Champagne: a web tool for the execution of crowdsourcing campaigns

Carlo Bernaschina, Ilio Catallo, Piero Fraternali, Davide Martinenghi and Marco Tagliasacchi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
first.last@polimi.it *

ABSTRACT

We present *Champagne*, a web tool for the execution of crowdsourcing campaigns. Through Champagne, task requesters can model crowdsourcing campaigns as a sequence of choices regarding different, independent crowdsourcing design decisions. Such decisions include, e.g., the possibility of qualifying some workers as expert reviewers, or of combining different quality assurance techniques to be used during campaign execution. In this regard, a walkthrough example showcasing the capabilities of the platform is reported. Moreover, we show that our modular approach in the design of campaigns overcomes many of the limitations exposed by the major platforms available in the market.

Categories and Subject Descriptors

H.1 [**Models and Principles**]: User/Machine Systems; H.5 [**Information Interfaces and Presentation**]: Group and Organization Interfaces —*computer-supported cooperative work, web-based interaction*

Keywords

Crowdsourcing, Web applications, Data collection

1. INTRODUCTION

Crowdsourcing is emerging as a compelling tool for soliciting contributions from individuals on the web. In particular, in the last few years a multitude of so-called *crowdsourcing marketplaces* have appeared on the web. Among the most widely used, we cite Amazon Mechanical Turk[1] and CrowdFlower[2]. Such crowdsourcing marketplaces are web platforms whereby *task requesters* can publish their own tasks, which are then solved by *crowd workers* registered to the platform. More precisely, crowdsourcing marketplaces take care of publishing tasks, collecting results, and handling worker payment on behalf of the requester.

Task requesters could expect a crowdsourcing marketplace not only to permit the deployment of crowdsourcing campaigns at ease, but also to enforce the high quality of results, as well as to facilitate their subsequent aggregation and collection. Ideally, platforms should take care of these steps with the highest possible degree of automation, so as to pose themselves as off-the-shelf solutions for the requester.

However, as crowdsourcing is gaining a more prominent role in both the industry and academia, current crowdsourcing platforms are struggling in providing the fine-grained parameter selection requesters need. In fact, currently available crowdsourcing marketplaces suffer from several limitations, such as the lack of ready-to-use solutions for implementing worker hierarchies. As a result, requesters are frequently forced to implement such functionalities themselves, which tremendously increases the effort required for taking advantage of crowd work. Different authors tried to address these shortcomings by proposing toolkits for programming the crowd (see, e.g., [2], [1], [3]). Such mechanisms try to expand the capabilities offered to task requesters by providing already implemented patterns and facilities on top of major crowdsourcing platforms. As an example, the authors in [2] presented a JavaScript toolkit for implementing human computation algorithms in an imperative fashion on top of Amazon Mechanical Turk. Although partially easing the complexity behind the creation of a crowdsourcing initiative, these toolkits still require task requesters to program their own solution, and thus cannot be considered as a proper ready-to-use solution. Moreover, each toolkit is usually tightly coupled with a particular crowdsourcing platform, thus reducing the applicability of each toolkit-assisted solution to the sole supported crowdsourcing platform.

A crowdsourcing platform capable of accommodating complex requirements seems therefore much needed. To meet this need, we present *Champagne*, a tool for the execution of crowdsourcing campaigns on the web. Through Champagne, task requesters can model crowdsourcing campaigns as a sequence of choices regarding different, independent design decisions. Such decisions include, e.g., the possibility of qualifying some workers as expert reviewers, or of combining different quality assurance techniques to be used during campaign execution.

The remainder of this demo description is organized as follows. In Section 2 we present the conceptual model of Champagne. In Section 3 we provide an overview of the system, specifically focusing on the system architecture and exposed

*This work is partly funded by the EC's FP7 "Smart H2O" project, and the EU and Regione Lombardia's "Proactive" project

[1] `http://mturk.com`

[2] `http://crowdflower.com`

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742838.

API. Finally, In Section 4 we introduce a walkthrough example with the aim of showcasing what attendees will be able to do at the demo.

2. CONCEPTUAL MODEL

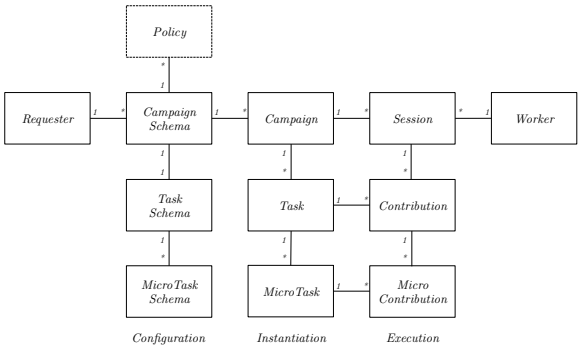

Figure 1: Champagne's conceptual model

The conceptual model of Champagne is reported in Figure 1. In this regard, figure elements have been conveniently positioned so as to induce a twofold relationship between entities. Specifically, reading the figure top to bottom naturally suggests a containment relationship between adjacent entities, i.e., each top-level entity acts as a container for the entity located below it (so that, e.g., different *Task*s may belong to the same *Campaign*). In a similar fashion, reading the figure left to right partitions entities according to their involvement in a specific phase of the crowdsourcing initiative lifecycle. As indicated by the labels located at the bottom of the figure, we identify three distinctive phases in the lifecycle of a crowdsourcing initiative, to which we refer as the *configuration*, *instantiation*, and *execution* phase, respectively. During the *configuration phase*, the task requester provides the specification of the crowdsourcing initiative. In this respect, in Champagne, each crowdsourcing initiative is described by means of the triple ⟨*Campaign schema*, *Task schema*, *MicroTask schema*⟩. This three-level configuration allows a set of tasks (each of which might be in turn composed of different, heterogenous microtasks) to be grouped together in order to form a campaign. After their definition, campaigns are ready to be created. As part of the *instantiation phase*, task requesters provide the needed data in order for the campaign (and the associated tasks and microtasks) to be instantiated. Once created, the campaign is made available on the platform, and crowd workers can start providing their contributions. We indicate this latter phase as the *execution phase*. Thanks to the decoupling between campaigns and campaign schemas, at any moment task requesters can generate new campaigns from their previously created campaign schemas. This mechanism allows task requesters to take advantage of the fact that, while in many cases campaign schemas are created once and then seldom changed, campaigns are instead repetitively created at any time needed. As a welcome byproduct, the platform itself provides a set of ready-to-use schemas, thus further decreasing possible barriers to entry for task requesters.

Furthermore, each campaign schema is associated with one or more *Policy* instances. Differently from all other entities in Figure 1, *Policy* denotes a family of concepts, rather

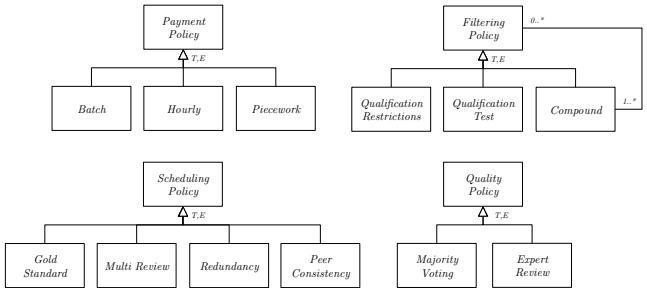

Figure 2: The available Policies in Champagne

than a single entity. As a matter of fact, each policy embeds a different, independent choice regarding the crowdsourcing campaign. Specifically, we argue the need of four fundamental structural decisions, namely, i) which payment method to adopt (*Payment policy*), ii) which crowd workers to allow in the crowdsourcing campaign (*Filtering policy*), iii) how to dispatch tasks to crowd workers (*Scheduling policy*), and iv) which quality assurance mechanism to adopt while collecting the contributions (*QA Policy*). For each such decision, Figure 2 depicts the different alternatives available in the platform. As an example, w.r.t. payment policies, task requesters may opt for either a hourly, batch or piecework payment.

3. SYSTEM ARCHITECTURE

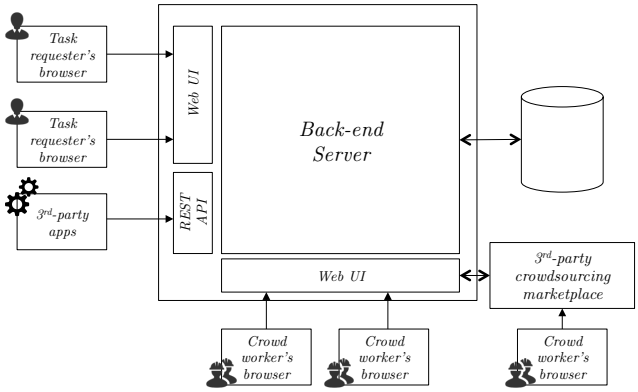

Figure 3: System architecture of Champagne

Starting from the conceptual model described in the previous section, we implemented a feature-complete proof-of-concept implementation of Champagne, whose architectural components are depicted in Figure 3. As shown, the system comprises a web user interface, a REST API for third-party integration, the back-end server, and the database. Let us comment in greater detail on each of such components.

The web interface enables the interaction between the system and its users, i.e., task requesters and crowd workers. As exemplified in the next section, task requesters can leverage the user interface in order to manage all the aspects related to their campaigns, from specification to deployment and monitoring. It is worth noting that, in contrast to other platforms, task requesters are not limited in the set of operations that can be executed through the user interface. Crowd workers interact with the user interface as the sole means for providing their contributions. Champagne assigns a unique URL to each instantiated campaign. This fact, to-

gether with the availability in many crowdsourcing platforms of the so-called "external tasks", allows the engagement of workers from external crowdsourcing platforms. Indeed, it is sufficient to publish on the target third-party marketplace one such external task. Crowd workers from the external platform will then be redirect to Champagne as soon as they accept the task.

Moreover, the platform exposes a set of REST API for the integration with third-party applications. The REST API doubles the set of operations a task requester can execute through the user interface, making them available in a programmatic fashion. To this end, we adopt JSON as the format for the representation of resources, which proved to be an effective way of exchanging data between applications. Note that the API offered by the platform is not meant as a compensation w.r.t. the deficiencies of the user interface (as is the case in, e.g., Amazon Mechanical Turk). This is because, as we said, the user interface and the REST API offers the same capabilities.

The back-end server implements the business logic of the platform. Given the intensive load on the application in terms of I/O operations, we opted for an event-driven approach, and choose Node.js[3] as the implementing technology. Finally, because of the richness and variability of the information we need to store, we opted for MongoDB[4], a document-oriented database.

4. WALKTHROUGH EXAMPLE

In this section we introduce a walkthrough example with the aim of showcasing the capabilities of the platform. Specifically, the set of operations required for the instantiation of a campaign is presented. A high-level description of the process is as follows. First, the task requester logs into the platform. If not already available, the task requester proceeds with the creation of the desired policies for the upcoming campaign. At this point, the campaign schema is ready to be assembled. In order to do so, the task requester associates the policies previously created with the campaign schema, and provides additional information required for the specification of the task and microtask schema. Finally, the task requester uploads the actual task data to the platform, which proceeds with the instantiation of the campaign. Let us now examine in depth each such step.

Step 1 - Signing in. As shown in Figure 4, after signing in, the task requester is prompted with a dashboard page summarizing statistics of interest regarding her campaigns. To this end, the dashboard is organized so as to provide access to crucial information at once in a intuitive and convenient way, especially if compared with other commercially-available platforms. In this respect, Amazon Mechanical Turk does not provide a dashboard page to its requesters, whereas CrowdFlower provides statistics only on a campaign base. Such a design has the downside of requiring the task requester to inspect each single running campaign separately in order to verify the overall correct execution. Note that, as far as this proof-of-concept implementation is concerned, the dashboard page reports the completion status of the currently instantiated campaigns. However, we plan to provide additional information, such as the time variation in the number of accepted contributions over the different campaigns, and more generally, statistics related to the crowd

[3] http://nodejs.org/
[4] http://mongodb.org

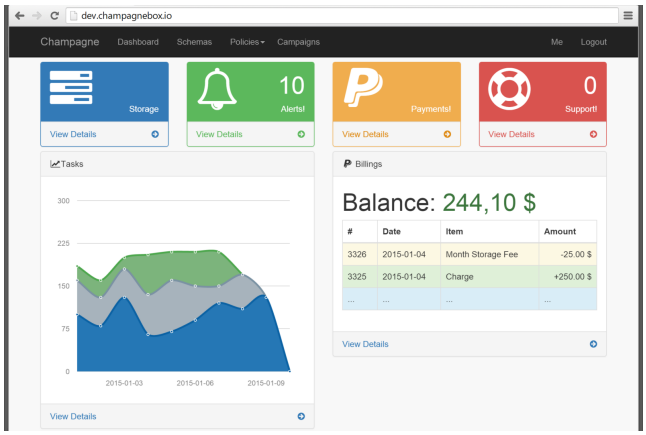

Figure 4: The task requester's dashboard

workers' performance throughout the different campaigns. Through the navigation menu, available at the top of the page, the task requester can create new policies to be later associated with campaign schemas. For the remainder of this section, we assume the task requester has not specified any policy yet. Therefore, we proceed with an explanation on how to create such policies. Due to the limited amount of space, we will show the creation of two policies out of the four introduced, specifically, filtering policies and payment policies.

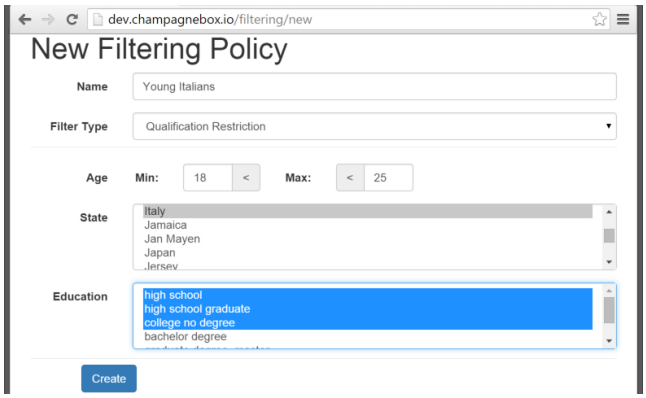

Figure 5: The creation of a filtering policy

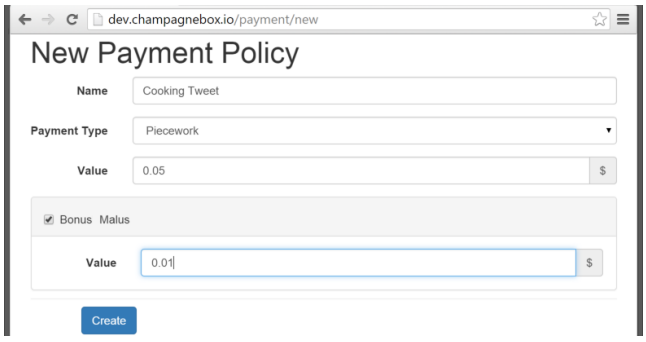

Figure 6: The creation of a payment policy

Step 2 - Creating policies. As mentioned in Section 2, we provide task requesters with the possibility of creating

policies independently of campaign schemas. We decided to do so in order to provide a flexible mechanism for creating crowdsourcing campaigns. Indeed, thanks to this high level of decoupling, it is possible to associate the same policy with different campaign schemas. As an example, in Figure 5 we show the creation of a filtering policy that limits the access to the associated campaigns to the sole crowd workers satisfying the specified requirements. In a similar manner, both Amazon Mechanical Turk and CrowdFlower provide simple filtering capabilities based on the geographical location of the workers. Nevertheless, Champagne permits the selection of additional constraints such as the education level and the age of the participants to the campaign. In order to reduce the risk of a malicious behavior, a crowd worker is inhibited from revising the same demographic data for a period of six months. In a similar fashion, it is possible to create payment policy, as shown in Figure 6. In this case, the task requester opted for a piecework payment, according to which crowd workers will be compensated 0.05 USD per completed task. Moreover, the task requester decided to grant crowd workers with 0.01 USD as a reward for each contribution judged as particularly significant or useful. Differently from other platforms, we also support alternative payment policies, such as, batch and hourly payments.

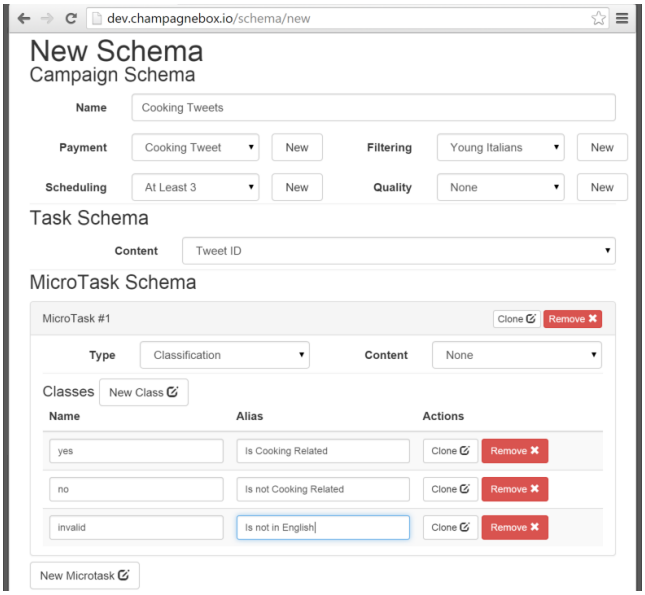

Figure 7: The creation of a new campaign schema

Step 3 - Specifying the campaign schema. As reported in Figure 7, the campaign schema is now ready to be assembled. As previously mentioned, this amounts to selecting the policies of interest, as well as defining the task and microtask schema. As an example, a campaign schema with the objective of classifying tweets as relevant for cooking is created under the name *"Cooking Tweet"*. Notice how the current schema is associated with the policies created at the previous step. In comparison, Amazon Mechanical Turk and CrowdFlower promote a hard coupling between policies and campaign schemas, so that it would not be possible to apply the same exact set of policies to different campaign schemas, therefore possibly forcing requester to duplicate the same policies over and over again. With respect to the task schema, the requester can specify the presence of some content to be visualized. We support images (plain URL, Instagram and Flickr), videos (Vimeo, Youtube and HTML5), tweets, and text. This capability stands as a clear advantage w.r.t. other platforms, in which the creation of campaigns involving complex content is hindered by the fact that the responsibility of correctly visualizing such content is totally on the task requester. In the example, the task requester requires the visualization of tweets as the task content. Moreover, a classification microtask is defined, which requires the crowd worker to indicate whether the tweet can be considered relevant for cooking.

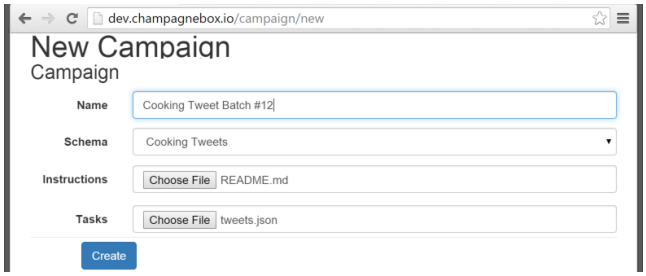

Figure 8: The creation of a new Campaign

Step 4 - Instantiating the campaign. In the final step, the task requester provides the data useful for the instantiation of the campaign under the form of a JSON file (Figure 8). With respect to the *"Coooking Tweet"* campaign, a possible data file could be as follows:

```
[ {
  "id": 0,
  "content": "https://twitter.com/lilbristow8/status/547382172794232832"
}, {
  "id": 1,
  "content": "https://twitter.com/ItsFoodPorn/status/544549175376224257"
}, ... ]
```

As easily understandable, JSON better supports the high variability in the specification of different campaigns when compared to schema-based format such as CSV, which is in fact adopted by both Amazon Mechanical Turk and CrowdFlower as the format for specifying tasks via user interface. Finally, notice that the exact same information can also be provided via the associated REST API call `POST https://api.champagnebox.io/campaign/new`.

5. CONCLUSIONS

We presented *Champagne*, a tool for the execution of crowdsourcing campaigns on the web. In the context of a walkthrough example, we highlighted some of the capabilities of our platform, and discussed how it compares with currently available solutions in the market. In particular, we showed that the modularity of the system allows the platform to support task requesters in the creation of campaigns that are not easily realizable with mainstream platforms. A full video outlining the presented walkthrough example is available at `http://www.champagnebox.io/www2015`.

References

[1] S. Ahmad et al. "The jabberwocky programming environment for structured social computing". In: *UIST*. 2011.

[2] G. Little et al. "Turkit: human computation algorithms on mechanical turk". In: *UIST*. 2010.

[3] P. Minder and A. Bernstein. "How to translate a book within an hour: towards general purpose programmable human computers with crowdlang". In: *WebSci*. 2012.