GraffDok — A Graffiti Documentation Application

Robin Holler

Potsdam University, Potsdam, D-14482, Germany
robin@wxyz.de

Abstract. GraffDok is an application helping to maintain an overview over sprayed images somewhere in a city. At the time of writing it aims at vandalism rather than at beautiful photographic graffiti in an underpass. Looking at hundreds of tags and scribbles on monuments, house walls, etc. it would be interesting to not only record them in writing but even make them accessible electronically, including images.

GraffDok's workflow is simple and only requires an EXIF-GPS-tagged photograph of a graffito. It automatically determines its location by using reverse geocoding with the given GPS-coordinates and the Gisgraphy WebService. While asking the user for some more meta data, GraffDok analyses the image in parallel with this and tries to detect fore- and background – before extracting the drawing lines and make them stand alone. The command line based tool ImageMagick is used here as well as for accessing EXIF data.

Any meta data is written to csv-files, which will stay easily accessible and can be integrated in TeX-files as well. The latter ones are converted to PDF at the end of the workflow, containing a table about all graffiti and a summary for each – including the generated characteristic graffiti pattern image.

Keywords: data acquisition, documentation, image analysis, image processing, geocoding, automation, automatization, graffiti, photography, LaTeX, meta data, ImageMagick, Gisgraphy.

1 Introduction: Workflow Scenario

GraffDok assumes there are lots of people out there carrying new smart mobile phones. They all have the possibility to capture their environment in photographs and upload them immediately to the www. So there's a nice big potential for crowd sourcing – not only for creating maps like OSM but for documentation too. At the moment GraffDok is still evolving and is working locally only but it is aiming at a large world wide database of graffiti.

First of all GraffDok is dealing with kinds of vandalism. But later on the beautiful graffiti will follow. That's the reason why the neutral word "documentation" lies within GraffDok's name and description: GraffDok can be used to create stats and even follow/locate a sprayer until he is arrested, but it does not

A.-L. Lamprecht et al. (Eds.): Process Design for Natural Scientists, CCIS 500, pp. 239–251, 2014.
DOI: 10.1007/978-3-662-45006-2_19

need to. There are some beautiful thoughts and purposes, e. g. a rating system for photogenic graffiti world wide.

In a wise foresight there will never be a big database with contributions from people all over the world until there is a quick and easy way to contribute. That's why this story starts with jABC: It may sound simple to stick picture and description together, creating a profile. But it requires great efforts to make this efficient, useful and thus sustainable. Any data must be complete and accurate. The images should be edited and saved as clean or even model like patterns – out of various photographs taken by mobile phone cameras! Only if all works out, it will be possible to search the database for graffiti by text and even by image.

Putting submission things in a standardized workflow seems therefor a good way to achieve entries of high quality only. It's always a good idea to take some time first, make some decisions about how to process the images best and then implement these thoughts in automatized processes. That way noone has to sit down and process all the images in editing software by sight. And yet there will be good results without significant deviations. There may be batch processing for images, but image processing is just one element of the process chain. With jABC one has not even to start diverse batch processing jobs. Just follow the entire workflow once it is created.

In future GraffDok will have to deal with creating a digital fingerprint of each graphic/pattern. Finding a way to express its characteristics as handy useful code, serving as findable marker in the database is not trivial, but necessary for searching by image. As a start GraffDok will concentrate on clean user submissions and image preprocessing. Establishing a large database with all that basic input will provide a solid foundation for further developments.

2 Service Analysis

GraffDok wants to add some readable and understandable geo information. That said it has to transform the coordinates into an address. The coordinates are given by the EXIF data attached to a (JPEG-)photograph out of a smart phone with GPS. So there's need for a (reverse) Geocoding Webservice. And there's a need for a tool able to read EXIF data and make them accessible, too.

There are many Webservices for geocoding. Google Maps can be used, for example. It's a simple service and most often it is as easy as creating a schematic URL containing latitude and longitude:

```
http://maps.googleapis.com/maps/api/geocode/xml?latlng=lat,long
```

would return an XML containing a lot of information to this point and local area. Most Geocoding Webservices are free to use and delivered valuable results while testing within jABC. In the end Gisgraphy [17] was implemented in the workflow as Geocoding WebService. Gisgraphy is not only free but based on an open source framework and offers its services via Java or REST API.

There are also a lot of tools able to read EXIF. There even is one, called exif [1]. But since we need an image processing tool as well: ImageMagick [3] is a powerful tool in terms of image processing and is not only able to process the images but will fetch all necessary EXIF data from an image, too. ImageMagick is most often used as command line tool without an graphical interface and is licensed under Apache 2.0.

Many graffiti images typically contain two classes: the surface in the background and the graffito itself in the foreground. In these cases the image analysis can be done by a nice bash script called fuzzythresh from Fred Weinhaus [26]. It is free for use in free apps as well as ImageMagick is. And it's fine for images with fore- and background typically revealing a bimodal shaped histogram.

Since GraffDok saves any meta data as csv-text-files and integrates these files in TeX-files, there's the need of an installed TeX distribution with enhancement by the LaTeX package csvsimple(.sty) [24]. Then it will be possible to import csv-files into TeX and finally form tables in PDF by running pdflatex on command line.

After all there shouldn't be any restrictions since all services are either free/libre software or free to use non-commercially. Moreover all important services should work independently from the platform in principal. But since GraffDok has evolved under an unix-like system (Ubuntu Linux) and did not use jETI yet, it may be possible that some code/paths have to be adjusted, e.g. to control some command line tools like ImageMagick by ExecuteCommand-SIBs. There is even little use of unix-tools like sed and cat, but it shouldn't be difficult to find an adequate replacement for Win and others.

3 Workflow Realization

The entire workflow is visible in figure 1: the four essential parts are framed by rounded rectangles.

The arrangement in the part on top shows when files are chosen and paths are defined, as well as where file names are extracted. There is a lot of interaction with the user since the program needs to get input first, most importantly a photograph of a graffito. As a result mainly common GUI SIBs are used here, except for one row of common basic SIBs dealing with strings and extraction of pattern.

So the first thing GraffDok does – after welcoming the user, of course – is to ask him for choosing the photograph. It should be a graffito detail, meaning a close framing without environment and concentrating on the motif: only the pattern on a certain surface. There is an option for a second wide angle image showing the area around the graffito. This second wide angle image will only be resized in further processing and could be added to the final graffito profile.

To keep it simple, GraffDok will make a suggestion for the working directory. By default the working directory is just derived from the selected image file path. Following the suggestion it will be build one level above. In other words: The image folder containing the original photograph would already lay inside

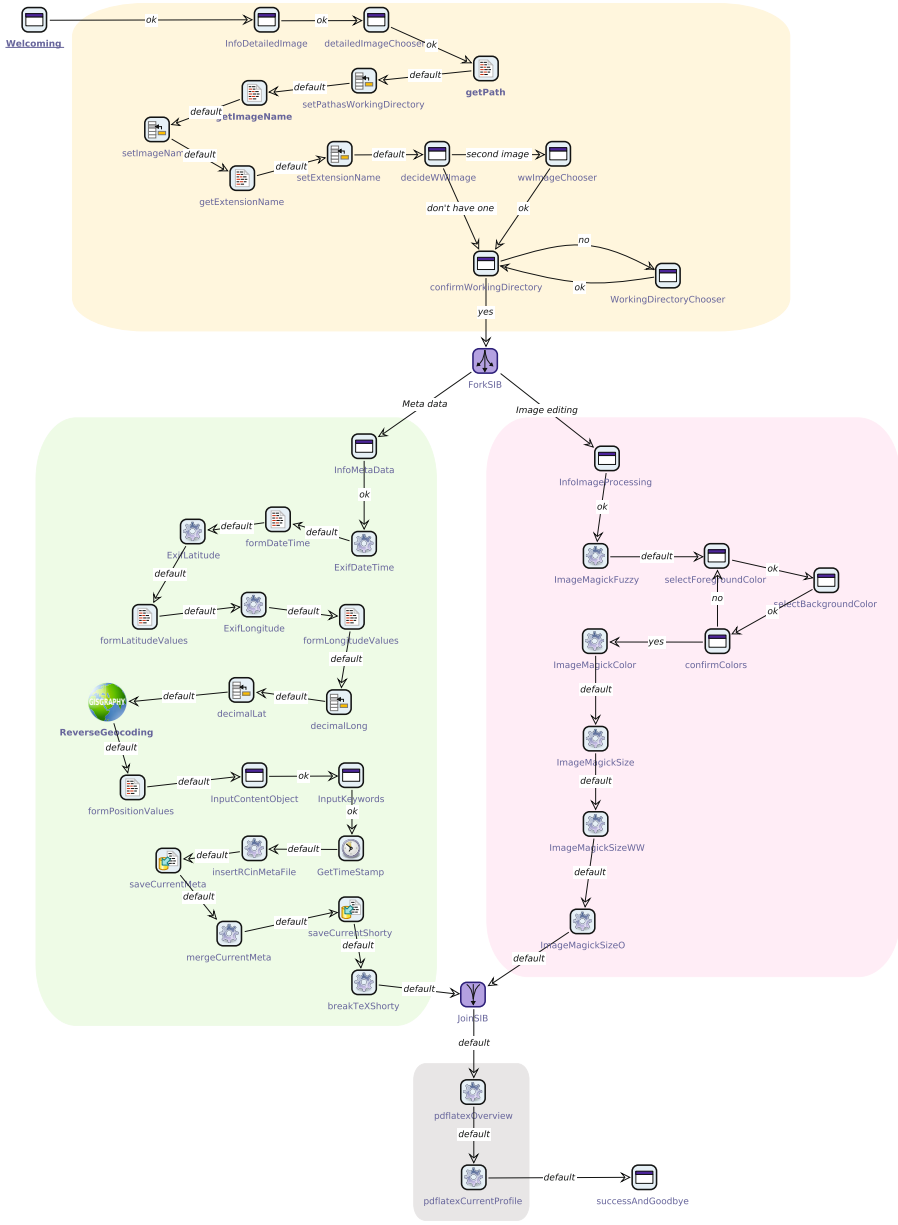

Fig. 1. The entire workflow devided in its four essential parts. Part on top: Choosing files, defining paths, file names, working directory. Middle left: Processing meta data and generating csv-/tex-files. Middle right: Processing images. Bottom part: Generating PDF documents (overview and current profile) including processed meta data as well as processed images.

the suggested working directory. Ideally the image folder is named "original" or similar since it will also serve as target folder for all further graffiti photographs.

But it's a user's decision to follow the suggestion. To make things work there have to be created two more folders inside the working directory. By default both should be created aside to the folder with original photographs. One called "Imagicks" will contain all processed images, the other one "MetaPDF" will save text-files and PDFs. Right in the working directory lies the fuzzythresh script which may create temporary files in there while working and iterating.

So the user implicitly gives the path of the suggested working directory by selecting the image of choice via file chooser. Technically a regular expression is used here to get the working directory out of the file path:

```
.*(?=/[\w]*/)
```

That expression takes any characters of a given file path until there is only one level of directory left. The last slash in a file path usually devides path from file name. So the latter one as well as the folder containing it, will both be cut off due to the positive lookbehind in the expression (lookbehind: second last slash, certain characters, last slash).

Example: From a chosen file path

```
~/jabc/GraffDok/original/IM0001.jpg
```

the working directory

```
~/jabc/GraffDok
```

would be derived.

That's what the common basic SIBs in the part on top of the workflow are for. In this case they are all using different but short and simple regular expressions, though it would be possible to use one complex regular expressions for all:

```
^(.*/)?(?:$|(.+?)(?:(\.[^.]*$)|$))
```

By using this expression for unix paths, one is able to address the target group by an index. The expression groups the string into the whole file path, the directory, the filename and finally a dot with extension.

By the way: the FileChooser is set to only show JPEG and TIFF in its dialog, assuming that these image formats are common and may hold the required EXIF data with the GPS coordinates. The limited choice is nice for finding files more quickly but can always be modified of course.

The middle left and right part will be processed in parallel. While the one on the right is all about image processing – consuming hardware power and time – the one on the left is processing meta data. This way the user is able to enter his descriptions while images are processed in the background.

As said the part on the right visualizes the image processing. The only interaction with the user is about which colors should be used in the processed image. Beside this interaction the acting of fuzzythresh and ImageMagick is controlled and initiated by ExecuteCommand-SIBs on command line.

Figure 2 shows, what a usual graffito photograph out of a mobile phone could look like. Of course the quality isn't splendid and without positive exposure correction most photographs will come out rather dark due to rather bright background surfaces in most cases, and the cam trying to expose it to a mid grey. But it should be sufficient to filter the pattern and its graphical characteristics.

Fig. 2. A sample photograph of a red harlekin face in Goettingen, Germany

With regard of the already addressed time to plan and create a good workflow, this is an important point. How to deal with various images differing in exposure/brightness, color, etc. without looking at each? There are questions on how to unify/preprocess and how to classify the images, to make the graffito class left.

In figure 2 a human eye has no problems with filtering the graffiti lines. But yet it could be visibly improved. And more important: It should be improved and made even clearer to make it contain only relevant information on the graffito itself. Then it will be easier or even possible to create a digital fingerprint – whatever it will look like in future.

In its current status GraffDok concentrates on emphasizing the characteristic of a pattern by using thresholds. ImageMagick offers many enhancements like simple global affecting filters for contrast and brightness or histogram stretching. Since the histograms of each color channel may be stretched differently the

questions comes up, how important accurate colors are. A linear-stretch should avoid heavy color shifts. The command line would look like:

```
convert SDIM001.jpg -linear-stretch 1x1% linearStr.jpg
```

and as a result the pattern is visible more clearly. But first it's an overall global action on the image and second there are too many useless background information left. So one might try manual leveling to cut off this redundant information. To avoid hard edges one may may use sigmoidal contrast, too. The command lines would look like:

```
convert SDIM001.jpg -level 25%,40% leveled.jpg
convert SDIM001.jpg -sigmoidal-contrast 18%,30% sigmoidal.jpg
```

The results are quite promising and much better for extraction of this graffito pattern. But: the values to achieve these results have been found while trying and looking at the picture. In a common workflow with many diverse pictures this wouldn't be possible. The same intense value settings with such strong thresholding would make some other graffiti certainly disappear.

That's why there is only one choice: An analysis of the image first and a processing accordingly to the analysis afterwards. Fuzzythresh looks at the histogram and its typical bimodal shape due to fore- and background – before thresholding it to a binary black and white image. The latter one afterwards shows the foreground only in black while the rest is white (or inverse). One may read the description of the script author [26].

To make fuzzythresh run on a command line, type

```
. ./fuzzythresh SDIM001.jpg SDIM_fuzzy.jpg
```

Due to the calculated (iteratively and a bit slow) threshold at 34.8709% for this image, the result is fine. There's only the pattern left, but it is black. Since the pattern itself seems a lot more important than the lost color this should be okay. Sprayers will probably also switch colors, rather than using the same color for a certain tag all the time.

Since there is a possibility to assign colors to the resulting binary image in ImageMagick, this may compensate the loss of original color. It even may be nicer because of dealing with a smaller and more exact color pool in the database. Also, the original image and its colors always remain accessible through the graffito profile in the database.

Back to the workflow, this is how images will be processed: first step is the fuzzy script with image analysis, second is a user interaction about the coloring of the generated black and white image. Three common gui SIBs handle this interaction, two as input SIBs for entering fore- and background color and one to confirm the selection.

On command line ImageMagick simply needs fore- and background color as parameters (fg/hg):

```
convert graffito_bw.gif +level-colors fg,hg graffito_color.gif
```

Now sizing is the only thing left in image processing. Assuming that the final profile document will have fixed spaces of 500 px in width and height for its images, resizing will be done by reducing to 500 x 500 px after the longest side and filling white color where necessary:

```
convert fuzzy.jpg -resize 500x500 -size 500x500 xc:white
+swap -gravity center  -composite fuzzy_sizedwhite_500.jpg
```

With finishing the middle right part of the workflow there will be up to five new processed images. Both originals (detail and wide angle) will have copies in decreased size of 500 x 500 px. And the more important detail image is available in three more versions as clean binary pattern, as image in clean adapted colors and as final image in reduced size. Figure 3 shows the original images in /original/ while figure 4 shows the processed images in /Imagicks/.

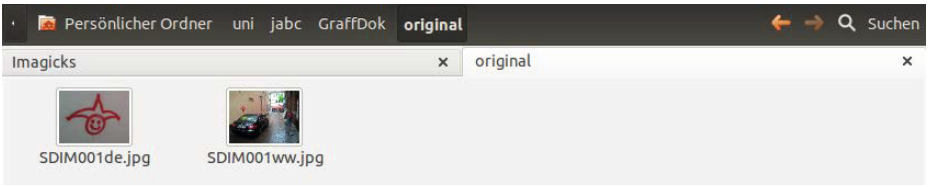

Fig. 3. One detailed and one wide angle photograph as originals

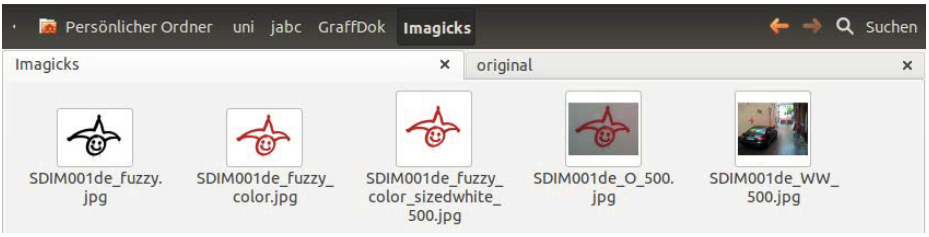

Fig. 4. Five images processed from two original images – ready to be used in the profile

The middle left part of the workflow is dealing with the meta data of the captured graffito. As already stated these processes will be executed in parallel with the image processing of the middle right part.

First GraffDok uses ImageMagick on command line to read specific EXIF data from the input image, e.g. date and latitude/longitude. Since there most often have to be made adjustments in each returned value, there are ExtractPattern-SIBs in between the ExecuteCommand-SIBs. Afterwards two PutExpression-SIBs help transforming the notation of the coordinates from ddd mm' ss,ss"

into ddd,dddddd for each: latitude and longitude. The reason for calculating to decimal values is the Webservice Gisgraphy, accepting only decimal values in the request query.

Since Gisgraphy not only returns few values but a list containing addresses in XML ordered by distance, one have to restrict the result to the one closest at the given coordinates. To extract all relevant information again an ExtractPattern-SIB is used.

To this point the middle left part of the workflow is processed really fast and without interaction to the user. However the following two InputDialog-SIBs will ask for the type/content of the graffito and the kind of object it was sprayed on. Then the user is also prompted to enter few keywords. Until no digital fingerprint works, these keywords are important to make the database searchable.

Since there are no hardware power consuming tasks to do in this part of the workflow, the submission of text by the user takes some time which is used to finish image processing in parallel.

All current processed and entered meta data is written in a simple comma-separated text file named currentMeta.csv. Each graffito and therefore each detailed photograph will be evaluated in this way. Each time all information is stored in a new line of a collecting csv-file named allMeta.csv.

ExecuteCommand is used to do some operations on text files. For example a carriage return is added to allMeta.csv. This will ensure writing in a new line with the current meta data. This is important because comma-separated text files won't work otherwise. So there are WriteTextFile-SIBs creating and writing current meta data in a csv-text-file and there are ExecuteCommand-SIBs for merging current data to allMeta.csv and to add line breaks.

The file allMeta.csv lays inside /MetaPDF/ and since it has only its headline and no entries by default, the headline shows what kind of information will be entered:

```
ID, date, content, object, city, stand name, street type, OSM,
 profile, keywords
```

Next to allMeta.csv inside /MetaPDF/ lays a lightweight TeX-file, staying unchanged but integrating the growing allMeta.csv:

```
\documentclass{article}
\usepackage[landscape]{geometry}
\geometry{hmargin=.1cm,vmargin=.4cm}
\usepackage[utf8]{inputenc}
\usepackage{csvsimple}

\begin{document}
\csvautotabular{allMeta.csv}
\end{document}
```

When the left and right part of the workflow are finished and join again, we'll have processed meta data written in text files (csv and tex) on the one hand, and we have processed clean and small images on the other hand. The little framed part to the end covers two final ExecuteCommand-SIBs. All processed text-files as well as the images are available to them now. So they can perform pdflatex on command line. As a result there will be two PDF documents: a table containing all meta data from each processed image (figure 5), and a profile showing meta data of the current image next to its characteristic pattern.

ID	date	content	object	city	stand name	street type	latitude	longitude	keywords
20120810-1523SDIM001	2012:05:05 13:17	face	public monument	Göttingen	Mühlenstraße	PEDESTRIAN	51 32 6 N	9 56 4 E	harlekin red laughing
20120810-1538SDIM001de	2012:05:05 13:17	scripture	private house	Göttingen	Mühlenstraße	PEDESTRIAN	51 32 6 N	9 56 4 E	RASH 60x20cm blue
20120810-1753201208-20	2012:05:05 15:08	fratze	sockel	Göttingen	Wilhelmsplatz	PEDESTRIAN	51 32 1 N	9 56 16 E	harlekin also laughing

Fig. 5. Generated overview-PDF, CVS integrated in TeX through csvsimple

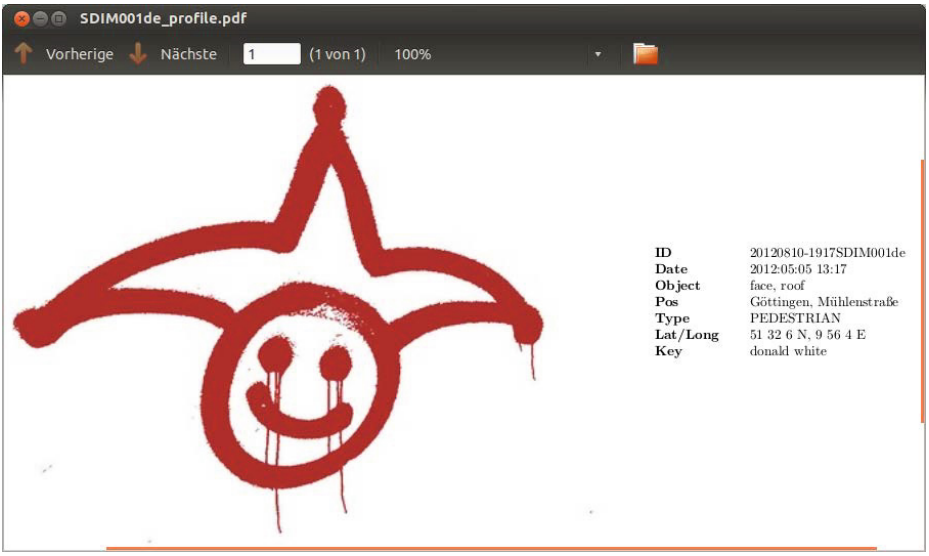

Fig. 6. Generated profile-PDF: graffito pattern next to current meta data

4 Conclusion

There are obvious advantages using GraffDok. The user still has to enter meta data each time he's processing a photograph and there's no way around human input or image description. But the user is guided, and all other things are done

automatically, e. g. a growing table as graffiti overview. And these processes are even done in the time the user needs to enter the description.

So it is not a real one-click-application, but except for the image description it is! And as long as we take simple photographs with distinct graffiti lines on a surface, the graffiti are processed and extracted from their background very well. In some time it should be easy to create nice stats, for example in which area the most graffiti can be found. The choice of collecting data as csv should also be a benefit then, because csv-files are quite easy to handle with GNU R. Also, csv-files are in general easily editable in any editor or tab-based calculation software.

TeX may be a good choice, too. The layout is very simple yet. But TeX offers precise positioning and standardized outputs for more complex profile PDF-documents in future. With the help of csvsimple both, TeX and CSV, are quite usable in combination, while concentrating on a minimum.

Of course there is room for improvement in this first version of GraffDok. First of all there should be a multiple file input in future, allowing multiple descriptions at a time, maybe with thumbnails. At least in case somebody takes a lot of images it is for sure annoying to start from the beginning with every graffito.

Also the image processing only works fine with the most common type of graffiti: distinct lines on homogenous backgrounds. In this simple case thresholding works well enough. But to handle graffiti with different colors and mixed backgrounds, in other words: more classes, it will become necessary to do a more complex image classification. To stay with Open Source and to avoid too many different software, one may take GNU R for the stats as well as for the image classification, e. g. via random forests.

A digital fingerprint is hard to develop but would be greatly useful. It's a pity it will still take some time until one can capture a graffito and GraffDok tells immediately where this pattern was found elsewhere – without crawling databases by own sight or keywords. In terms of image recognition some free and open source frameworks for Augmented Reality may help. Especially those working markerless, recognizing any image without predefined binary markers/codes.

There are few more options to make this project even better. There are ways and tools like tth doing tex2html. By converting all TeX-Profiles into HTML-Websites and converting the overview to a website as well, one could make the database accessible to everyone via web. It would be easy to hyperlink from the overview to a profile or to OSM showing the waypoint.

And the whole bunch of visualizing stats may follow, e. g. spatial distribution. An interface/GUI for editing old entries would be nice, too. And a beautifully visualised comparison of entries – maybe telling, that there already was a similar graffito at a certain place, but maybe was cleaned and sprayed on again.

This article is part of a larger evaluation [7], which aimed at illustrating the power of simplicity-oriented development [14] by validating the claim that process modeling can indeed be handed over to the domain experts by providing them with a graphical modeling framework [23] that covers low-level details in a service-oriented fashion [16], integrates high-level modeling in the overall development process in a way that user-level models become directly executable [15,12], and supports ad-hoc adaptations and evolution [11,13].

The project described in this article can be characterized as follows:

- Scientific domain: geoinformatics
- Number of models: 1
- Number of hierarchy levels: 1
- Total number of SIBs: 46
- SIB libraries used (cf. [10]): common-sibs (45), gisgraphy-sibs (1)
- Service technologies used: REST web services

The geoinformatics part of this volume contains eight other articles on workflow applications in this domain [5,19,4,25,21,20,2,22]. Further geoinformatics workflow projects with the jABC have recently been started. Ongoing work is also exploring how to apply semantics-based (semi-) automatic workflow composition techniques (as provided by, e.g., [18]) to support the workflow design process, as described in [8,9,6] for the bioinformatics domain.

References

1. exif – the simple command line interface to libexif, an open source EXIF library written in C, `http://libexif.sourceforge.net/` (Online; last accessed January 20, 2013)
2. Hibbe, M.: Spotlocator Project Documentation. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 149–158. Springer, Heidelberg (2014)
3. ImageMagick Studio LLC. ImageMagick – open source software for editing images, `http://www.imagemagick.org/` (Online; last accessed January 20, 2013)
4. Kind, J.: Creation of Topographic Maps. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 225–234. Springer, Heidelberg (2014)
5. Kuntzsch, C.: Visualization of Data Transfer Paths. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 140–148. Springer, Heidelberg (2014)
6. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
7. Lamprecht, A.-L., Margaria, T.: Scientific Workflows and XMDD. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 1–13. Springer, Heidelberg (2014)
8. Lamprecht, A.-L., Margaria, T., Steffen, B.: Bio-jETI: A framework for semantics-based service composition. BMC Bioinformatics 10(suppl. 10), S8 (2009)

9. Lamprecht, A.-L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. Journal of Biomedical Semantics 2(suppl. 1), S5 (2011)
10. Lamprecht, A.-L., Wickert, A.: The Course's SIB Libraries. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 30–44. Springer, Heidelberg (2014)
11. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2009)
12. Margaria, T., Steffen, B.: Business Process Modelling in the jABC: The One-Thing-Approach. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling. IGI Global (2009)
13. Margaria, T., Steffen, B.: Continuous Model-Driven Engineering. IEEE Computer 42(10), 106–109 (2009)
14. Margaria, P., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer 43(6), 90–92 (2010)
15. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Process Design for Natural Scientists, pp. 217–236. Springer, London (2012)
16. Margaria, T., Steffen, B., Reitenspiess, M.: Service-Oriented Design: The Roots. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 450–464. Springer, Heidelberg (2005)
17. Masclet, D.: Gisgraphy project – open source framework offering geolocalisation and geocoding services, `http://www.gisgraphy.com/` (Online; last accessed January 20, 2013)
18. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose Programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Engineering. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
19. Noack, F.: CREADED: Coloured-Relief Application for Digital Elevation Data. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 182–195. Springer, Heidelberg (2014)
20. Respondeck, T.: A Workflow for Computing Potential Areas for Wind Turbines. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 196–211. Springer, Heidelberg (2014)
21. Scheele, L.: Location Analysis for Placing Artificial Reefs. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 212–224. Springer, Heidelberg (2014)
22. Sens, H.: Web-Based Map Generalization Tools Put to the Test: a jABC Workflow. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 171–181. Springer, Heidelberg (2014)
23. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007)
24. Sturm, T.F.: csvsimple – open source LaTeX package for processing of files with comma separated values, `http://www.ctan.org/tex-archive/macros/latex/contrib/csvsimple/` (Online; last accessed January 20, 2013)
25. Teske, D.: Geocoder Accuracy Ranking. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 159–170. Springer, Heidelberg (2014)
26. Weinhaus, F.: fuzzythresh script – bash script for automatically thresholding a two class image to binary black and white format using fuzzy c-means, free use in free software, `http://www.fmwconcepts.com/imagemagick/fuzzythresh/` (Online; last accessed January 20, 2013)