Toolet: Web-based tool appropriation for hobby programmers

Jeremías P. Contell, Oscar Díaz

University of the Basque Country (UPV/EHU), ONEKIN Web Engineering Group,
San Sebastián (Spain)
(jeremias.perez, oscar.diaz)@ehu.es

1 Context & Goals

Technology appropriation is being defined as "the process through which users adopt, adapt, and then incorporate a system with their practices" [1]. This process takes place no matter whether the system is physical or virtual, but it is specially interesting for Web applications. Rationales are twofold. First, opportunity: the Web is becoming the place where an increasing number of applications are being migrated and hence, where users undertaken a larger number of their activities. It can then be expected numerous petitions for adapting Web applications to the users' practice specifics. Second, possibility: Web rendering is "malleable", i.e. client-side code can be transcoded at the browser using augmentation approaches. In addition, two additional practices are paving the way forward: APIs and semantic annotations. APIs account for a programatic way to access complementary functionality that might not be available (or not in the desired way) in the Web counterpart. On the other side, semantic annotations facilitate the understanding and the hackability of the client-side code. This opens an opportunity for Web appropriation.

But this is not enough. Malleability might make Web appropriation possible but not affordable. And the problem is that appropriation is highly idionsincratic. Unlike Web personalization (thought by designers), Web appropriation should be mostly conducted by the applications' users themselves. Indeed, appropriation implies adapting a sytem to the user's practice in ways they might not be conceived by the application designers. Hence, we see a challenge in describing Web appropriation on an adequate level of abstraction. The answer very much depends on both the kind of Web application and the target audience. We focus on *Web tools* and *hobby programmers*. By *Web tools* is meant Web applications that support the manipulation of a software artefact through the Web. Examples of Web tools include Google Docs or editors for different kind of artefacts, e.g. mind maps[1]. As for hobby programmers, they are defined as somebody who spends ten or more hours a month programming, but is not paid primarily to be a programmer[2]. This basically means that development should be of a level of

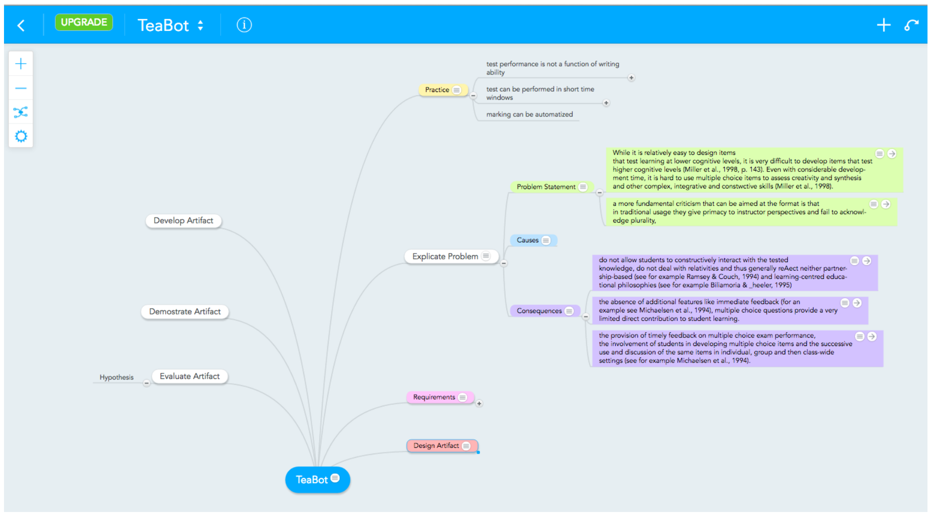
[1] https://www.mindmup.com/

[2] http://www.itworld.com/article/2702038/application-management/hobbyist-programmers–don-t-call-us-hobbyists.html

complexity that fits within this ten-hour frame. Unfortunately, if appropriation of Web tools is conducted in terms of raw JavaScript code, these ten hours will fall short. Raising the abstraction level and providing tool support is a possible way ahead. We explore this way through ***"Toolet"***, an editor for Web tool appropriation built on top of Google's Sheet. Next sections describes work-in-progress with the help of an example: the appropriation of *MindMeister*[3].

2 An appropriation scenario

MindMeister is a popular mind-map drawing tool. *MindMeister* is thought for



test performance is not a function of writing ability

test can be performed in short time windows

marking can be automatized

While it is relatively easy to design items
that test learning at lower cognitive levels, it is very difficult to develop items that test higher cognitive levels (Miller et al., 1998, p. 143). Even with considerable development time, it is hard to use multiple choice items to assess creativity and synthesis and other complex, integrative and constwctive skills (Miller et al., 1998).

a more fundamental criticism that can be aimed at the format is that
in traditional usage they give primacy to instructor perspectives and fail to acknowledge plurality,

do not allow students to constructively interact with the tested
knowledge, do not deal with relativities and thus generally reAect neither partnership-based (see for example Ramsey & Couch, 1994) and learning-centred educational philosophies (see for example Biliamoria & _heeler, 1995)

the absence of additional features like immediate feedback (for an example see Michaelsen et al., 1994), multiple choice questions provide a very limited direct contribution to student learning.

the provision of timely feedback on multiple choice exam performance,
the involvement of students in developing multiple choice items and the successive use and discussion of the same items in individual, group and then class-wide settings (see for example Michaelsen et al., 1994).

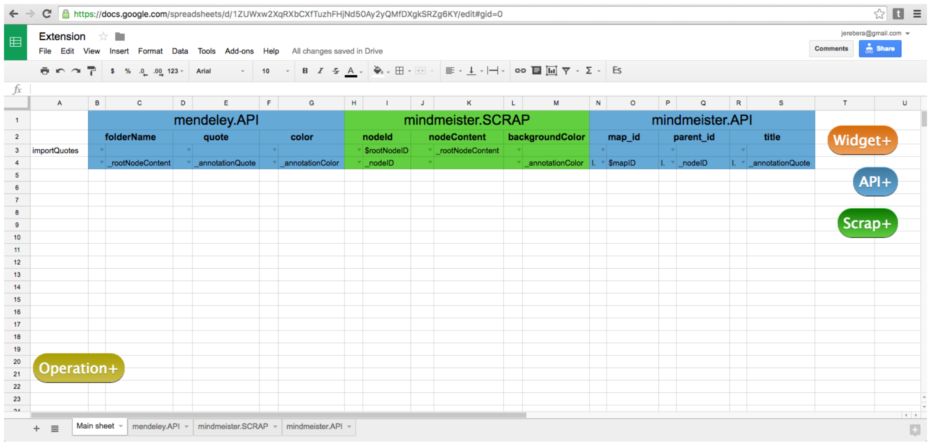
Fig. 1. *A* MindMeister *map: colored leaf nodes are imported from Mendeley document quotes.*

schema drawing and brain storming. At the ONEKIN group, we use intensively *MindMeister* for brain storming but also for conducting PhD projects along Design Science guidelines [2]. This approach guides research projects through different milestones: define the setting, explicate the problem's cause, analyze the problem's consequences, etc. Students can add new nodes to their maps as they get acknowledgeable about the different issues. This is the intended use of mind maps in general, and *MindMeister* in particular. However, we realize that quite often nodes originated not from our own insights but by quoting articles. It goes without saying that research projects start with a thorough literature review, where problems, their causes and consequences might already be well

[3] https://www.mindmeister.com/

documented. The point to be notice is that "nodes" might not always originate while drawing the mind map (at *MindMeister*) but when reading articles (e.g. at *Mendeley*). Node creation at places other than the map editor might not be anticipated by *MindMeister* designers but it turns out of importance to our way of working. What we wanted was the possibility to create nodes for *MindMeister* maps but at places other than *MindMeister* (e.g. *Mendeley*). Specifically, we wanted to highlight problems, causes and consequences when reading articles in Mendeley, and eventually, move to the *MindMeister* map, and find the underlined quotes as nodes hanging from the right places, i.e. the *Problem Statement* node, the *Causes* node or the *Consequences* node (see Figure 1). The relationship between a quote and what this quote stands for (i.e. a cause, a consequence, etc) is set in terms of colors: e.g. a quote in green accounts for the concept node with green background. Next, we address this scenario with *Toolet*.

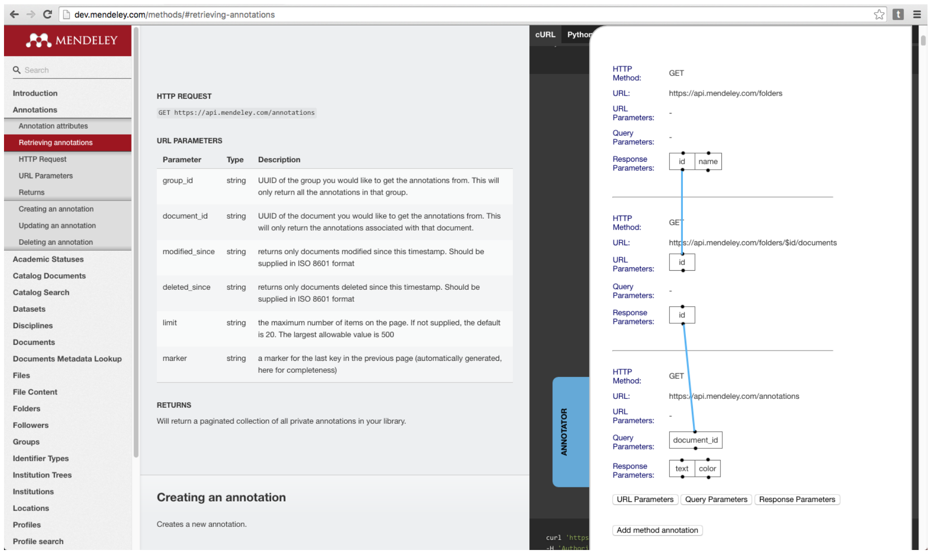
3 Toolet at work


https://docs.google.com/spreadsheets/d/1ZUWxw2XqRXbCXfTuzhFHjNd50Ay2yQMfDXgkSRZg6KY/edit#gid=0
Extension
File Edit View Insert Format Data Tools Add-ons Help
All changes saved in Drive
jerebera@gmail.com
Comments
Share
Widget+
API+
Scrap+
Operation+
Main sheet
mendeley.API
mindmeister.SCRAP
mindmeister.API

	mendeley.API			mindmeister.SCRAP			mindmeister.API		
	folderName	quote	color	nodeId	nodeContent	backgroundColor	map_id	parent_id	title
importQuotes				$rootNodeID	_rootNodeContent				
	_rootNodeContent	_annotationQuote	_annotationColor	_nodeID		_annotationColor	I. $mapID	I. _nodeID	I. _annotationQuote

Fig. 2. Toolet *main view. Tables' signatures are hold in the main sheet whereas each table body is kept into a separated companion sheet.*

Toolet conceives Web tools as a set of *operations* acting upon a set of *tables*. Rather than facing the different ways data can be represented, we strive to hide this heterogeneity through a common tabular representation. In this way, the operational semantics of new *operations* are defined in terms of table manipulations.



HTTP REQUEST

`GET https://api.mendeley.com/annotations`

URL PARAMETERS

Parameter	Type	Description
group_id	string	UUID of the group you would like to get the annotations from. This will only return all the annotations in that group.
document_id	string	UUID of the document you would like to get the annotations from. This will only return the annotations associated with that document.
modified_since	string	returns only documents modified since this timestamp. Should be supplied in ISO 8601 format
deleted_since	string	returns only documents deleted since this timestamp. Should be supplied in ISO 8601 format
limit	string	the maximum number of items on the page. If not supplied, the default is 20. The largest allowable value is 500
marker	string	a marker for the last key in the previous page (automatically generated, here for completeness)

RETURNS

Will return a paginated collection of all private annotations in your library.

Creating an annotation

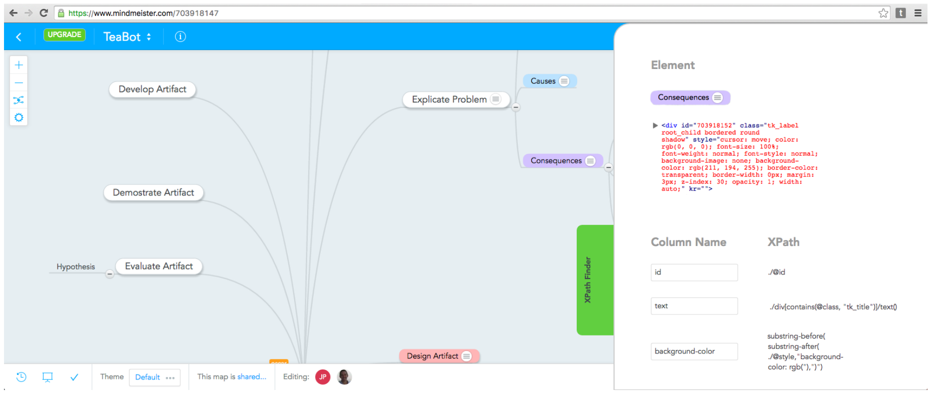
Creates a new annotation.

HTTP Method:	GET
URL:	https://api.mendeley.com/folders
URL Parameters:	-
Query Parameters:	-
Response Parameters:	id name

HTTP Method:	GET
URL:	https://api.mendeley.com/folders/$id/documents
URL Parameters:	id
Query Parameters:	-
Response Parameters:	id

HTTP Method:	GET
URL:	https://api.mendeley.com/annotations
URL Parameters:	-
Query Parameters:	document_id
Response Parameters:	text color

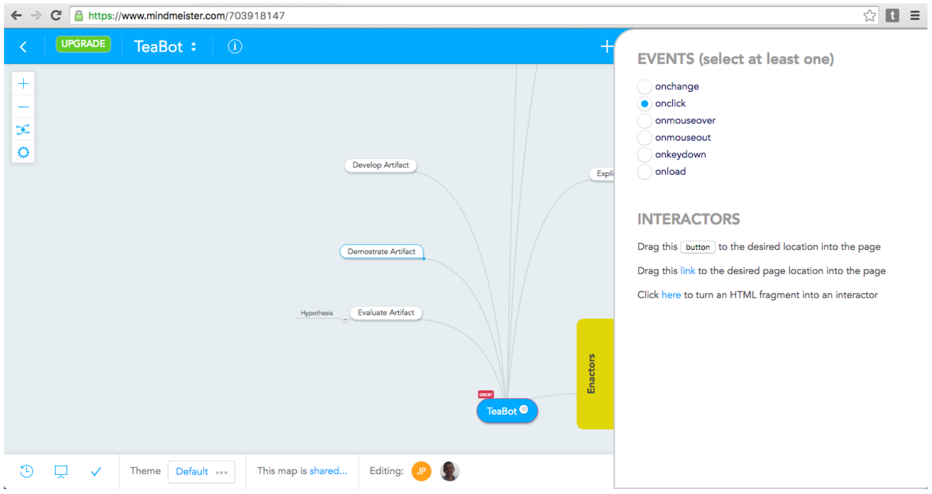
Fig. 3. *API tables: parameters not participating in any pipe become the table's columns. Method chains are obtained by annotating API documentation.*



```
<div id="703918152" class="tk_label
root_child bordered round
shadow" style="cursor: move; color:
rgb(0, 0, 0); font-size: 100%;
font-weight: normal; font-style: normal;
background-image: none; background-
color: rgb(211, 194, 255); border-color:
transparent; border-width: 0px; margin:
3px; z-index: 30; opacity: 1; width:
auto;" kr="">
```

Column Name	XPath
id	./@id
text	./div[contains(@class, "tk_title")]/text()
background-color	substring-before(substring-after(./@style,"background-color: rgb("),")")

Fig. 4. *Scrap tables: data scrapped become the table's columns.*



EVENTS (select at least one)

onchange
[✓ onclick]
onmouseover
onmouseout
onkeydown
onload

INTERACTORS

Drag this button to the desired location into the page

Drag this link to the desired page location into the page

Click here to turn an HTML fragment into an interactor

Fig. 5. *Operation Enactor: name, GUI element, placement and triggering event.*

3.1 Table definition

Table definition is a two-fold process. First, the table signature includes the column names and the type of the table. Second, the table body holds how columns are obtained depending on their type, namely:

- API tables, whose columns are obtained through API calls,
- scrap tables, whose columns are obtained by scrapping the HTML code,
- Widget tables, whose columns stand for the state of Web Components introduced as part of the appropriation.

Toolet is realized as a plug-in for Google's Sheet (see Figure 2). Tables' signatures are hold in the main sheet whereas each table body is kept into a separated companion sheet. A button exists for each kind of table. On clicking, *Toolet* moves to the corresponding mode: creating a table by annotating API documentation, creating a table through scraping the tool to be appropriated, or creating a table that account for a brand-new widget. Table names reflect their origins using a kind of format notation (*origin.type*, e.g. mendeley.API). Next, we illustrate the different modes through the running example.

API tables. Click on the API button for a new browser tap to come up ("the annotating tab"). From now on, visited pages are augmented with an annotator, i.e. a widget for annotating Web data along four main categories: *HTTP request, query parameter, return parameter,* and *URL parameter.* The aim: recovering information that permits to populate an API table. In our example, we would like to obtain quotes from the documents held in a given folder. No single Mendeley method provides that. Rather, a chain of method calls is needed. To obtain such chain, the user first collects information about those methods through the annotator. Second, a pipe-like approach serves to link one method's output parameter with another method's input parameter Figure 3 provides an example. Users browse along the *Mendeley* documentation. When they find the methods of interest, annotator buttons are used to annotate the method call and the method parameters. Resulting annotations appear in the annotator tab. The display of parameters hold anchors from which pipelines can be set. In the example, folder method output ID serves to feed the documents method input ID parameter that stands for the document identifier. Parameters not participating in any pipe become the table's columns. This is the case of three parameter: *name*, *text* and *color*.

Once the annotating tab is closed, *Toolet* goes back to Google's Sheet where the API table is created. At this time, users can change column names at wish. Figure 2 shows the output for the *mendeley.API* table. For clarification purposes, the user change the initial column names *name* and *text* into *folderName* and *quote*, respectively. But this is not enough. So far, we obtained the quotes for the folder's documents. We need to know from where to hang these quotes: their parent node. This is obtained through color matching: quotes become children of color-like nodes. We obtain node colors through the *MindMeister* API. In this case, the *mindmeister.API* table is created with three columns: *nodeID*, *nodeContent* and *backgroundColor*.

Scrap tables. Click on the *Scrap* button for a new browser tab ("the XPath finder tab") to be opened. The provided URL is taken as the Web tool to be appropriated. Henceforth, visited pages are augmented with an XPath finder, i.e. hovering around will highlight different HTML content. Click on the content of interest for *Toolset* to obtain the XPath expression. Keep clicking till *Toolet* infer an XPath expression that select all HTML nodes of interest. The Finder will show an HTML representative of the elements being selected. Now, the user can select which properties will become table columns while *Toolet* infer the XPath counterpart. Figure 4 shows the case for three properties: *id, text* and *background-color*. Close the tab for returning to Google's Sheet. Column names can be changed at wish.

Widget tables. "Bridging" is often needed to contextualize someone else's data/services into the target tool. In some cases, this bridging is idiosyncratic, i.e. needs to be resolved by the user himself. For our example, which background colors to use for which concept is set by users. Hence, user interaction is required. And this is achieved through a widget. A widget is then first characterized by the data it holds rather than the means to obtain this data from the user. In our sample, the user should stain nodes through a palette of colors. The use of the same range of colors in *Mendeley* and *MindMeister* serves to set the mapping between quotes (in *Mendeley*) and parent nodes (in *MindMeister*). By clicking on the *Widget* button, developers are moved to *JSFiddle* where the widget is developed using HTML and JavaScript. This limits programming to widget development. The rest of the plumbing is taken care by *Toolet*.

3.2 Operation definition

An operation is a triplet: *<name, operationalSemantics, enactor>*. The *enactor* refers to the means to trigger the operation. This includes the GUI element (e.g. buttons), its placement (e.g. the page footer) and the triggering event (e.g. on clicking). Click the *Operation* button for the Web tool to be displayed in a separated browser tab. The page is augmented with a a panel for enactor definition: name, GUI element, placement and event (see Figure 5). Once this tab is closed, we go back to *Toolet's* main view. Next, the operational semantics. This is defined *á la* Query By Example (QBE), a GUI-based query language where users write queries by creating *example tables* on the screen. Benefits include that user needs minimal information to get started and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables. This fits our purposes. In addition, QBE graphical queries can be easily converted to SQL statements, ready to be processed by the *Toolet* engine.

In QBE, data needs and data updates are denoted by filling cells with system constants (e.g. $rootNodeID) or variables (e.g. _nodeID). Selection, insertion, deletion, and modification of a tuple are specified through the commands *P.*, *I.*, *D.*, and *U.*, respectively. Figure 2 provides an example for the operation *importQuotes*. Each row stands for a SQL operation:

1. the first row retrieves the *nodeContent* for the root node (kept in an application variable *$rootNodeID*). This *nodeContent* holds the node label and, in this case, should coincide with a *Mendeley's* folder,
2. the second row creates a new node for each quote. New quote nodes become children of the existing color-like nodes. In this case, the operation expand along different tables, given rise to a table join. Here, we have to select tuples from three tables with the same value in certain columns. We first retrieve quotes (and their colors) from the *mendeley.API* table. Next, we obtain the color-like node already in the map (from *mindmeister.SCRAP*).

So captured semantics is then transparently mapped into the corresponding API calls or HTML scraping counterparts.

4 Conclusions

We described the early stages of *Toolet*, a Google's Sheet plug-in to help users to adapt Web tools to their own practices. The aim is to hide complexity through a common table view. No matter how data is obtained (through APIs, Web Scrapping or user interactions), it is all represented as table columns. Next, new operations, better said, the operational semantics of these operations is specified using Query By Example.

We believe there will be an increasing pressure to Web appropriation. The large list of feature requests that queue up in the tools' Web sites so seems to suggest. *MindMeister* is a case in point. Its feature request page holds over a hundred petitions!![4] The question is how many of these petitions would be self-satisfied if suitable tools were available. *Toolet* attempts to provide some first insights.

References

1. Fidock, J., and Carroll, J. Why Do Users Employ the Same System in So Many Different Ways? *IEEE Intelligent Systems 26*, 4 (jul 2011), 32–39.
2. Johannesson, P., and Perjons, E. *An Introduction to Design Science.* Springer International Publishing, Cham, 2014.

[4] https://support.mindmeister.com/hc/en-us/community/topics/200108207-MindMeister-Feature-Requests