Opening Up the Natural History Heritage for Europeana

D20 – DELIVERABLE 6.4.2

Project Acronym: **OpenUp!**

Grant Agreement No: **270890**

Project Title: **Opening up the Natural History Heritage for Europeana**

Productive system for caching environment

D20 – Deliverable 6.4.2

Revision: 5a

Authors:

Wolfgang Koller NHMW

Heimo Rainer NHMW

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

0 REVISION AND DISTRIBUTION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
1	2013-08-06	Wolfgang Koller	NHMW	Initial setup of document
2	2013-08-07	Wolfgang Koller	NHMW	Details on caching layers functionality
3	2013-08-09	Wolfgang Koller	NHMW	Incorporating feedback from Heimo Rainer
4	2013-08-14	Wolfgang Koller	NHMW, TMG	Extending document based on feedback from TMG
5	2013-08-20	Wolfgang Koller	NHMW	Final amendments
5a	2013-08-22	Coordination Team (A. Michel, P. Böttinger)	BGBM	Minor editing

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Distribution

Recipient	Date	Version	Accepted YES/NO
TMG	2013-08-09	3	NO
TMG	2013-08-14	4	YES
Work Package Leader (Heimo Rainer, WP6)	2013-08-07	2	YES
Project Coordinator (W. Berendsohn, BGBM)	2013-08-22	5a	YES

Table of Contents

1 OVERVIEW

The common names web service[1] provides an interface for retrieving common names for a given scientific name (for detailed information on the interface and the web service please consult D17-D6.2.4[2]). This process is heavily used during the enrichment phase of the content for Europeana (as outlined in Figure 1).

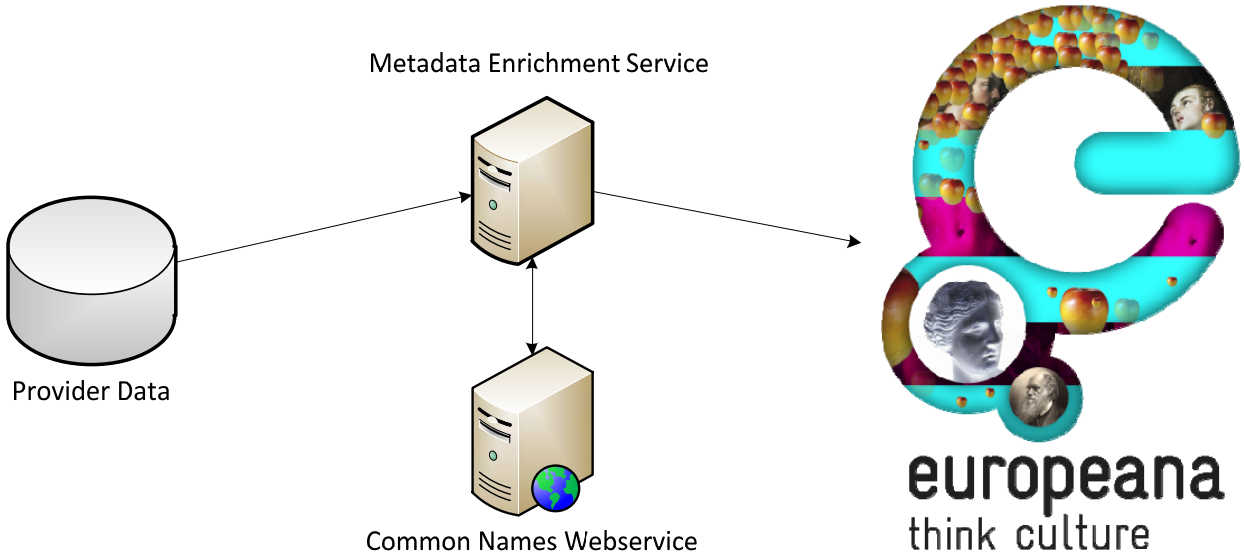

Figure 1: Common names web service utilization

In order to increase the performance of the common names web service, two caching layers are implemented. The first one being a service-level caching layer which is used to directly cache request and responses to the actual common names web service. If an incoming request was answered before, the response will be taken from a database cache and returned immediately (the request is therefore not forwarded to all available sources).

The second caching layer is part of the sub-service layer. It caches all requests to sub-services which are queried from the common names web service. In addition it takes care of timing out the cache if responses are too old.

While the first layer reduces the overall load, the second layer especially ensures that other services are not overloaded by the same requests from our service. Together with the GNA nameParser (which maps scientific names in different variants to a canonical version) this heavily reduces the load to external sources.

[1] http://openup.nhm-wien.ac.at/commonNames/

[2] D17-D6.2.4 Productive system for harvesting and parsing reference information

2 SERVICE CACHE

The service level cache is built on application level. As outlined in D17-D6.2.4 the service implementation is based on the Yii-Framework[3]. It offers an output caching facility which can be easily configured. Based on this the response to requests with the same “query” or “queries” parameter to the common names web service are fetched from a database cache instead of forwarded to all available sources.

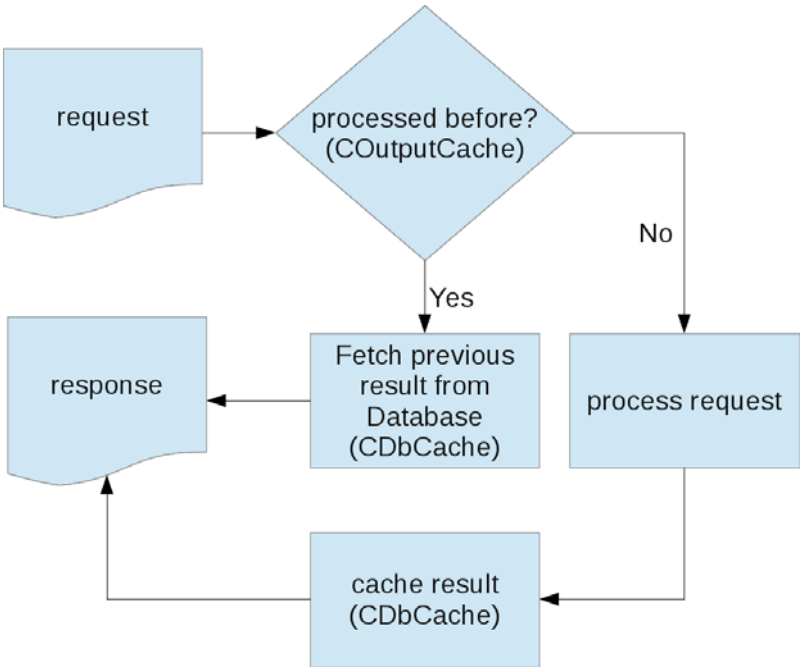

Figure 2: Processing of service caching layer

This yields a high performance gain since the request is not forwarded anymore, but instead the final output is returned immediately.

This works as outlined in Figure 22. Before a request is actually processed the COutputCache facility checks for a previously processed request which has the same “query” and “queries” parameters. If a previously processed result is found, it is fetched from the database using the CDbCache utility. If not, it is processed as usual and then stored for future calls. COutputCache also takes care of the lifetime of the cache, which is configured to be valid for 24 hours.

[3] http://www.yiiframework.com/

3 SUB-SERVICE CACHE

The sub service cache layer is based on a database cache as well. Since most of the web services use a different protocol, the raw response data is cached in a database in order to avoid duplicate calls to sub-services. In order to simplify the setup of new services, base classes were introduced to handle JSON-RPC, SOAP and REST services (which covers all cases encountered so far). As shown in Figure 33, all specialized web service classes extend from the same base class (WSComponent). It provides the actual service cache layer functionality.

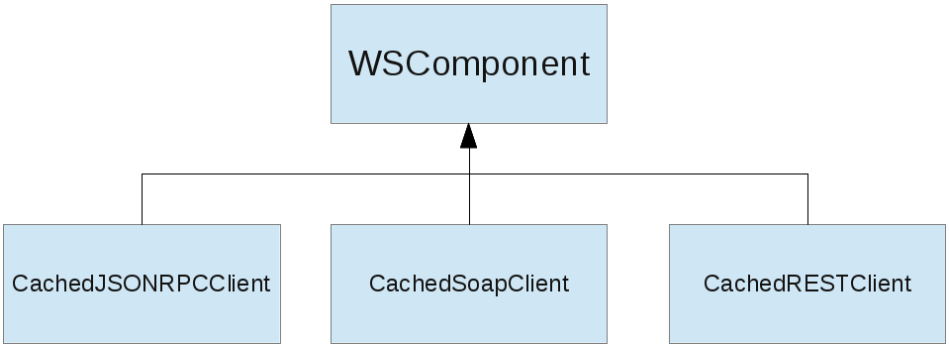

Figure 3: Sub-service cache layer structure

3.1 setCachedResponse

The actual caching functions are provided by the common WSComponent base class. It provides simple methods to check for a previously cached response and to save a new response.

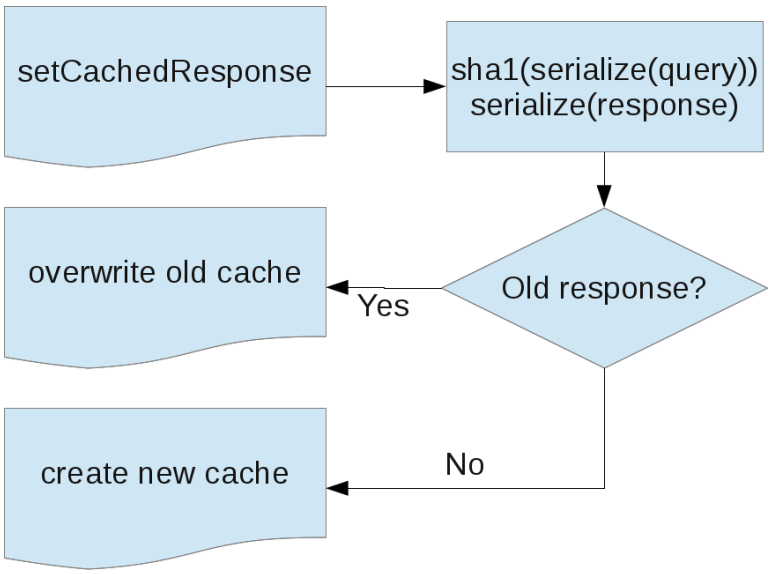

Figure 4: Response caching

As outlined in Figure 44, responses are cached in serialized form. This is required due to different data structures of the responses (e.g. array, objects, etc.) which cannot be handled by the database directly, but work flawlessly in a serialized form.

The query is stored as the sha1 hash of the serialized query. The hash is used to speed up searches for the query when looking up a cached response. At this point no collisions were found for the sha1 hash algorithm, which means all queries yield a unique hash for lookup.

3.2 getCachedResponse

Receiving a previously cached response works in a similar way to storing it. Figure 55 shows the processing on an incoming cache lookup request. The query is again serialized and the sha1 hash is calculated. It is then used to look up a previously stored response. If none is found, a NULL value is returned. If an entry is found, it is checked against the timeout property (which is specified on a sub-service basis). If the cache entry is outdated, a NULL value is returned. If not, the cached response is unserialized and returned.

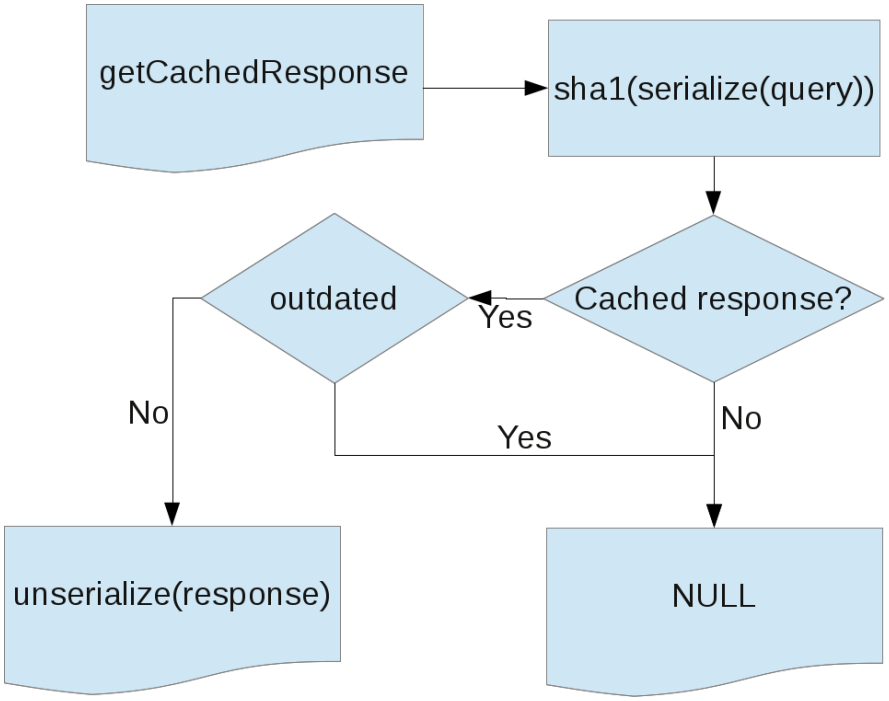

Figure 5: Receiving previously cached response

This approach is completely transparent for all underlying service classes, thus a lookup in the cache and an actual call of the web service yield exactly the same results.

4 LIST OF FIGURES

5 LIST OF REFERENCES

Yii Framework, http://www.yiiframework.com/

GNA nameParser, https://rubygems.org/gems/biodiversity19