Spotlocator – Guess Where the Photo Was Taken!

Marcel Hibbe

Potsdam University, Potsdam, D-14482, Germany
mhibbe@gmail.com

Abstract. Spotlocator is a game wherein people have to guess the spots of where photos were taken. The photos of a defined area for each game are from panoramio.com. They are published at `http://spotlocator.drupalgardens.com` with an ID. Everyone can guess the photo spots by sending a special tweet via Twitter that contains the hashtag #spotlocator, the guessed coordinates and the ID of the photo. An evaluation is published for all tweets. The players are informed about the distance to the real photo spots and the positions are shown on a map.

Keywords: Geoinformation, Spotlocator, panoramio, twitter, game, google static maps, REST.

1 Introduction: Workflow Scenario

The basic idea of this project is a game wherein people have to guess the spots of where photos were taken. It should improve people's geographical knowledge about their neighborhood. The administrator can define the search area of the current game. Thereafter information about geo-referenced photos are downloaded from Panoramio (`www.panoramio.com`) for the specified area. For every photo a unique ID is generated which gets stored in a file containing additional data such as the coordinates and the author. The photos are published on the website `http://spotlocator.drupalgardens.com` together with their IDs. The challenge for the players is to search the photo locations on site. For that they need a Twitter account (`www.twitter.com`), and a smartphone with GPS support. When a player has found a photo location, he sends a tweet in the following format:

#spotlocator 52.123456,13.123456 id:123

The hashtag #spotlocator is the name of the game and is used to identify tweets containing information about the assumed photo locations. The numbers in the middle are geographical coordinates which have to be modified by the player for the current photo. The ID refers to the Photo-ID published on the website. This way, every person having a Twitter account and owning a smartphone can join the game.

For evaluation all the tweets containing the Hashtag #spotlocator are downloaded. The distance to the original photo locations is calculated by comparing

A.-L. Lamprecht et al. (Eds.): Process Design for Natural Scientists, CCIS 500, pp. 149–160, 2014.
DOI: 10.1007/978-3-662-45006-2_12

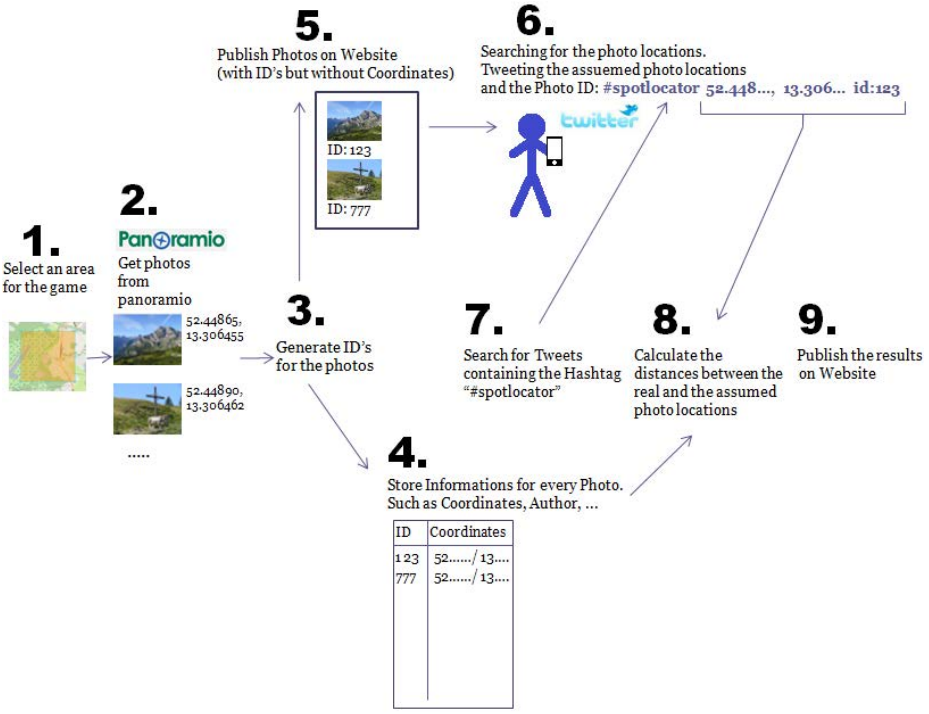

Fig. 1. Workflow of the game

the assumed coordinates with the coordinates stored in the file containing the original photo information. The results are published on the website, so everyone can check how accurate his assumption was. Figure 1 shows the workflow of the game.

The reason for using Panoramio was that there is already a pool of data that can be used and there is no need to create an own data basis. The reason for using Twitter was the wide distribution of the service so that people don't have to register any additional account for playing the game.

2 Service Analysis

The services needed for realizing the workflow is the Panoramio REST API [21] to get the photo Information and the Twitter REST API to download the tweets. Additional tools are Dropbox, which is used to publish the generated HTML sites. The content management system Drupal has been used to design a website where the generated HTML sites are embedded via an iFrame.

A site that uses the Panoramio API can show photos that appear on Panoramio free of charge. The site must credit the photographer and link to the original photo page. You must also display the text "Photos provided by Panoramio

are under the copyright of their owners" under Panoramio-photos. To use, copy, print, or download a photo from Panoramio, you must get explicit permission from the photographer [22]. For this reason the spotlocator program will not have a print-function. To use the REST API in jABC, you have to do a GET on an URL like this:

```
http://www.panoramio.com/map/get_panoramas.php?set=public&from=
0&to=20&minx=-180&miny=-90&maxx=180&maxy=90&size=medium&mapfilter=
true
```

Minx, miny, maxx and maxy define the area to show photos from. "set=public" means that only popular photos are used and "size=medium" specifies the image size. You can define the number of photos to be displayed using "from=X" and "to=Y", where Y-X is the number of photos included. The value 0 represents the latest photo uploaded to Panoramio. For example, "from=0 to=20" will extract a set of the last 20 photos uploaded to Panoramio, "from=20 to=40" the previous set of 20 photos and so on. The result data is formatted using JavaScript Object Notation (JSON) [3]. To do a GET command in jABC, the REST SIB "FetchTextualDataFromURL" is used. It fetches textual content behind the given URL and puts the result into the execution context. To use the Twitter REST API [28] a GET command is used as well and the result data is formatted using JavaScript Object Notation (JSON), too. The URL to search for Tweets with the hashtag #spotlocator is:

```
http://search.twitter.com/search.json?q=40spotlocator
```

As the Twitter search is mainly a real-time search, it's only possible to find Tweets of the last few days (between 4 and 10 days). This is enough to fit the needs of the spotlocator game.

As a result of the jABC workflow there are generated HTML files containing the current game, the results of the guesses and an archive for old games. These files get stored in the public folder of Dropbox, which has to be installed at the local machine. For each HTML file there is a public URL (rightclick - copy public URL). These URLs are used to display these sites via an iframe in a website created using the content management system Drupal Gardens. As creating a website is not a topic of this seminar paper, this documentation will limit to the generation of the HTML files via jABC. The website can be visited at `http://spotlocator.drupalgardens.com`.

All in all, the only requirement to run the program is the desktop version of Dropbox. As the guesses published by the Twitter users refer to the photos on the Website `http://spotlocator.drupalgardens.com` (currently updated by the author), it is not recommended to send tweets referring to photos which have been generated by another machine. This would lead to redundant photo IDs so that the evaluation would show up wrong results. Because of this it is allowed

to run this program on another machine, but tweets should always refer to the photos on the website which is administrated by the author.

3 Workflow Realization

At the beginning of the program there is the possibility to make some configurations. If the program is running on another machine, you can set the paths where the program data will be saved. If you are using Dropbox to publish the generated HTML files, you have to choose the Dropbox public folder. The table depicted in Figure 2 shows all the files used by the program.

File	action	description
Spotlocator_NewGame.htm	write (Dropbox)	HTML file containing the current game
Spotlocator_Archive.htm	read / write (Dropbox)	HTML file containing the old games
Spotlocator_Report.htm	write (Dropbox)	HTML file containing the evaluation for the tweets
Spotlocator_photoinfo.htm	read / write	File containing all photo informations (ID's, coordinates, author, ...)
Spotlocator_current_number.txt	read / write	File containing the current photo-counter

Fig. 2. Files used by the program

After configuration there is an option (ShowBranchingDialog) between creating a new game or evaluating the tweets from Twitter. The following 2 Chapters will describe each option.

3.1 Create a New Game

To create a new game, the user first has to enter four geographical coordinates (ShowInputDialog) to define a bounding box in which the photo spots must be guessed. The site http://www.openstreetmap.org/export is suitable to get the coordinates. In another Input Dialog the user can enter the number of photos that have to be guessed in the specified area. These inputs are stored in variables that are used to build the GET URL for Panoramio:

```
http://www.panoramio.com/map/get_panoramas.
php?set=public&from=0&to=${number_of_photos}&minx=${area_west}&
miny=${area_south}&maxx=${area_east}&maxy=${area_north}&
size=medium&mapfilter=true
```

The GET request is performed using the REST-SIB "FetchTextualDataFromURL". The result containing the JSON-Code is stored in the variable "panoramio_get". For one photo the code looks like the following:

```
{
  "count": 773840,"photos": [
    {
      "photo_id": 532693,
      "photo_title": "Wheatfield in afternoon light",
      "photo_url": "http://www.panoramio.com/photo/532693",
      "photo_file_url":
"http://static2.bareka.com/photos/medium/532693.jpg",
      "longitude": 11.280727,
      "latitude": 59.643198,
      "width": 500,
      "height": 333,
      "upload_date": "22 January 2007",
      "owner_id": 39160,
      "owner_name": "Snemann",
      "owner_url": "http://www.panoramio.com/user/39160",
    }
}
```


The next task is to extract the strings that are needed in the program. This is done for every string with the SIB "ExtractPattern" by using regular expressions [20].
The expression to extract the pattern for the photo file URL is:

```
"photo_file_url":"([\p{Graph}[\p{Space}]&&[^"]]+)
```

This command is searching for the text "photo_file_url":". After this string there is the URL that has to be extracted. "Graph" and "Space" define the characters that are allowed to be extracted. The symbols

```
[^"]
```

mean that the next quote defines the end of the extraction and "+" means that this search is done until there are no more matching patterns. As a result, the ExtractPattern SIB stores all photo file URL in an array. This procedure is also done for latitude, longitude, photo_id and owner_name. At this point the JSON-Code is no longer needed because all information has been extracted into arrays.

The next four SIBs are for creating an archive for old games. If the program is executed for the first time a new file called "Spolocator_Archive.htm" is created without content. The next time the program runs, the previous game ("Spotlocator_NewGame.htm") will always be added to the archive. This is done by the SIBs "ReadTextFile" and "WriteTextFile". The SIB "PutExpression" merges the previous game and the old archive into one variable that becomes the content of the new archive. After that, the photo information "Spotlocator_photoinfo.htm" of earlier games is loaded from hard drive with the "ReadTextFile" SIB. If there is no such file it will be created by the "WriteTextFile" SIB.

The following "PutExpression" SIB is for creating the beginning of the HTML file "Spotlocator_NewGame.htm". To give the players an overview where they have to search for the photo spots, there are 2 Google Maps (Google Static Maps [1]) that show the bounding box in 2 different zoom levels. The links for the maps are created by using the variables that the user defined at the beginning. Additionally, there is the option to open the bounding box in OpenStreetMap.

```
<head>
    <style>
    body{ font-size:13px; font-family:Helvetica; }
    </style>
</head>

<center>
<b>the photos are form the following area:</b>
<br><br>
<img src="http://maps.google.com/maps/api/staticmap?
path=color:0x0000FF80|weight:4|
${area_north},${area_west}|${area_nord},${area_east}|
${area_south},${area_east}|${area_south},${area_west}|
${area_north},${area_west}&size=300x300&zoom=7&sensor=true">
  

<img src="http://maps.google.com/maps/api/staticmap?
path=color:0x0000FF80|weight:5|
${area_north},${area_west}|${area_nord},${area_east}|
${area_south},${area_east}|${area_south},${area_west}|
${area_north},${area_west}&size=500x300&sensor=true">

<br><br>
To view this area in OpenStreetMap
<a href="http://www.openstreetmap.org/?minlon=${area_west}
&minlat=${area_south}&maxlon=${area_east}
&maxlat=${area_north}&box=yes" target="_blank">click here</a>.
<br><hr><br><br>
</center>
```

The next SIB "ReadTextFile" reads the current counter of the photo IDs from the file "Spotlocator_current_number.txt". Storing the ID on the hard drive is necessary, because otherwise the counter would start at 0 again at every program start. If the file with the counter does not exist, it will be created. The following loop show in Figure 3 is a main part of creating a new game. The loop summarizes the information for every photo and adds it to the Information about the other Photos. Then the HTML-part for the current photo is generated which is added to the whole HTML Code. After that the photo ID and the loop number is increased.

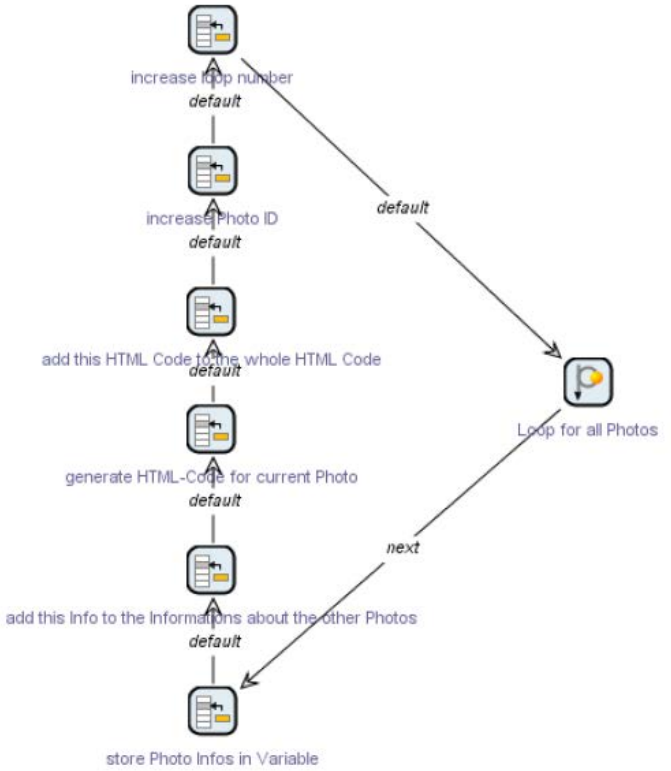

Fig. 3. Loop for summarizing photo information

```
<center>
<h2>photo number: ${photonumber}</h2>
<a href="http://www.panoramio.com/photo/
${panoramio_int_photoid[loopnumber]}"
target="_blank">
<img src="${panoramio_photoadresses[loopnumber]}"></a>

<br><small>Copyright of this photo:
<i>${panoramio_ownername[loopnumber]}</i></small>
<br><br>

You know where this photo was taken?
Send a tweet with the following text:
 #spotlocator <i>latitude</i><b>,</b><i>longitude</i>
 id:<b>${photonumber}</b>
</center>
```

After the loop, the current photo ID and all the photo informations are saved to hard drive. The HTML file "Spotlocator_NewGame.htm" is saved into the Dropbox public folder. The user gets a confirmation that the game was created successfully and additional information about the next steps.

3.2 Game Evaluation

The Game evaluation starts by downloading all tweets with the Hashtag #spotlocator. This is done by the REST SIB "FetchTextualDataFromURL". The URL is: `http://search.twitter.com/search.json?q=spotlocator`.

The answer in JSON-Format is the basis for the ExtractPattern SIBs to extract the desired information. Like at the game creation, this is done by using regular expressions. The extracted information is the Twitter Names of the owners, their nicknames, the coordinates and the photo ID of the photo they refer to. In the next step, the photo information ("Spotlocator_photoinfo.htm") that is stored on the hard drive is read with the "ReadTextFile" SIB. "ExtractPattern" is used again to store the information as arrays. The following "PutExpression" SIB is for creating the beginning of the HTML file "Spotlocator_Report.htm". The loop shown in Figure 4 is to compare photos mentioned in the tweets with the original photo information.

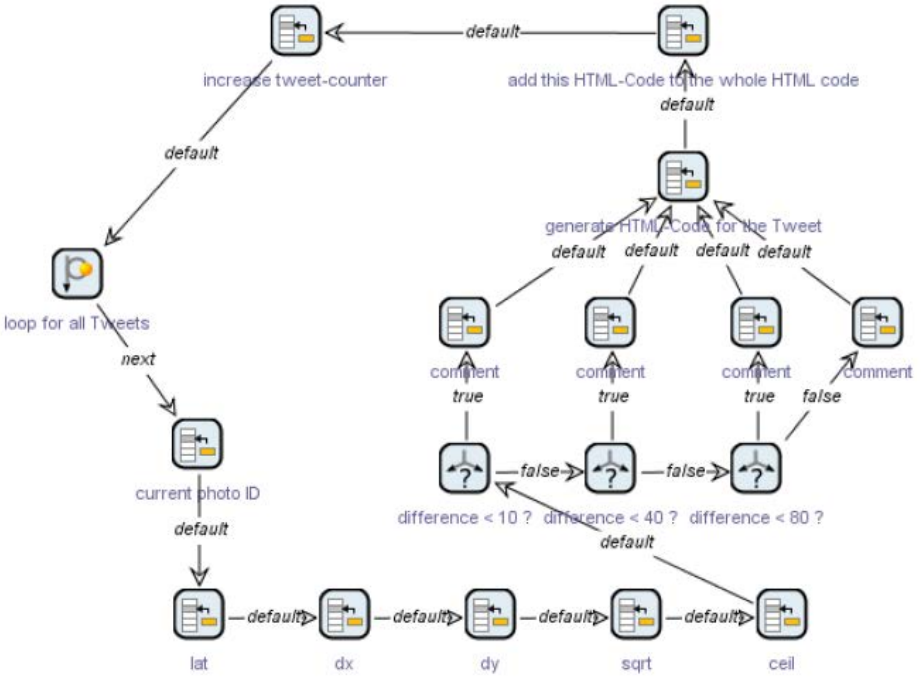

Fig. 4. Loop for photo comparison

At the beginning of the loop, the current photo ID that is mentioned in the tweet is written into a variable with the value $tweets_photoid[tweetnumber]. The next 5 SIBs calculate the distance between the assumed photo locations and the real photo locations. This is realized by using the theorem of Pythagoras while considering that the distance between lines of longitude varies with the latitude. The algorithm for this is [5]:

```
lat = (lat_real + lat_twitter) / 2 * 0.01745
dx = 111.3 * cos(lat) * (lon_real - lon_twitter)
dy = 111.3 * (lat_real - lat_twitter)
distance = sqrt(dx * dx + dy * dy)*1000
```

Lat and lon are the geographical coordinates in degree. Distance has the unit meter. The distance is rounded up with the command "ceil". The next "EvaluateCondition" and "PutExpression" SIBs generate a comment like "very good" or "not so good! good luck for the next time" for the different ranges of difference. Afterwards the HTML part for the current tweet is generated. This is added to the whole HTML Code. After that the loop number is increased. After the loop the HTML file "Spotlocator_Report.htm" is saved into the Dropbox public folder. The user gets a confirmation that the game was evaluated successfully and additional information about the next steps.

4 Conclusion

The workflow could be realized as planned and the game is running without problems. A current inconvenience is that the game creation and evaluation have to be executed manually, so that the website `http://spotlocator.drupalgardens.com` is not always up to date. Ideally, the program would be running continuously on a server so that the information will update in real time. Another constraint is that there can be problems if a Twitter user did not obey to the required tweet format. In future versions this will be improved by changing the regular expressions. At this time, a new game can only be created by the jABC administrator. A better solution for this would have to be realized in future versions, too. An idea is that everyone can create his own game via a special tweet. Furthermore the evaluation could be improved by ranking the best tips for a photo. To build more complex websites a SIB would be useful which is able to post blog entries. Another possibility to interact with the players would be to directly reply to the tweets by tweeting the results. This would require a special SIB as well. All in all, this project could be realized as a jABC workflow without problems. As the results are nearly solely HTML files which have to be published via a web server, PHP would be a good alternative. On the other hand the advantage of jABC is the flexibility to quickly make modifications and to have an overall view of the game.

This article is part of a larger evaluation [8], which aimed at illustrating the power of simplicity-oriented development [15] by validating the claim that process modeling can indeed be handed over to the domain experts by providing them with a graphical modeling framework [26] that covers low-level details in a service-oriented fashion [17], integrates high-level modeling in the overall development process in a way that user-level models become directly executable [16,13], and supports ad-hoc adaptations and evolution [12,14].

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: 79
- SIB libraries used (cf. [11]): common-sibs (18), rest-sibs (2)
- Service technologies used: REST web services

The geoinformatics part of this volume contains eight other articles on workflow applications in this domain [6,19,4,27,2,24,23,25]. 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 [9,10,7] for the bioinformatics domain.

References

1. GOOGLE. Google Static Maps API V2 Developer Guide (last accessed July 24, 2012), `https://developers.google.com/maps/documentation/staticmaps/`
2. Holler, R.: GraffDok — A Graffiti Documentation Application. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 239–251. Springer, Heidelberg (2014)
3. Json.Org. JavaScript Object Notation (last accessed July 24, 2012), `http://www.json.org/`
4. Kind, J.: Creation of topographic maps. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 229–238. Springer, Heidelberg (2014)
5. Kompf, M.: Entfernungsberechnung (last accessed July 24, 2012), `http://www.kompf.de/gps/distcalc.html`
6. 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)
7. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
8. 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)

9. Lamprecht, A.-L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinformatics 10(suppl. 10), S8 (2009)
10. Lamprecht, A.-L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. Journal of Biomedical Semantics 2(suppl. 1), S5 (2011)
11. 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)
12. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2009)
13. 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)
14. Margaria, T., Steffen, B.: Continuous Model-Driven Engineering. IEEE Computer 42(10), 106–109 (2009)
15. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer 43(6), 90–92 (2010)
16. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012)
17. 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)
18. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose Programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
19. Noack, F.: CREADED: Colored-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. ORACLE. Pattern, `http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html` (last accessed August 1, 2012)
21. PANORAMIO. Panoramio API Zeigen Sie Fotos von Panoramio auf Ihrer Webseite an, `http://www.panoramio.com/api/data/api.html` (last accessed July 2, 2012)
22. PANORAMIO. Panoramio Copyright, `http://www.panoramio.com/help/copyright` (last accessed July 23, 2012)
23. Respondek, 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. 200–215. Springer, Heidelberg (2014)
24. Scheele, L.: Location analysis for placing artificial reefs. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 216–228. Springer, Heidelberg (2014)
25. 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. 175–185. Springer, Heidelberg (2014)
26. 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)

27. Teske, D.: Geocoder accuracy ranking. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 161–174. Springer, Heidelberg (2014)
28. TWITTER. Twitter Developers, `https://dev.twitter.com/docs` (last accessed August 1, 2012)