedSize() . "\n";
217         $message .= "                Directory Title: " .
218             $element->getDirectoryTitle() . "\n";
219
220         $dircat = $element->getDirectoryCategory();
221
222         $message .= " Full Viewable Name: " . $dircat
223             ->getFullViewableName() . "\n";
224         $message .= " Special Encoding: " . $dircat
225             ->getSpecialEncoding() . "\n";
226     }
227
228     // send to user.
229     $headers = "From: Vimes Retrieval Architecture
230         prototype <basvg@cs.ru.nl>\r\n";
231     if ( !mail( $email, "Vimes Retrieval Results Report",
232         $message, $headers ) )
233     {
234         parent::setErrorMsg( "Unable to send user mail
235             with results, leaving request in queue
236             ..." );
237         return FALSE;
238     }
239
240     // set to finished and remove from queue.
241     //
242     $request->setRequestStatus( REQUEST_FINISHED );
243     if ( ! $this->requestStatusUpdated( $request ) )
244     {
245         $msg = "Mail sent to user with results, but
246             unable to ";
247         $msg .= "update request number ' " . $request->
248             getRequestID();
249         $msg .= "'to status FINISHED, leaving in queue
250             ... ";
251         parent::setErrorMsg( $msg );
252         return FALSE;
253     }
254
255     // results returned, status updated, removed from
256     queue.
257     //
258     return TRUE;
259 }
260
261 /**
262 * processQueue - runs the contents of the queue, processing
263     each request based on the
264 * actions still to be performed in this order; Search ->
265     Transform -> Reply -> Delete.
266 * @access public
267 *
268 * @return void
269 */

```



```

249 public function processQueue()
250 {
251     // process the entire current queue
252     //
253     $mqm = new MysqlQueueManager;
254     $log = new LogManager;
255
256     if ( count( $queueArray = $mqm->getQueued() ) == 0 )
257     {
258         // nothing in the queue.
259         //
260         $msg = "Nothing in queue, number of entries: "
261             . count( $queueArray );
262         $log->fileLogger( $msg );
263         return;
264     }
265
266     // loop thru jobs, checking for states; SEARCH,
267     SEARCHED, TRANSFORM, TRANSFORMED,
268     // FINISHED and deal with them.
269     //
270     foreach ( $queueArray as $request )
271     {
272         switch ( $request->getRequestStatus() )
273         {
274             case 0: // REQUEST_START.
275
276                 // need to do a search.
277                 //
278                 if ( ! $this->doSearch( $request ) )
279                 {
280                     // failed, log this.
281                     $log->fileLogger( $this->
282                         getErrorMsg() );
283                     break;
284                 }
285
286                 // success, log this.
287                 //
288                 $msg = "Search completed for this
289                     queued request: " . $request->
290                     getRequestID();
291                 $msg .= " / " . $request->
292                     getRequestStatus();
293                 $log->fileLogger( $msg );
294                 break;
295
296             case 1: // REQUEST_SEARCH.
297
298                 // need to do a search.
299                 //
300                 if ( ! $this->doSearch( $request ) )
301                 {
302                     // failed, log this.
303                     $log->fileLogger( $this->
304                         getErrorMsg() );
305                     break;
306                 }
307
308                 // success, log this.
309                 //
310                 $msg = "Search completed for this
311                     queued request: " . $request->
312                     getRequestID();
313                 $msg .= " / " . $request->
314                     getRequestStatus();
315                 $log->fileLogger( $msg );
316                 break;
317
318             case 2: // REQUEST_SEARCHED.

```

```

309
310 // need to do a transform.
311 //
312 if ( ! $this->doTransform( $request )
313 )
314 {
315     // failed, log this.
316     $log->fileLogger( $this->
317         getErrorMsg() );
318     break;
319 }
320 // success, log this.
321 //
322 $msg = "Transform completed for this
323     queued request: " . $request->
324     getRequestID();
325 $msg .= " / " . $request->
326     getRequestStatus();
327 $log->fileLogger( $msg );
328 break;
329
330 case 3: // REQUEST_TRANSFORM.
331
332 // need to do a transform.
333 //
334 if ( ! $this->doTransform( $request )
335 )
336 {
337     // failed, log this.
338     $log->fileLogger( $this->
339         getErrorMsg() );
340     break;
341 }
342 // success, log this.
343 //
344 $msg = "Transform completed for this
345     queued request: " . $request->
346     getRequestID();
347 $msg .= " / " . $request->
348     getRequestStatus();
349 $log->fileLogger( $msg );
350 break;
351
352 case 4: // REQUEST_TRANSFORMED.
353
354 // need to reply to user.
355 //
356 if ( ! $this->replyUser( $request ) )
357 {
358     // failed, log this.
359     $log->fileLogger( $this->
360         getErrorMsg() );
361     break;
362 }
363 // success, log this.
364 //
365 $msg = "Replied to user completed for
366     this queued request: " . $request
367     ->getRequestID();
368 $msg .= " / " . $request->
369     getRequestStatus();
370 $log->fileLogger( $msg );
371 break;
372
373 case 5: // REQUEST_FINISHED.

```

```

365         // need to remove this job.
366         //
367         if ( ! $this->requestRemoved( $request
368             ) )
369         {
370             // failed, log this.
371             $log->fileLogger( $this->
372                 getErrorMsg() );
373             break;
374         }
375         // success, log this.
376         //
377         $msg = "Removed request : " .
378             $request->getRequestID();
379         $msg .= " / " . $request->
380             getRequestStatus() . " as finished
381             processing!";
382         $log->fileLogger( $msg );
383         break;
384     }
385 }
386
387 /**
388 * requestQueued - adds new request to request queue.
389 * @access public
390 *
391 * @param Request The request object to be added to the queue.
392 * @return bool True if request queued, otherwise False.
393 */
394 public function requestQueued( $request )
395 {
396     $mqm = new MysqlQueueManager;
397     $log = new LogManager;
398
399     if ( ! $mqm->enqueued( $request ) )
400     {
401         parent::setErrorMsg( "Unable to enqueue the
402             given Request..." );
403         return FALSE;
404     }
405
406     $msg = "Request enqueued: " . $request->getRequestID()
407         . " / " . $request->getRequestStatus();
408     $log->fileLogger( $msg );
409     return TRUE;
410 }
411
412 /**
413 * requestRemoved - deletes request from the request queue.
414 * @access public
415 *
416 * @param Request The request object to be removed from the
417     queue.
418 * @return bool True if request is removed from queue,
419     otherwise False.
420 */
421 public function requestRemoved( $request )
422 {
423     $mqm = new MysqlQueueManager;
424     if ( ! $mqm->dequeued( $request ) )
425     {
426         parent::setErrorMsg( "Unable to dequeue the
427             given Request..." );
428         return FALSE;
429     }
430
431     return TRUE;

```

```

425     }
426
427
428     /**
429     * showRequests - print queue listing.
430     * @access public
431     *
432     * @return void
433     */
434     public function showRequests()
435     {
436         // dump queue to stdout.
437         //
438         $mqm = new MysqlQueueManager;
439         $mqm->printQueueToScreen();
440         return;
441     }
442
443     /**
444     * requestResultsUpdated - updates the request results of queue
445     * entry in
446     * database.
447     * @access public
448     *
449     * @param Request Request object to be updated.
450     * @return bool True if updated, otherwise false.
451     */
452     public function requestResultsUpdated( $request )
453     {
454         $serial_results = serialize( $request->
455             getRequestResults() );
456
457         $update = "UPDATE queue ";
458         $update .= " SET requestresults= '" . $serial_results
459             . "' ";
460         $update .= "WHERE requestid = '" . $request->
461             getRequestID() . "' ";
462
463         $db = new MysqlDB();
464
465         if ( ! $db->connected() )
466         {
467             parent::setErrorMsg( "Unable to connect to
468                 database..." );
469             return FALSE;
470         }
471
472         // update returns nr affected rows, should only be one
473         // !
474         //
475         $results = $db->execute($update);
476
477         if ( $results != 1 )
478         {
479             parent::setErrorMsg( "Update of request
480                 results did not affect a single row as it
481                 should have..." );
482             return FALSE;
483         }
484
485         // results updated.
486         //
487         return TRUE;
488     }
489
490     /**
491     * requestStatusUpdated - updates the request status from queue
492     * entry in
493     * database.
494     * @access public
495

```

```

486      *
487      * @param Request Request object to be updated.
488      * @return bool True if updated, otherwise false.
489      */
490      public function requestStatusUpdated( $request )
491      {
492          $update = "UPDATE queue ";
493          $update .= " SET requeststatus = '" . $request->
494              getRequestStatus() . "' ";
495          $update .= "WHERE requestid = '" . $request->
496              getRequestID() . "'";
497
498          $db = new MySQLDB();
499
500          if ( ! $db->connected() )
501          {
502              parent::setErrorMsg( "Unable to connect to
503                  database..." );
504              return FALSE;
505          }
506
507          // update returns nr affected rows, should only be one
508          //
509          $results = $db->execute($update);
510
511          if ( $results != 1 )
512          {
513              parent::setErrorMsg( "Update of request status
514                  did not affect a single row as it should
515                  have..." );
516              return FALSE;
517          }
518
519          // status updated.
520          //
521          return TRUE;
522      }
523  }
524  ?>

```

References

- Gils, B. v., Proper, H., and Bommel, P. v. (2003a). A conceptual model for information supply. *Data & Knowledge Engineering*, 51:189–222.
- Gils, B. v., Proper, H., Bommel, P. v., and Schabell, E. (2003b). Profile-based retrieval on the world wide web. In Bra, P. d., editor, *Proceedings of the Conferentie Informatiewetenschap (INFWET2003)*, pages 91–98, Eindhoven, The Netherlands, EU.
- Gils, B. v., Proper, H., Bommel, P. v., and Vrieze, P. d. (2004). Transformation selection for aptness-based web retrieval. Technical report, Radboud University Nijmegen Institute for Computing and Information Science. accepted for publication in: Australian Database Conference 2005 (ADC-2005).