Implementing Vimes - the broker component.

Eric D. Schabell
ericks@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 - suppervisor 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:	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:	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:	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:	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:	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:	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:	1. Transformer is reachable for broker. 2. Database is reachable for broker (queue).
Triggers:	A queue run.
Basic Course of Events:	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:	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:	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:	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:	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:	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:	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:	1. Database is reachable for broker (queue). 2. Queue is not empty.
Triggers:	1. Process user request. 2. Process search results. 3. Process transform results.
Basic Course of Events:	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:	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:	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:	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:	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:	1. User submits a validated search query to Vimes. 2. Data is processed into a request that is queued: (a) keywords (b) forms (c) formats (d) limits (e) email 3. Broker queues request. 4. Broker notifies user request has been accepted.
Alternative Path:	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:	1. Broker retrieves a queued request. 2. Broker sends request to Search component: (a) request_id (b) keywords (c) forms (d) formats (e) limits 3. Broker annotates request as sent to Search component. 4. Broker queues request.
Alternative Path:	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:	1. Broker retrieves a queued request. 2. Broker sends request to Transform component: (a) request_id (b) results (c) forms (d) formats 3. Broker annotates request as sent to Transform component. 4. Broker queues request.
Alternative Path:	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:	1. Broker receives a completed search query form the Searcher: (a) request_id (b) search_results 2. Broker caches response. 3. Broker annotates request in queue as Search completed.
Alternative Path:	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:	1. Broker receives a completed transformation results form the Transformer: (a) request_id (b) search_results (c) transform_results 2. Broker caches response. 3. Broker annotates request in queue as Transform completed.
Alternative Path:	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:	1. Broker receives process request queue. 2. Broker processes each queued request for status changes. 3. Requests processes search results: (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: (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:	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:	1. Broker retrieves request from queue that has completed. 2. Broker sends request results to User: (a) request_id (b) search_results (c) email 3. Broker removes completed request from queue.
Alternative Path:	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.

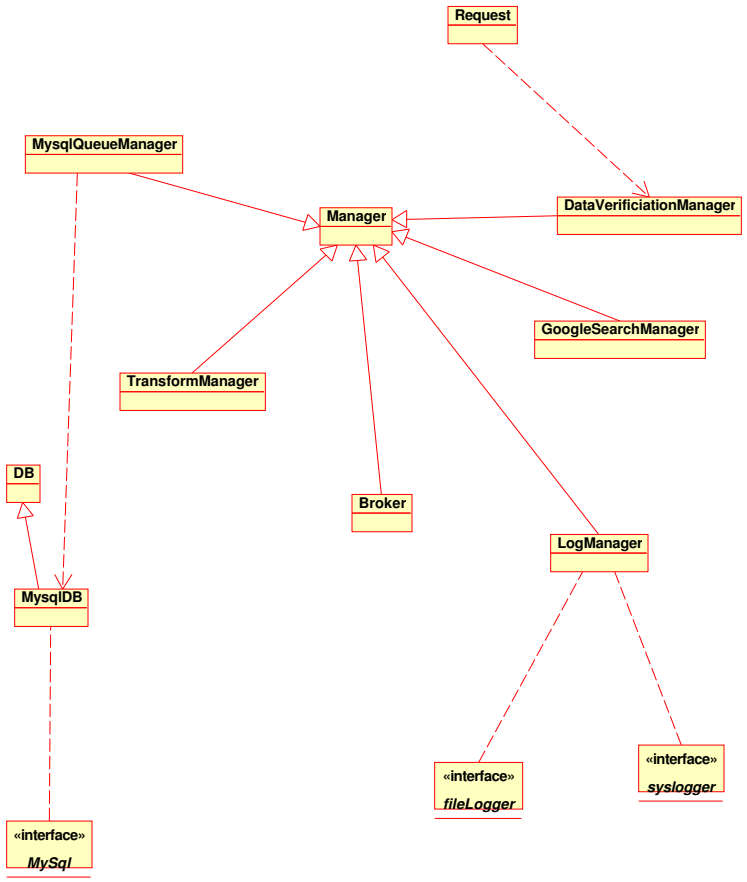

Figure 1: Class diagram overview

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.

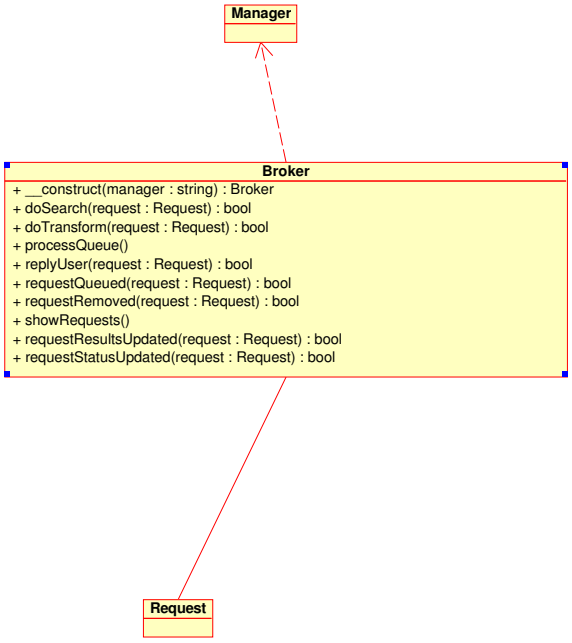

Figure 2: Broker class diagram

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
       the Vimes
16 * framework for searching and transforming retrieval results. This
      class
17 * is a sub-class of Manager.
18 * @access public
19 *
20 * @package VIMES
21 * @subpackage Manager
22 */
23 class Broker extends Manager
24 {
25         /**
26         * Constructor - initialize the Broker.
27         * @access public
28         *
29         * @param string Type is Broker.
30         * @return Broker Broker object.
31         */
32         public function __construct( $manager="Broker" )
33         {
34                 parent::__construct( $manager );
35   }
36
37         /**
38         * doSearch - sends a request off to the Search component for
39         * processing and updates status of the request. Should the
             Broker
40         * be unable to contact the Search component (returns false)
             then
41         * the job will remain in the queue and the status will not be
             changed.
42         * @access public
43         *
44         * @param Request The request object to be sent.
45         * @return bool True if request search done, otherwise False.
```

```
        */
        public function doSearch( $request )
        {
                $dataArray = $request->getRequestData();

                // pass request off to the search component.
                //
                $gsm = new GoogleSearchManager;

                // set our search info.
                //
                $gsm->setKey( "6KDTjCDfy0oGl/n+QC7GZQsveJkQw8bT" );
                $gsm->setQueryString( $dataArray['keywords'] );
                $gsm->setMaxResults( $dataArray['limits'] );
                $gsm->setSafeSearch( TRUE );

                // do the search.
                //
                $search_results = $gsm->doSearch();
                if ( !$search_results )
                {
                        // errors occurred.
                        //
                        parent::setErrorMsg( $gsm->getError() );
                        return FALSE;
                }
                else
                {
                        // success, set status to search, save results
                             and update status.
                        //
                        $request->setRequestStatus( REQUEST_SEARCH );
                        $request->setRequestResults( array(
                            $search_results ) );

                        if ( ! $this->requestStatusUpdated( $request )
                             )
                        {
                                parent::setErrorMsg( "Search completed
                                    , but unable to set status to
                                    searched, leaving in queu marked
                                    as queued...");
                                return FALSE;
                        }
                        elseif ( ! $this->requestResultsUpdated(
                            $request ) )
                        {
                                parent::setErrorMsg( "Search done, but
                                     unable to save results, leaving
                                    in queue marked as search..");
                                return FALSE;
                        }
                        else
                        {
                                $request->setRequestStatus(
                                    REQUEST_SEARCHED );
                                if ( ! $this->requestStatusUpdated(
                                    $request ) )
                                {
                                        parent::setErrorMsg( "Search
                                            completed, set results,
                                            but unable to set status
                                            to final searched status
                                            ..." );
                                        return FALSE;
                                }
                        }
                }
                // search completed, results saved, status on final
                    searched.
```

```
                //
                return TRUE;
        }

        /**
        * doTransform - sends a request off to the Transform component
             for
        * processing and updates status of the request. Should the
            Broker
        * be unable to contact the Transform component (returns false)
             then
        * the job will remain in the queue and the status will not be
            changed.
        * @access public
        *
        * @param Request The request object to be sent.
        * @return bool True if request transformation done, otherwise
            False.
        */
        public function doTransform( $request )
        {
                // pass request off to the search component.
                //
                $transform = new TransformManager;
                $transform_results = $transform->doTransform( $request
                     );

                if ( !$transform_results )
                {
                        // errors occurred.
                        //
                        parent::setErrorMsg( $transform->getError() );
                        return FALSE;
                }
                else
                {
                        // check success, set status to transformed,
                            update status,
                        // requeue request.
                        //
                        if ( $request->getRequestStatus() ==
                            REQUEST_TRANSFORM )
                        {
                                $request->setRequestStatus(
                                    REQUEST_TRANSFORMED );

                                if ( ! $this->requestStatusUpdated(
                                    $request ) )
                                {
                                        parent::setErrorMsg( "
                                            Transform completed, but
                                            unable to queue status to
                                            searched, leaving in queue
                                             with nothing updated...")
                                            ;
                                        return FALSE;
                                }
                                /**
                                // TODO: implement update once we
                                    actually do something... think it
                                    will not be here but in Transform
                                    class.
                                elseif ( ! $this->
                                    requestResultsUpdated( $request )
                                    )
                                {
                                        // need to roll back status
                                            update.
                                        //
```

```
                                $request->setRequestStatus(
                                    REQUEST_SEARCHED );
                                    if ( ! $this->
                                        requestStatusUpdated(
                                        $request ) )
                                    {
                                            // something wrong,
                                                don't set errorMsg
                                                 as we are
                                            // interested in what
                                                the method called
                                                has to say
                                            // about this error.
                                            return FALSE;
                                    }
                                    parent::setErrorMsg( "
                                        Transform unable to save
                                        results, rolled back queue
                                         to searched status..." );
                                    return FALSE;
                            }
                            */
                    }
            }
            // transform processed, results saved, status final
                transformed.
            //
            return TRUE;
    }

    /**
    * replyUser - send results to user via email provided.
    * @access public
    *
    * @param Request The request to be sent to user.
    * @return bool True if sent, otherwise False.
    */
    public function replyUser( $request )
    {
            // get email.
            //
            $dataArray = $request->getRequestData();
            $email = $dataArray['email'];

            // get results array.
            //
            $resultsArray  = $request->getRequestResults();
            $search_result = $resultsArray[0];

            // build email.
            //
            $message        = "Results from your Vimes Retrieval
                request:\n\n";
            $message .= "  Request number: " . $request->
                getRequestID() . "\n";
            $message .= "        Keywords: " . $dataArray['
                keywords'] . "\n";
            $message .= "           Forms: " . $dataArray['forms']
                 . "\n";
            $message .= "         Formats: " . $dataArray['formats
                '] . "\n";
            $message .= "          Limits: " . $dataArray['limits
                '] . "\n\n";
            $message .= "
                ========================================  ";

            // now add the results elements.
            //
            $re = $search_result->getResultElements();
            foreach($re as $element)
```

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

```
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: "
                                . count( $queueArray );
261                         $log->fileLogger( $msg );
262                         return;
263                 }
264
265                 // loop thru jobs, checking for states; SEARCH,
                     SEARCHED, TRANSFORM, TRANSFORMED,
266                 // FINISHED and deal with them.
267                 //
268                 foreach ( $queueArray as $request )
269                 {
270                         switch ( $request->getRequestStatus() )
271                         {
272                         case 0:   // REQUEST_START.
273
274                                 // need to do a search.
275                                 //
276                                 if ( ! $this->doSearch( $request ) )
277                                 {
278                                         // failed, log this.
279                                         $log->fileLogger( $this->
                                             getErrorMsg() );
280                                         break;
281                                 }
282
283                                 // success, log this.
284                                 //
285                                 $msg  = "Search completed for this
                                     queued request: " . $request->
                                     getRequestID();
286                                 $msg .= " / " . $request->
                                     getRequestStatus();
287                                 $log->fileLogger( $msg );
288                                 break;
289
290                         case 1:  // REQUEST_SEARCH.
291
292                                 // need to do a search.
293                                 //
294                                 if ( ! $this->doSearch( $request ) )
295                                 {
296                                         // failed, log this.
297                                         $log->fileLogger( $this->
                                             getErrorMsg() );
298                                         break;
299                                 }
300
301                                 // success, log this.
302                                 //
303                                 $msg  = "Search completed for this
                                     queued request: " . $request->
                                     getRequestID();
304                                 $msg .= " / " . $request->
                                     getRequestStatus();
305                                 $log->fileLogger( $msg );
306                                 break;
307
308                         case 2:  // REQUEST_SEARCHED.
```

```
                        // need to do a transform.
                        //
                        if ( ! $this->doTransform( $request )
                            )
                        {
                                // failed, log this.
                                $log->fileLogger( $this->
                                    getErrorMsg() );
                                break;
                        }

                        // success, log this.
                        //
                        $msg  = "Transform completed for this
                            queued request: " . $request->
                            getRequestID();
                        $msg .= " / " . $request->
                            getRequestStatus();
                        $log->fileLogger( $msg );
                        break;

                case 3:  // REQUEST_TRANSFORM.

                        // need to do a transform.
                        //
                        if ( ! $this->doTransform( $request )
                            )
                        {
                                // failed, log this.
                                $log->fileLogger( $this->
                                    getErrorMsg() );
                                break;
                        }

                        // success, log this.
                        //
                        $msg  = "Transform completed for this
                            queued request: " . $request->
                            getRequestID();
                        $msg .= " / " . $request->
                            getRequestStatus();
                        $log->fileLogger( $msg );
                        break;


                case 4:  // REQUEST_TRANSFORMED.

                        // need to reply to user.
                        //
                        if ( ! $this->replyUser( $request ) )
                        {
                                // failed, log this.
                                $log->fileLogger( $this->
                                    getErrorMsg() );
                                break;
                        }

                        // success, log this.
                        //
                        $msg  = "Replied to user completed for
                             this queued request: " . $request
                            ->getRequestID();
                        $msg .= " / " . $request->
                            getRequestStatus();
                        $log->fileLogger( $msg );
                        break;

                case 5:  // REQUEST_FINISHED.
```

```
                                // need to remove this job.
                                //
                                if ( ! $this->requestRemoved( $request
                                     ) )
                                {
                                        // failed, log this.
                                        $log->fileLogger( $this->
                                            getErrorMsg() );
                                        break;
                                }

                                // success, log this.
                                //
                                $msg  = "Removed request : " .
                                    $request->getRequestID();
                                $msg .= " / " . $request->
                                    getRequestStatus() . " as finished
                                     processing!";
                                $log->fileLogger( $msg );
                                break;
                        }
                }
        }

        /**
        * requestQueued - adds new request to request queue.
        * @access public
        *
        * @param Request The request object to be added to the queue.
        * @return bool True if request queued, otherwise False.
        */
        public function requestQueued( $request )
        {
                $mqm = new MysqlQueueManager;
                $log = new LogManager;

                if ( ! $mqm->enqueued( $request ) )
                {
                        parent::setErrorMsg( "Unable to enqueue the
                            given Request...");
                        return FALSE;
                }

                $msg = "Request enqueued: " . $request->getRequestID()
                     . " / " . $request->getRequestStatus();
                $log->fileLogger( $msg );
                return TRUE;
        }


        /**
        * requestRemoved - deletes request from the request queue.
        * @access public
        *
        * @param Request The request object to be removed from the
            queue.
        * @return bool True if request is remvoed from queue,
            otherwise False.
        */
        public function requestRemoved( $request )
        {
                $mqm = new MysqlQueueManager;
                if ( ! $mqm->dequeued( $request ) )
                {
                        parent::setErrorMsg( "Unable to dequeue the
                            given Request...");
                        return FALSE;
                }

                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
                entry in
445         * database.
446         * @access public
447         *
448         * @param Request Request object to be updated.
449         * @return bool True if updated, otherwise false.
450         */
451         public function requestResultsUpdated( $request )
452         {
453                 $serial_results = serialize( $request->
                    getRequestResults() );
454
455                 $update  = "UPDATE queue ";
456                 $update .= " SET requestresults= '" . $serial_results
                        . "' ";
457                 $update .= "WHERE requestid = '" . $request->
                    getRequestID() . "';";
458
459                 $db = new MysqlDB();
460
461                 if ( ! $db->connected() )
462                 {
463                         parent::setErrorMsg( "Unable to connect to
                            database..." );
464                         return FALSE;
465                 }
466
467                 // update returns nr affected rows, should only be one
                        !
468                 //
469                 $results = $db->execute($update);
470
471                 if ( $results != 1 )
472                 {
473                         parent::setErrorMsg( "Update of request
                            results did not affect a single row as it
                            should have..." );
474                         return FALSE;
475                 }
476
477                 // results updated.
478                 //
479                 return TRUE;
480         }
481
482         /**
483         * requestStatusUpdated - updates the request status from queue
                entry in
484         * database.
485         * @access public
```

```
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->
                        getRequestStatus() . "' ";
494                 $update .= "WHERE requestid = '" . $request->
                        getRequestID() . "';";
495
496                 $db = new MysqlDB();
497
498                 if ( ! $db->connected() )
499                 {
500                         parent::setErrorMsg( "Unable to connect to
                                database..." );
501                         return FALSE;
502                 }
503
504                 // update returns nr affected rows, should only be one
                        !
505                 //
506                 $results = $db->execute($update);
507
508                 if ( $results  != 1 )
509                 {
510                         parent::setErrorMsg( "Update of request status
                                 did not affect a single row as it should
                                have..." );
511                         return FALSE;
512                 }
513
514                 // status updated.
515                 //
516                 return TRUE;
517         }
518 }
519
520 ?>
```

References

Gils, B. v., Proper, H., and Bommel, P. v. (2003a). A conceptual model for information suppy. *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).