Editor: Gustavo Rossi • gustavo@lifia.info.unlp.edu.ar

Crowdsourcing Processes: A Survey of Approaches and Opportunities

Pavel Kucherbaev • *University of Trento, Italy*

Florian Daniel • *University of Trento, Italy, and Tomsk Polytechnic University, Russia*

Stefano Tranquillini and Maurizio Marchese • *University of Trento, Italy*

This article makes a case for crowdsourcing approaches that are able to manage crowdsourcing processes, that is, crowdsourcing scenarios that go beyond the mere outsourcing of multiple instances of a micro-task and instead require the coordination of multiple different crowd and machine tasks. It introduces the necessary background and terminology, identifies a set of analysis dimensions, and surveys state-of-the-art tools, highlighting strong and weak aspects and promising future research and development directions.

Crowdsourcing is the outsourcing of a unit of work to a crowd of people via an open call for contributions.[1] Thanks to the availability of online crowdsourcing platforms, such as Amazon Mechanical Turk or CrowdFlower, the practice has experienced a tremendous growth over the last few years[2] and demonstrated its viability in a variety of different fields, such as data collection and analysis or human computation — all practices that leverage on so-called micro-tasks, which ask workers to complete simple assignments (for example, label an image or translate a sentence) in exchange for an optional reward (such as a few cents or dollars). The power of crowdsourcing is represented by the crowd, which might be huge and span the world, and its ability to process thousands of tasks in a short time.

The practice is, however, also increasingly struggling with the inherent limitations of crowdsourcing platforms: not all types of work can easily be boiled down to simple micro-tasks, most platforms still require significant amounts of manual work and configuration, and there's very limited support for structured work — that is, work that requires the integration of different tasks and multiple actors, such as machines, individuals and the crowd. We call these kinds of structured works *crowdsourcing processes*, since they require the coordination of multiple tasks, actors, and operations inside an integrated execution logic.

Without proper support for the design and execution of crowdsourcing processes, running them requires a huge amount of manual development, data management, and coordination effort as well as specialized expertise. This shortcoming is acknowledged by the recent emergence of advanced crowdsourcing approaches, such as TurKit,[3] Jabberwocky,[4] and CrowdDB,[5] which all aim to ease the development and execution of crowdsourcing processes, typically by building on top of existing crowdsourcing platforms. However, they all come with a different perspective on the problem and, hence, present different features and capabilities.

With this in mind, here we introduce the reader to the problem of developing and running crowdsourcing processes and we provide an up-to-date picture of the approaches that have emerged so far. We identify a set of dimensions for the analysis of platforms for crowdsourcing processes and

 Published by the IEEE Computer Society 1089-7801/16/$33.00 © 2016 IEEE

review the state of the art accordingly. This analysis produces a set of considerations that might direct future research and development efforts.

Crowdsourcing Processes

Although not explicitly named as "crowdsourcing processes," the literature is rich with examples of scenarios that could benefit from explicit design and runtime support for crowdsourcing processes. Here, we briefly list some examples:

- Anand Kulkarni and his colleagues[6] crowdsource *article writing* (an article about the attractions of New York City) that involves tasks like structuring an article, writing narrative, splitting content into sections, adding pictures, iterating over content, and coordinating workers that write, correct, or structure text.
- Aniket Kittur and his colleagues[7] crowdsource a *trip-planning* scenario (a road trip from New York City to San Francisco) that requires, for instance, collecting routes, voting for routes, collecting details about hotels, restaurants, attractions, and iterating over the options based on feedback from the crowdsourcer (who crowdsources the micro-tasks; often called the "requester").
- Matthew Marge and his colleagues[8] study different *audio transcription* experiments (route instructions for robots), which require, for example, hosting audio records, deploying tasks in different batches, transcribing fragments and gluing them together, iterating over transcriptions until no typos are left, and controlling that workers don't contribute to different batches to avoid learning effects.
- In other work, Stefano Tranquillini and his colleagues[9] *mine patterns* from models with the help of the crowd, a scenario that requires dedicated task interfaces for the interactive selection of patterns, along with coordination of pattern identification and assessment tasks, automatically splitting/aggregating the available dataset, filtering patterns, and so on.
- The Galaxy Zoo project (www.galaxyzoo.org) is a good example of an *image classification* process that involves tasks such as classifying images into spiral, elliptical, irregular, or no galaxy, using a redundant number of workers, describing identified galaxies in function of their galaxy type (such as the number of arms in a spiral galaxy), and asking experts to resolve possible disagreements.

These examples show that in many practical settings, crowdsourcing isn't just a matter of deploying a set of simple micro-tasks on a given platform. Instead, it may comprise several different tasks (writing, transcribing, classifying, aggregating, spell checking, and voting), actors (crowdsourcers, workers, and experts), and automated operations (data splitting, resolving redundancy or multiple delegations, making decisions about whether to involve an expert, and synchronizing tasks). Running such processes on top of micro-task crowdsourcing platforms requires significant amounts of manual work — for example, to split or aggregate datasets or tasks, design task UIs for each task in the process, deploy tasks on the target platform, monitor task executions, collect data, integrate them, split them again, and so on. This is highly time consuming and inefficient, and there's huge potential for automation.

Dimensions of Analysis

To compare the capabilities of existing solutions for the development of crowdsourcing processes, from the aforementioned examples we derive the following core dimensions and subdimensions:

Definition language. Developing a crowdsourcing process requires a definition language following some paradigm and notation.

- A *paradigm* tells whether the language is imperative, declarative, or configuration-based (restricted to predefined templates or patterns).
- A *notation* specifies the specific language used, such as Scala, Business Process Model and Notation (BPMN), or extensions thereof.

Task support. Crowd tasks are micro-tasks performed by the workers of the crowd; they leverage on crowd providers and may provide for different crowd-management features. Machine tasks are automated operations performed by a machine, such as a data transformation.

- The *crowd provider* tells which crowd provider (crowdsourcing platform) is supported.
- *Crowd management* tells whether additional crowd management features (such as preselection or separation of duties) are supported.
- The *machine task definition* tells how machine tasks are specified — for example, via Web services or scripts.

Control flow support. Automating work means automatically coordinating tasks — that is, controlling the flow of action. The following are control flow features that crowdsourcing processes might need:

- task instantiation (individual and multiple instances);
- sequential execution;
- parallel execution;
- decision points for conditional flows;
- looping/iterating over similar tasks or data items;
- subprocesses (or routines/procedures) to support reuse.

Data management support. Next to progressing the computation from one task to another, it's also mandatory to provide each task with the necessary input data. The following are the basic data management requirements highlighted in our scenario:

- *Data hosting* tells whether the tool hosts data (such as audio transcriptions) or references to data (the URLs to the audio files).
- *Data passing* tells whether data are passed via data flows, by value (variables) or by reference (shared memory).
- *Data splitting/aggregation* tells how data transformations are specified.

Development support. Implementing a crowdsourcing process further requires designing suitable crowd tasks and deploying them on the crowdsourcing platform.

- *Crowd task design* tells if and how the tool supports the design of crowd tasks.
- *Task deployment* tells if and how the tool supports the deployment of tasks.

Quality control support. Finally, a crucial aspect in crowdsourcing is quality control. This dimension therefore looks at which built-in quality control techniques are supported (for example, iterating over text until no typos are left).

Approaches and Tools

The approaches we review in the following are the result of two years of watching emerging technologies in the context of crowdsourcing. In particular, we consider general-purpose approaches that don't restrict the types of tasks or processes you can crowdsource. Also, at the time of writing, suitable research papers or online resources must have been available so that we could make an informed assessment of the identified dimensions. These criteria led us to the 11 approaches that we describe next.

Selected Approaches

TurKit[3] is a JavaScript-inspired scripting language that allows one to programmatically deploy tasks on Mechanical Turk and to pass data among tasks.

AutoMan[10] is a Scala-based programming language similar to TurKit that automatically manages the scheduling and pricing of task instances and the acceptance and rejection of results, given a target result quality.

Jabberwocky[4] is a MapReduce-based human computation framework with a parallel programming framework and language.

CrowdComputer[9] is a BPMN-based design and runtime environment for complex crowdsourcing processes with support for crowd and machine tasks as well as individuals (for example, experts).

CrowdLang[11] is a BPMN-inspired modeling language with crowdsourcing-specific constructs.

CrowdWeaver[12] is a similar model-based tool with a proprietary notation.

CrowdDB[5] is an SQL-extension that lets you embed crowd tasks (such as inputs and comparisons) into SQL queries.

AskSheet[13] is a Google Sheet extension with functions that allow the spreadsheet to leverage on crowdsourcing tasks.

Turkomatic[6] is a crowdsourcing tool for complex tasks that delegates not only work to the crowd but also task management operations (such as splitting tasks).

CrowdForge[7] is a Django-based crowdsourcing framework for crowdsourcing processes similar to Turkomatic that, however, follows the Partition-Map-Reduce approach.

CrowdSearcher[14] is a system that lets you design processes using reusable design patterns and leverage on machine and crowd tasks as well as on tasks deployed on Facebook.

We're also aware of other instruments, such as CrowdFlow, Quirk, TurkDB, WorkFusion, and CrowdFlower Workflows, but we weren't able to collect enough public information on them. Other approaches, such as CrowdTruth or QualityCrowd2, are tailored to specific domains (collecting gold data for machine learning and video quality assessment, respectively).

Comparing Features

Table 1 (see p. 54) describes the selected platforms applying the dimensions and subdimensions of analysis introduced earlier. We also add a "public availability" dimension to the analysis, to reflect if and how an approach can be tried out and tested. To better highlight commonalities and differences, we group the approaches according to the paradigm of their process *definition language* (the suborder doesn't follow any temporal or functional order):

- In the *imperative, textual* approach, a person (the crowdsourcer) writes code telling how the process is executed. The specific notations used are Scala, a JavaScript-like language, or a proprietary language (Dog).
- For the *imperative, visual* approach, a person models how to execute the process visually using graphical abstractions. The concrete notations are BPMN extensions, BPMN-like notations, or custom notations.
- In the *declarative* approach, a person defines what should be processed or obtained as an output. The SQL or spreadsheet formulas are examples of notations used.
- For the *configuration* approach, a person fills configuration properties that set up a *predefined* process logic. In this case, the crowdsourcer is typically guided through the configuration by a wizard.

As for *task support*, the most-used *crowd provider* is Mechanical Turk (MTurk), which is, however, restricted to crowdsourcers from the US only; CrowdFlower doesn't have this restriction. Some approaches self-manage their own crowd. CrowdSearcher proposes an alternative interpretation and also supports deploying tasks on Facebook, which adds extra opportunities such as access to people who wouldn't use conventional crowdsourcing platforms (such as teenagers) and volunteer work by people in the

crowdsourcer's own social network. CrowdComputer, given its roots in business process management (BPM) that focuses on coordinating human work, also supports assigning tasks to individuals (such as an expert) via a conventional BPM engine. *Crowd management* features are only scarcely supported and mostly focus on worker preselection, bonus payments, and approval or rejection of results. *Machine tasks* come in different flavors: the imperative, textual approaches allow the crowdsourcer to write his or her own scripts; the visual approaches support reusable modules such as Web services; the declarative approaches are limited to their environment's native capabilities; and the configuration approaches may provide for customizable, built-in machine tasks (such as for data management).

From the *control flow* perspective, all the platforms support automated *task instantiation*. Given their imperative nature, both the textual and the visual approaches support most of the control flow features; control flow support by the declarative and configuration approaches is platform-specific. *Sequential execution* is supported by all except AskSheet (spreadsheet functions are evaluated in parallel). *Parallel execution* is also more platform-specific. *Decision points* come either as if-statements in imperative, textual approaches and AskSheet, graphical gateways in the imperative, visual approaches, or adaptation rules in CrowdSearcher. There are no explicit decision points in Turkomatic, where the workers decide at runtime whether to split a task or execute it. *Iterative execution* isn't supported in platforms without decision points. *Subprocesses* are only weakly supported; if supported, they're either reusable functions (imperative, textual approaches), BPMN processes (CrowdComputer), or Python scripts (CrowdForge).

Regarding *data management*, all platforms support the *hosting* of data, except CrowdComputer, which only manages data references. While this requires the crowdsourcer to manage the actual data himself, it reduces data transfer and lets the crowdsourcer protect data that's sensitive (such as images with nudity) or subject to local regulations (such as healthcare data). CrowdLang and CrowdForge pass data by value; CrowdComputer and AskSheet pass data by reference; the other approaches use direct data flows. *Data splitting* and *aggregating* logics are either built-in operators, custom crowd tasks, or coded in the underlying process definition language.

Development support for task design comes in different flavors: manual, automatic, wizard-based, or predefined tasks. Manual design asks the crowdsourcer, for instance, to develop HTML-based Web forms (CrowdComputer) or XML task definitions (CrowdForge). AutoMan is instead able to automatically generate task user interfaces out of an SQL query and the affected table schemas. A wizard-based design is proposed by CrowdSearcher, while Turkomatic and AutoMan are examples of platforms that support only predefined text editing and voting tasks. *Task deployment* is generally automatic; CrowdComputer asks the crowdsourcer to host task implementations, which might also require manual intervention.

Also, for *quality control*, we identified four main approaches: *rating* (a crowd task is used to rate work of another task); *voting* (a crowd task is used to collect preferences for results of another task); *consensus* (new results are accepted until at least two or more results match); and *control questions* (extra questions, for which the correct answers are known, are injected into a crowd task to evaluate a worker). Automan stands out in this context: it lets the crowdsourcer define an overall budget and a target confidence level for the results and automatically manages the necessary pricing, approval, and rejection of tasks.

As for the *availability* of the approaches, four out of 11 platforms are open source projects, but only two are actually deployed online and ready for use; six platforms aren't available at all. In this respect, it's interesting to note that all the approaches are research prototypes. We're aware that companies such as CrowdFlower and Workfusion deploy and run crowdsourcing processes on behalf of their enterprise customers at a daily basis. Workflows, CrowdFlower's internal platform for crowdsourcing processes, is also available for enterprise customers; however, the commercial offering is still fairly limited, if non-existent. This could be an indication that the goals and effectiveness of platforms for crowdsourcing processes aren't yet clear and crisp enough for the market.

Discussion and Outlook

The selection of crowdsourcing approaches discussed in this article shows that a diverse and growing ecosystem of sophisticated solutions already exists. As usual with automation instruments, their usefulness in practice is a tradeoff between how often a process is repeated (for example, to test different crowdsourcing settings) and how easy it is to use the instrument (compared to manual crowdsourcing). If not in their current form (stand-alone platforms), we expect that eventually — after the initial prototypes introduced in this article — support for crowdsourcing processes will percolate into and enhance existing crowdsourcing platforms, as is already happening with CrowdFlower Workflows.

We further discussed our analysis with Lukas Biewald, CEO of CrowdFlower (the company operates as both crowd provider and crowdsourcer on behalf of its key customers), so as to jointly identify some of the challenges that the crowdsourcing community will have to approach next to foster tools for crowdsourcing processes. Here are our thoughts.

Integration. The prevalence of proprietary notations for process definition

Table 1. Analysis of crowdsourcing platforms for crowdsourcing processes.

		TurKit	AutoMan	Jabberwocky	CrowdComputer	CrowdLang
Definition language	Paradigm	Imperative, textual	Imperative, textual	Imperative, textual	Imperative, visual	Imperative, visual
	Notation	JavaScript-like	Scala	Dog	BPMN extension*	BPMN-like
Task support	Crowd provider	MTurk	MTurk	Self	Self, BPM engine	MTurk
	Crowd management	–	Approvement and rejection	Profile-based (expertise, demographics, groups) preselection	Profile-based preselection	–
	Machine tasks definition	Script	Script	Script	Generic Web services	Generic machine tasks
Control flow support	Task instantiation	✓	✓	✓	✓	✓
	Sequential execution	✓	✓	✓	✓	✓
	Parallel execution	–	–	✓	✓	✓
	Decison points	✓	✓	✓	✓	✓
	Looping/iterative execution	✓	✓	✓	✓	✓
	Subprocess	✓	✓	✓	✓	–
Data management support	Data hosting	Data	Data	Data	References	Data
	Data passing among tasks	By value	By value	By value	By reference	By value
	Data splitting, aggregating	Script	Script	Script	Built-in	Built-in
Development support	Task design	Manual	Predefined	Manual	Manual	Automatic
	Task deployment	Automatic	Automatic	Automatic	Automatic and manual	Automatic
Quality control support		Voting	Confidence levels under given budget	–	Custom logics	–
Public availability		Open source	Open source	–	Open source, deployed online	–

* BPMN = Business Process Model and Notation.

risks to make integration with other computing environments cumbersome. Textual approaches (except AutoMan: Scala) are especially hard to integrate into other programming environments; the same holds true for the visual approaches (except for CrowdComputer: BPMN) and the configuration-based approaches. Only the declarative approaches seem well-integrated into their host environments (databases and spreadsheets). However, many of the surveyed approaches are equipped with APIs that can be programmed and leveraged on for integration from the outside.

Quality control. The supported techniques to control the quality of the results produced by the crowd are still rather limited, and quality is controlled at the granularity of individual crowd tasks only. More complex quality control logics (for example, providing quality that guarantees the ability to raise exceptions and to dynamically compensate for low quality) or logics that control quality at the granularity of entire crowdsourcing processes (being able, for example, to maximize the quality of outputs while at the same time keeping a given budget and time restrictions) still require more research.

Adaptive process execution. Crowdsourcing usually requires a significant

CrowdWeaver	CrowdDB	AskSheet	Turkomatic	CrowdForge	CrowdSearcher
Imperative, visual	Declarative	Declarative	Declarative	Configuration	Configuration
Custom modeling language	Extended SQL	Google Spreadsheet formula	–	Wizard for config, Python for custom processes	Wizard plus adaptation rules
CrowdFlower	MTurk	MTurk	MTurk	MTurk	MTurk, Facebook
–	Approvement, rejection, bonus payment	–	–	–	–
Generic machine tasks	SQL operations	Spreadsheet functions	–	–	Data management operations
✓	✓	✓	✓	✓	✓
✓	✓	–	✓	✓	✓
✓	–	✓	✓	✓	✓
–	–	✓	–	–	✓
–	–	–	–	–	✓
–	–	–	–	✓	–
Data	Data	Data	Data	Data	Data
Data flow	Data flow	By reference	Self-managed data flow	By value	Data flow
Built-in	SQL operations	Spreadsheet functions	By crowd	By crowd	Built-in
Wizard	Automatic	Wizard	Predefined	Manual	Wizard
Automatic	Automatic	Automatic	Automatic	Automatic	Automatic
Control questions, consensus	Consensus	Rating, consensus	Voting	Voting	Consensus
–	–	–	–	Open source	Deployed online

testing and fine-tuning effort for both individual tasks and entire processes. Many times, processes are constructed by running a task, analyzing its output, deciding whether postprocessing of the data is needed or whether the next crowd task can be executed, and so on. This, on the one hand, asks for novel testing techniques for crowdsourcing processes and, on the other hand, for crowdsourcing processes that can be started, even if not yet completely defined, and that can be refined at runtime – for example, by adding ad hoc tasks or operations.

Worker selection and training. The success of crowdsourcing depends first and foremost on the quality of work produced, and this, in turn, depends on the workers' skills and abilities. However, the solution isn't always about selecting workers with the necessary skills, though – especially if, for example, a given skill or domain knowledge isn't present at all. A challenge for future crowdsourcing practice is therefore to understand how to train workers for specific skills, how to motivate them to participate in training, how to reward and certify training, and how to properly value training in the selection of workers. These are all advanced crowd-management aspects that will require effective answers.

This survey is based on the analysis of research papers and hands-on tests of the available prototypes. Acknowledging the limitations of this approach (the level of detail of papers, impossibility to access prototypes, and the pace of evolution), we intend to add a new section on crowdsourcing processes to the "Crowdsourcing" entry in Wikipedia (https://en.wikipedia.org/wiki/Crowdsourcing), enabling everybody to integrate and extend this analysis as a community effort.

Acknowledgments

We thank Lukas Biewald (CrowdFlower), Nicola Sambin (SpazioDati), Patrick Minder and Abraham Bernstein (CrowdLang), Alexander J. Quinn (AskSheet), and Marco Brambilla (CrowdSearcher) for their help.

References

1. J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, Crown Publishing Group, 2008.
2. Crowdsourcing Week, "2014 Global Crowdsourcing Pulsecheck: 1st Annual Survey Topline Results," *Slide Share*, Apr. 2015; www.slideshare.net/crowdsourcingweek/2014-global-crowdsourcing-pulsecheck-1st-annual-survey-topline-results.
3. G. Little et al., "TurKit: Human Computation Algorithms on Mechanical Turk," *Proc. 23rd Ann. ACM Symp. User Interface Software and Technology*, 2010, pp. 57–66.
4. S. Ahmad et al., "The Jabberwocky Programming Environment for Structured Social Computing," *Proc. 24th Ann. ACM Symp. User Interface Software and Technology*, 2011, pp. 53–64.
5. M.J. Franklin et al., "CrowdDB: Answering Queries with Crowdsourcing," *Proc. SIGMOD*, 2011, pp. 61–72.
6. A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively Crowdsourcing Workflows with Turkomatic," *Proc. ACM 2012 Conf. Computer Supported Cooperative Work*, 2012, pp. 1003–1012.
7. A. Kittur et al., "Crowdforge: Crowdsourcing Complex Work," *Proc. 24th Ann. ACM Symp. User Interface Software and Technology*, 2011, pp. 43–52.
8. M. Marge, S. Banerjee, and A.I. Rudnicky. "Using the Amazon Mechanical Turk for Transcription of Spoken Language," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, 2010, pp. 5270–5273.
9. S. Tranquillini et al., "Modeling, Enacting and Integrating Custom Crowdsourcing Processes," *ACM Trans. Web*, vol. 9, no. 2, 2015, article no. 7.
10. D.W. Barowy et al., "AutoMan: A Platform for Integrating Human-based and Digital Computation," *Proc. ACM Int'l Conf. Object Oriented Programming Systems Languages and Applications*, 2012, pp. 639–654.
11. P. Minder and A. Bernstein, "CrowdLang — Programming Human Computation Systems," *Proc. Web Science Conf.*, 2012, pp. 209–212.
12. A. Kittur et al., "Crowdweaver: Visually Managing Complex Crowd Work," *Proc. ACM 2012 Conf. Computer Supported Cooperative Work*, 2012, pp. 1033–1036.
13. A.J. Quinn and B.B. Bederson, "AskSheet: Efficient Human Computation for Decision Making with Spreadsheets," *Proc. 17th ACM Conf. Computer Supported Cooperative Work & Social Computing*, 2014, pp. 1456–1466.
14. A. Bozzon et al., "Pattern-Based Specification of Crowdsourcing Applications," *Proc. Int'l Conf. Web Eng.*, LNCS 8541, Springer, 2014, pp. 218–235.

Pavel Kucherbaev is a PhD student in the Department of Information Engineering and Computer Science at the University of Trento, Italy, and the European Institute of Technology Digital program. His research focuses on quality control and timeliness of crowdsourcing micro-tasks. Contact him at pavel.kucherbaev@unitn.it.

Florian Daniel is a senior research fellow at the University of Trento, Italy, and professor at the Tomsk Polytechnic University, Russia. His research focuses on crowdsourcing, Web engineering, mashups, service-oriented computing, and business process management. Daniel has a PhD in information technology from Politecnico di Milano. Contact him at florian.daniel@unitn.it.

Stefano Tranquillini is a postdoctoral researcher in the Department of Information Engineering and Computer Science at the University of Trento, Italy. His main research interests are in the areas of business process management, crowdsourcing, and the integration of the two. Tranquillini has a PhD in information engineering and computer science from the University of Trento. Contact him at stefano.tranquillini@unitn.it.

Maurizio Marchese is an associate professor in the Department of Information Engineering and Computer Science at the University of Trento, Italy, and a director of education in the European Institute of Technology ICT Labs initiative for the Trento node. His main research interests are in social informatics, where he studies how information systems can realize social goals, apply social concepts, and become sources of information relevant for social sciences and for analysis of social phenomena. Contact him at maurizio.marchese@unitn.it.