MediaWiki

Experience report

TU Wien
184.159 Software Architecture, 2014W

Group 19
Cismasiu Anca (0727280)
Gamerith Stefan (0925081)
Krapfenbauer Klaus (0926457)
Reithuber Manuel (0725031)
Schreiber Thomas (1054647)

Table of Contents

Our approach as a team

In order to decide which open source project to choose, we had two group meetings. In the first meeting, we reviewed each project based on age, the programming languages used, whether or not it is still actively maintained and how widespread its use is. After excluding the unappealing ones based on this criteria, we postponed the final decision for a week, in which each team member had time to get better acquainted with the remaining projects, and to cast his/her vote. Based on this democratic process, we established MediaWiki as our project of choice.

With the project chosen, we had team meetings where we defined how we would handle the documentation. Internally, we gathered an overview of MediaWiki together and started out with the Component & Connector view all together. We read through the documentation and followed debug traces in the source code for the most important use case scenarios, using Google Drive as a collaborative tool to synchronize.

Then we split up our team in two groups: one for improving and refining the still coarser grained C&C view, and the other to draw the activity diagram. Since we started out together on the C&C view, we all had the same base understanding about MediaWiki’s architecture and therefore achieved consistency in the different views. An advantage of doing most of the work face-to-face was that communication and collaboration was easy. It was easy for a team member to share any new insight with the rest.

We chose Visual Paradigm as the modeling tool. Although none of us had previous experience with it, we found it to be best suited for our needs after a short evaluation, because it doesn’t require any other software, has cross-platform support and is actively maintained.

After both teams completed their individual sub-processes, we once again joined our workforce to create the documentation. For that, each group member was assigned some components and connectors from the C&C view for which it had to provide a short description along with the sources and how the component/connector is connected to other components/connectors in the architecture.

The recovery process

The recovery process in terms of information gathering was not too hard because MediaWiki is very well documented. It uses a dedicated documentation wiki[1], and the source code docs are automatically generated using doxygen[2]. Inline comments in the source code explain the important steps that may not be obvious. The comments are precise and easy to understand (aside from some leftover TODOs). If workarounds or unintuitive implementation decisions are present, they are mostly documented. For example, in the /includes folder there is a bash script named ‘limit.sh’. We, of course, wondered why there is a bash script in a platform-independent software, but as soon as we opened the file, we were greeted by a comment “Why is this a shell script?” as well as the explanation of the actual reason[3].

To identify the main components for our C&C view, we started out by reading the chapter provided in the book, as well as the corresponding MediaWiki article[4]. As the chapter seems to have been provided by the MediaWiki community itself, it focused on the essentials and was pretty accurate. It was also interesting that the chapter gave us a short historical overview on the origins of MediaWiki and the impact it had on the resulting code. The problematic thing about this chapter was that it did not fully match the architecture of the current version, since it was written in 2011. In these three years of development, six new versions have been released, which, of course, impacted the architecture to some degree. This became clear while modeling the activity view, as some steps referenced by the book chapter simply do not exist or are not used anymore in the current version. To find these changes, we conducted extensive research to extract the current process.

Identifying components and connectors

As mentioned above, we started by reading the chapter in the book and trying to map the components to corresponding files in the code. To identify further components, we looked in the /includes-Folder. We found that many of these are main modules, logically grouped into folders which directly translate to components in our C&C view (e.g. the parser folder contains most files affiliated with the parser component, the database folder is made out of files with database related logic). Other files placed directly in the /include-Folder also form components or at least conduct important tasks for the framework (e.g. Setup.php forms the Installer component).

The problem with this approach was that we ended up with a very large amount of components of very different granularity and relevance. We decided to aggregate the

[1] http://www.mediawiki.org/wiki/MediaWiki
[2] https://doc.wikimedia.org/mediawiki-core/master/php/html/
[3] /include/limit.sh
[4] http://www.mediawiki.org/wiki/Manual:MediaWiki_architecture

smaller ones into larger logical components and to discard the implicit or obvious ones (that are not especially relevant for the core system or are not worth mentioning, as they are present in almost every system), like the User component, since it just provides login/logout mechanisms and some DAOs and therefore only plays a minor role in the big picture or the Autoloader component, as it is just a class loader.

Some components and the connectors were highly debated internally which caused the initial draft of the C&C view to be revised quite often and spawned some hour-long discussions both online and face-to-face. What made recovery difficult, to some extent, is that MediaWiki has a relatively shallow inheritance structure[5] and many components are loosely connected by global functions, global variables or indirect method invocations. After all these discussions we are now quite certain that our C&C view models the actual architecture of MediaWiki quite well and is able to convey this basic information to new developers.

Since MediaWiki is a large project, we know that our C&C view is far from complete. But as mentioned above, we are confident that we got the main components, the main connectors and the main connections right.

Identifying elements for the activity view

For the activity view, we wanted to cover the two most important scenarios of MediaWiki: Viewing an article and editing an article.

As with the C&C view, we started out from the information supplied in the book[6]. Since the chapter only gave us a very rough outline, we had to gather further information ourselves. Our main source was the code itself. We installed MediaWiki on our own server, configured it and closely inspected the artifacts generated, mainly the configuration files and the database. For identifying the internal activities of MediaWiki when viewing an article, we set up a debugger (in our case: xdebug[7]).

We set a breakpoint for our entry point (index.php), requested the view of an article and followed the source step-by-step. For every step, we tried to interpret the meaning of the executed code and documented the main activities that we identified. After the first run, we aggregated our documentation into main activities and main decisions, which we then put into a Visual Paradigm diagram. Afterwards, in a second walkthrough, we identified all activities and decisions in the source code (again) and created the mapping for the documentation.

The main differences to the book chapter are that we included the ‘edit article’ scenario which was not covered in the original chapter. In addition our view is much more fine-grained than the description in the book. Since we chose performance as one quality, which we inspect in greater detail, there are many elements that contribute to performance: We see

[5] https://doc.wikimedia.org/mediawiki-core/master/php/html/inherits.html

[6] http://www.mediawiki.org/wiki/Manual:MediaWiki_architecture#Execution_workflow_of_a_web_request

[7] http://xdebug.org/

caches as an integral part of WikiMedia (since it is arguably very important for Wikipedia[8]), this is also represented in our views, especially in the activity view, where some of the used caches as well as the load balancer are specifically addressed.

Creation of the mappings

As the mapping for the activity view was created by setting breakpoints and executing the source code step-by-step, we are confident that the view-to-source mapping for the activity view is both correct and complete for the current version of MediaWiki and the chosen activity scenario.

As the source code was also a heavily used information source for the recovery of the architecture for the C&C view we are certain that the mapping for this is also correct. In the mapping process we tried to assign source code files and/or blocks to components, usually by executing and inspecting it through the debugger trace or sometimes just by digging into the documentation provided for each function or class.

As we did not illustrate each and every connection between components in the C&C-view, we know that our mapping is far from complete. Because MediaWiki’s components are mostly loosely coupled via global function calls, global variables or their own classloader, they all have many connections to each other. Of course we tried to capture as many of these connections as possible (e.g. by searching for invocations of a component’s method in the remaining source via string comparison or the ‘find usages’ method provided by our IDEs), but many of these connections are only made in edge cases (e.g. during maintenance) so we decided to not include them in our view for simplicity’s sake. Therefore, our architecture-to-source mapping for the C&C view should be correct, but not complete.

Choosing and describing the qualities

When choosing the two qualities that we want to describe in greater detail, we quickly and democratically settled on modifiability and performance.

Performance was selected because it is a main focus of MediaWiki and therefore, information on this topic is abundant, from tutorials on how to set up the best cache[9], to documentation about code rewrites that have been done to improve performance[10]. In the codebase of the current version, a simple string search for ‘cache’ yields 10.700 results, while the search for ‘article’ yields 10.500 results although MediaWiki’s main purpose is the management and display of articles. While this non-scientific method does, of course, not prove anything, at least it gives a feeling for the emphasis which is put on performance in MediaWiki’s development.

The second quality that we chose is modifiability. Since MediaWiki runs thousands of independent installations worldwide, each with its own requirements, it needs to be easily configurable and modifiable, to meet the needs of each concrete installation. Also, because

[8] http://www.mediawiki.org/wiki/Manual:Cache
[9] https://www.mediawiki.org/wiki/Manual:Performance_tuning
[10] https://upload.wikimedia.org/wikipedia/commons/b/bb/ResourceLoader_LCA2012.pdf

there is a demand for instructions on how to modify the installation (e.g. by adding skins, extensions, ec.), there is good documentation available[11] and the hooks where extensions add functionality can be found all throughout the source.

Use of our classes taken in Software Architecture

The lectures in Software Architecture that we attended, as well as studying for the final exam at the end of November proved to be useful in some regards. For one, we were able to quickly grasp the concept of components and connectors, as well as the approach of treating connectors as first-class citizens. Without the lecture, we probably would have inclined to just mistake packages or modules for components.

Other than that, the main effort of the assignment was clearly digging through the large amount of source code and documentation that a project like MediaWiki's possesses and not losing the overview while recovering the architecture from there.

The MediaWiki community

For our assignment, we did not reach out to the MediaWiki community. As the provided documentation was quite accurate, there was no need to contact the developers for further clarification.

As the provided documentation is also quite current, there was no need for us to update it. Only the page featuring the chapter from the book is outdated. We did not update it yet, but are considering to do so once we finished this assignment and our presentation. MediaWiki's community seems to be very open-minded and welcoming for new developers, also they use their own product for communication, so updating their documentation is as easy as editing a Wikipedia article.

[11] https://www.mediawiki.org/wiki/Manual:Skinning