Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

CrowdWON: A Modelling Language for Crowd Processes Based on Workflow Nets

David Sánchez-Charles and **Victor Muntés-Mulero**
CA Labs, CA Technologies, Barcelona (Spain).
{David.Sanchez, Victor.Muntes}@ca.com

Marc Solé
Universitat Politècnica de Catalunya - BarcelonaTech,
Barcelona (Spain).
msole@ac.upc.edu

Jordi Nin
Barcelona Supercomputing Center (BSC),
Universitat Politècnica de Catalunya - BarcelonaTech,
Barcelona (Spain).
{nin@ac.upc.edu

Abstract

Although crowdsourcing has been proven efficient as a mechanism to solve independent tasks for on-line production, it is still unclear how to define and manage workflows in complex tasks that require the participation and coordination of different workers. Despite the existence of different frameworks to define workflows, we still lack a commonly accepted solution that is able to describe the most common workflows in current and future platforms. In this paper, we propose CrowdWON, a new graphical framework to describe and monitor crowd processes, the proposed language is able to represent the workflow of most well-known existing applications, extend previous modelling frameworks, and assist in the future generation of crowdsourcing platforms. Beyond previous proposals, CrowdWON allows for the formal definition of adaptative workflows, that depend on the skills of the crowd workers and/or process deadlines. CrowdWON also allows expressing constraints on workers based on previous individual contributions. Finally, we show how our proposal can be used to describe well known crowdsourcing workflows.

1 Introduction

According to massolution (Massolution 2013), there was a continuous growth of crowd-based companies from 2009 to 2012. Analogously, the crowdfunding market has also grown significantly (Thorpe 2013). In general, trends seem to indicate that industrial interest in crowdsourcing will continue increasing in the near future. Although business processes may require to solve complex tasks, crowdsourcing is mainly used to solve independent tasks. Previous work such as (Bernstein et al. 2010) shows that finding the proper combination of tasks to solve a complex problem using crowdsourcing is not straightforward. In fact, designing proper workflows is one of the major issues in crowdsourcing according to employees of the crowd-based company CrowdFlower[1] (Kittur et al. 2012). Although some previous works propose mechanisms to express collaboration patterns in a visual way, their expressive power is still not sufficient to describe the variety of current crowdsourcing processes. Just as an example, none of the visual workflow languages used for crowdsourcing allow users to define the complete workflow behind a continuously open competition such as those in Threadless[2] and Innocentive[3], in an easy way.

Recent research (Sun et al. 2014) shows the interest of industry on increasing the participation of in-house workers in crowdsourcing processes. In many of these applications, requesters need to express restrictions on the characteristics of individual workers. Unfortunately, previous workflow languages used in crowdsourcing cannot express usual rules on workers such as allowing to solve a task to only those individuals who contributed in a previous task of a process. Besides, processes in crowdsourcing may change dynamically depending on worker skillsets or deadline. Authors in (Kulkarni, Can, and Hartmann 2012), discuss the need for continuously adapting workflows depending on the context. Another important concern is compliance with deadlines (van der Aalst, Rosemann, and Dumas 2007) (Mason and Watts 2009). To our knowledge, none of the workflow languages for crowdsourcing presented in the literature allows for expressing these usual requirements accurately.

In this paper, we define CrowdWON, a graphical modelling tool suitable for describing crowdsourcing processes using petri net extensions. Our main contributions are:

- We propose a flexible language that allows defining workflows for a large variety of scenarios, including open competitions.
- We propose the first graphical language for crowdsourcing platforms able to express sophisticated restrictions on the workers participating in a task.
- We propose a language that allows to define dynamic workflows that adapt to the status of the process; in particular, it adapts to deadlines and worker profiles.

Our modelling language makes it easier to define busi-

[1]http://www.crowdflower.com

[2]http://www.threadless.com

[3]http://www.innocentive.com

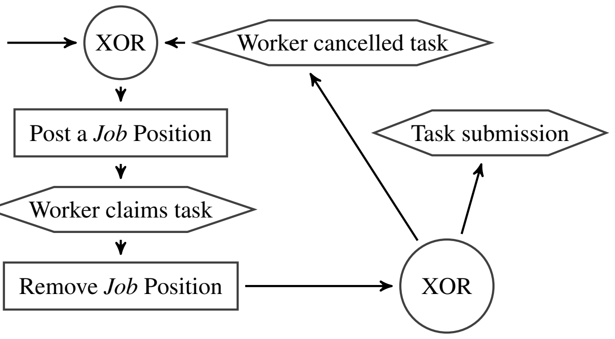


Figure 1: Description of the execution of a task in a generic crowdsourcing platform using an EPC diagram.

ness processes in crowdsourcing. Thanks to these proposal we can easily visualize and improve the collaboration between individuals in the crowd, computers and companies. Our adaptative flows make it possible to design collaboration patterns that allow to react when the performance of the process is below expectations.

In Section 2, we review previous contributions in the crowdsourcing context and existing modelling frameworks. The definition of our modelling language can be found in Section 3. Then, in Section 4 we propose a mechanism for describing changes in the flow triggered by deadline constraints. In Section 5, we describe two examples of actual crowdsourcing processes using CrowdWON. Finally in Section 6, we conclude and set some guidelines for future improvements of CrowdWON.

2 Previous Work

Despite the increasing popularity of crowdsourcing, the number of specialized process-modelling frameworks proposed in the literature is still limited. With TurKit (Little et al. 2010), anyone is able to describe the iterations of simple tasks in crowdsourcing. Jabberwocky (Ahmad et al. 2011) focuses on the reusability of sequential processes, but its application to iterative processes is not clear. In Turkomatic (Kulkarni, Can, and Hartmann 2012), individuals collaborate in the design of the process and contribute, through refining the textual description of tasks, to its resolution. CrowdWeaver (Kittur et al. 2012) is the first graphical modelling framework for workflow definition and management. With this tool, individuals can design a crowdsourcing process by combining human and predefined computer tasks.

In the aforementioned modelling frameworks, tasks and processes are completed after a known number of individuals contribute to them. However, using these previous proposals, it is not straightforward to represent other scenarios, such as for instance, crowdsourced open competitions, in which the end of the process is defined by a deadline. Deadline management are usually based on simple mechanisms such as notifications to the process manager, so that she can manually adjust the workflow (*e.g.* increasing the financial reward in order to reduce the time-to-completion of tasks (Mason and Watts 2009)). Task routing techniques (Hassan, O'Riain, and Curry 2013) showed that tasks can be adapted to the profile of crowd workers. However, none of these previous proposals allows expressing automatic transformations of the workflow based on the context.

Petri Nets (Petri 1966) are a common tool to model processes and workflows. Petri Nets can be defined as directed bipartite graphs, in which nodes represent inactive systems (*places*) or active systems (*transitions*). Information is represented through tokens (pointers to places) and the execution of a process changes the position of tokens. The basic idea of Petri Nets is that transitions transform information from one state (place) to another. Several extensions of Petri Nets can be found in the literature. We highlight two of them related to process modelling. Workflow Nets (van der Aalst 1998) are Petri Nets with two special places to represent the start and end of the process, and dualistic Petri Nets (Dawis, Dawis, and Koo 2001) which allow tokens to point to places and transitions: A token points to the transition while the transformation is being performed. Due to the complexity of describing human collaboration and time management, other modelling languages–*e.g. BPMN*[4] and *EPC*[5]–include conditions based on human and timed events. Unfortunately, the use of these event-based frameworks produces complex diagrams in the crowdsourcing context. Figure 1 shows the execution of a single task using the EPC language.

3 Crowdsourcing Workflow Net model

We propose CrowdWON, a language to describe crowdsourcing processes which is a combination of Workflow Nets and Dualistic Petri Nets. As in the latter, a token pointing to a task (or transition) represents an individual performing such task. We define the status of tasks based on the position of the token. Figure 2 shows the different status depending on the token position. Places are implicitly represented in any direct connection between two tasks. Besides, in order to adapt to usual processes in crowdsourcing, we allow tokens to return to a previous place. With this, we can design a *reverse firing rule*: If a worker canceled the task, the token should return to its previous place.

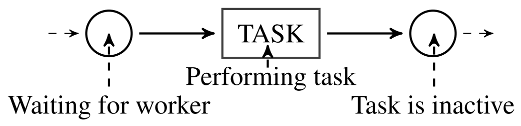


Figure 2: Status of the task depending on the token location.

It is important to remark the simplicity of CrowdWON thanks to this representation. As we mentioned before, Figure 1 describes the execution of a single task using EPC (event-driven language). In our proposal, by considering the position of a token, we are able to simplify the representation to only three nodes: Two places connected to a task (see Figure 2). A *Job position* is created when a token arrives to the first place, and it will be removed when the token leaves the place. If the worker cancels the task, the token will return

[4]Business Process Model and Notation. `hwwp://www.bpmn.org`

[5]Event-driven process chain

to the first place repeating the creation of the job position (describing the loop in Figure 1). The second place is only involved when the worker submits the task.

Tasks in CrowdWON are a tuple (*ID*, $typein$, $typeout$, $kernel$) that defines the execution of the task. *ID* is a unique identifier allowing future references of the task; $typein$ is a specification of the structure of the input data ($typeout$ specifies the returned structure); and $kernel$ defines the method used to perform the task. Depending on the type of task, $kernel$ may contain different information: Human tasks (represented with a box) require at least a user interface; computer tasks (double-edged box) require a service protocol; a special case of tasks are control flows, used to describe non-sequential processes. In Subsection 3.2, we describe the different control flows available in our model.

Tokens in CrowdWON are a tuple ($current$, $data$). The token is pointing to the $current$ node, and $data$ is the information that is being processed by the $current$ node. If $current$ is a task, then $data$ must be structured as specified in $typein$. That allows individuals to complete the task using the information contained in $data$. After the task is completed, the contribution is stored in $data$ as specified in $typeout$. Therefore, it is important that two consecutive tasks have compatible data structures. At the end of the process, $data$ is the proposed solution to the problem. Note that this construction only allows one individual to perform the task with the same piece of information. Later in Section 3.2, we introduce a mechanism to describe resolution of tasks by multiple workers.

3.1 Worker management

Popular crowdsourcing platforms allow defining restrictions on which individuals can claim the task. Those restrictions are usually related to the ratio of successful submitted tasks, passing some preliminary tests and geo-location. One of the main contributions of CrowdWON is the proposal of a language that allows expressing worker selection constraints in a simple way. These restrictions may be complex: a worker might not be able to take a task if she participated in a particular previous task in the process, workers with a lack of certain skills might only be allowed to solve a task when deadline compliance is at risk, as the last resort, etc. Although these restrictions may be usual in many circumstances, previous languages do not allow to express them. CrowdWON allows for expressing these types of constraints.

Definition. *Let us call **worker requirement** to a set of constraints on the group of individuals allowed to choose a certain task. Constraints are defined over any property of the profile of individuals (such as ratio of approved contributions, knowledge about a topic, age, gender or location), belonging to a group of individuals or any combination (using conjunctions, disjunctions or negations).*

CrowdWON provides a set of predefined groups of a given task ID to facilitate the design of a worker requirement:

- $e(\text{ID})$ is the set of workers who completed the task.
- $r(\text{ID})$ is the set of workers who failed to complete the task or revoked it.
- $a(\text{ID})$ is the set of all workers that at some point claimed the task.

Definition. *Let us call **worker selection** to an ordered list of tuples (r, d), where r is a worker requirement and d is an optional deadline.*

The list defines a priority on the requirements: Individuals must satisfy the first worker requirement in the list in order to claim the task. When a requirement is an empty set, or the deadline has already been met, the tuple is removed from the list. An empty list puts no restriction on individuals, and so anybody can claim the task.

In Figure 3, an example of our worker selection model is depicted. The *Review* task only accepts workers with a success ratio greater than 90%. On the other hand, the selection node has a more complex worker selection: firstly, only workers with a ratio greater than 80% are considered. After 1 day, the worker selection will decrease the threshold to 40%. After 2 days, anyone can claim the task.

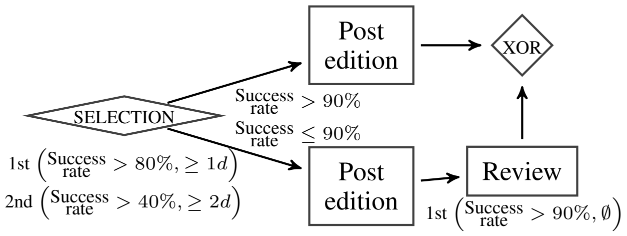
3.2 Control flow tasks

In CrowdWON, analogously to other modelling languages, the two most basic tasks to control the data flow are AND and OR operators. The AND operator must wait until it receives a token from each of the incoming edges. Then, all these tokens are automatically merged into a single one that remains in the flow of tasks. If the operator has more than one outgoing edge, then a copy of the token is created for every outgoing path. In this case, the *kernel* parameter may specify extra rules for the merging and creation of the tokens (such as splitting a list instead of creating copies). On the other hand, the XOR operator only accepts one token at a time from incoming edges, and only one outgoing path is executed. The operator will look at data contained in the token to decide the outgoing path. Conditions for execution will be written in terms of the received data structure i. The default path will be represented by an empty condition.

SELECTION Authors in (Hassan, O'Riain, and Curry 2013) show an example of a task that adapts to the expertise of workers: If an individual has knowledge in the American culture, he will answer more questions related to America than other countries. In order to describe these adaptable processes, we extended the XOR operator to include conditions over the profile of workers. The SELECTION node is a control flow task that first finds an individual following the worker selection criteria attached to the node; and then the profile of the selected worker is used to choose an outgoing path.

Figure 3 shows an example of a workflow that behaves differently depending on the individuals involved in solving the task. Inexperienced workers (*e.g.* success rate lesser than 90%) can perform the post edition task, reducing the workload of experienced workers.

MAPREDUCE STRUCTURE Collaboration in crowdsourcing processes follows an asynchronous approach: crowd workers are asked to perform tasks individually, and then an aggregation of their contributions is the final output.



Post edition
XOR
SELECTION
Success rate > 90%
Success rate ≤ 90%
1st (Success rate > 80%, ≥ 1d)
2nd (Success rate > 40%, ≥ 2d)
Post edition
Review
1st (Success rate > 90%, ∅)

Figure 3: Example of the Selection operator. If inexperienced workers claim the first task, then an extra review phase will be required.

The MapReduce structure defines the level and mechanisms of collaboration.

Following the PartitionMapReduce approach of CrowdForge (Kittur et al. 2011), we divided our structure in:

- A *sub-process*. The operator contains another crowdsourcing workflow net, defining how data will be processed. We can graphically represent it in the parent process as in Figure 4.
- A *generator of tokens*. Every token enters the structure through this generator. It has some rules to create copies (or chunks) of the received data. These tokens are processed independently as described in the sub-process. A list of common generators can be found in Table 1
- Finally, we have an *aggregation mechanism* that produces a single output from all the independent contributions. This last part is connected to the generator, so it knows how many tokens are being manipulated by the sub-process. It is also able to request the creation of additional tokens. A list of common aggregation mechanisms can be found in Table 2.

Figure 4 represents the basic definition of a crowdsourced contest: an *infinity generator* allows individuals to accept the *Submit* task at any moment; then, in order to decide a winner contribution, a review phase is used to score them by computing the average opinion of N workers.

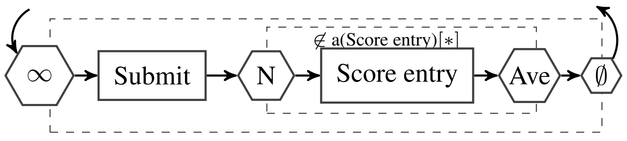


Figure 4: Basic description of a crowdsourced competition: An unknown number of submissions are independently reviewed. In the figure, an average of N evaluations is used to rank submissions.

As for token management, the token in the parent process points to the structure until the aggregation mechanism returns a value. Generated tokens are executed in parallel instances of the same sub-process. This particularly affects our worker management model. In Section 3.1, we introduced three functions on tasks that returns workers involved in their resolution. Due to the parallel execution of the sub-process, these sets only returns workers of the current instance. Nevertheless, it might be useful to also consider workers in parallel executions. See Figure 4 for an example. In order to avoid multiple reviews from the same individual, a restriction on workers must consider parallel review tasks.

Generator icon	**Description**
N	Create N copies of the token.
$N+$	Create N copies of the token. The generator will create more tokens if requested.
∞	The first task in the sub-process is always available to claim. A deadline specifies when the platform should stop offering the task.
For each	It is used to independently process elements in a list generated by the crowd.

Table 1: Examples of generators used in crowdsourcing processes.

Aggregation icon	**Description**
MV	Majority vote. Only the most repeated answer will be considered.
$N\%$	Returns solutions repeated by more than $N\%$ individuals. Optionally, it can request more tokens if there is no consensus.
$\emptyset$	There is no aggregation. It returns a list of contributions.

Table 2: Examples of aggregation mechanisms used in crowdsourcing processes.

In order to tackle this issue, we define the operator $[*]$ that requests a higher-level point of view. By attaching the operator $[*]$ to one of the sets, we are looking at the task from the parent process. So, we can use the condition $workers\ not\ in\ a(\text{Score entry})[*]$ to avoid the duplication of reviewers in Figure 4. The operator can be stacked, getting access to a broader point of view: subsequent parent processes will be considered.

LOOP STRUCTURE Even though iterative processes can be modelled by a proper combination of XOR operators, this approach does not allow us to properly limit the number of iterations or measure changes between iterations. Consequently, we introduced a new mechanism to properly define iterative processes. Besides, this will allow us to improve the worker management of tasks inside a loop.

As in the MapReduce Structure, the LOOP Structure contains a sub-process that will be executed as many times as needed until an *exit condition* is satisfied. As an example, Action-Verification units presented in (Muntés-Mulero et al. 2013) are described using our model in Figure 5. An Action-Verification Unit is a pattern used in tasks where human review is required to measure and ensure quality. In this figure, the *Action* task is repeated until the *Verification* task accepts the contribution or the sub-process is executed at least 3 times.

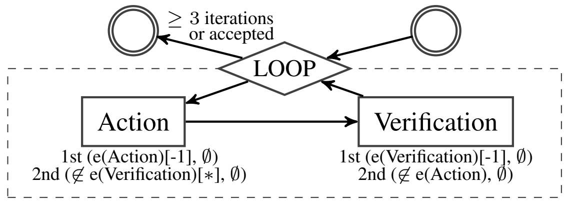

≥ 3 iterations or accepted
LOOP
Action
Verification
1st (e(Action)[-1], ∅)
2nd (∉ e(Verification)[∗], ∅)
1st (e(Verification)[-1], ∅)
2nd (∉ e(Action), ∅)

Figure 5: In this process, Action and Verification tasks are performed iteratively. Note that nobody can review an action performed by themselves, and reviewers are not allowed to contribute to following actions.

The LOOP structure also inherits the $[*]$ operator from the MapReduce. The absence of the operator represents workers in the current iteration. By using $[*]$, we will consider workers from all iterations. We can restrain the scope by replacing $*$ with a finite set of non-positive integers. 0 represents workers involved in the current iteration; -1 represents workers involved in the previous iteration; etc.

3.3 Workflow Transformation

We already described workflows that adapt to the profile of involved individuals. Nevertheless, Turkomatic (Kulkarni, Can, and Hartmann 2012) showed that changes can also come from decisions made by the crowd. And due to external necessities (such as deadlines), process designers may need to adapt the workflow while it is being performed by the crowd. CrowdWON models those unusual changes in the process with a map φ between the places of two workflows. When a transformation is requested (by a computer task or the process administrator), every token pointing to a place p will be now pointing to the place $\varphi(p)$. If a token is pointing to a task t, the platform will wait until it naturally arrives at a place. Since tasks are usually performed by humans, we should not discard their contributions without prior warning.

The description of a generic process in Turkomatic can be found in Figure 6. An initial task allows individuals to decide if the subsequent task must be completed at once or needs to be split into smaller tasks. If so, they provide a separation of the task and the *Request map?* task request the workflow transformation φ. Note that workflow (a) is now a sub-process in workflow (b), allowing further subdivisions of the problem.

4 Deadline Management

In many different scenarios, the production must be finished before a certain date. Deadline management is the use of any mechanism to ensure the process is completed within a certain time frame. Previous mechanisms in crowdsourcing were based on manual modifications of the process. All these manual modifications can be formalized with the use of our workflow maps. Nevertheless, CrowdWON provides a mechanism to automatize those transformations. Note that we already included time conditions in the selection of workers. In order to increase the capabilities of our deadline management, we introduce the following timed events:

- **Deadline of submission**. Individuals have an amount of time to complete claimed tasks.
- **Deadline of claim**. Individuals have a limited amount of time to claim a task.

The amount of time available can be fixed at the design phase of the process, or it can be computed when a token arrives at a node. Deadlines will be graphically represented with an extra circle, representing a clock. In general, the platform will perform an alternative path (represented with dashed arrows) when the deadline is met. In a more complex scenario, one may additionally trigger a workflow transformation.

An example of deadline management is depicted in Figure 7. Individuals have 8 hours to submit a contribution to the *Post edition* or *Review* task. If they spend more than that, the platform will revoke the claim and republish the task (the dashed arrow sends the token to the place before). The only mandatory task is the final edition: If the process must end before 8 hours, the review phase will be skipped. If the process did not start and we have less than 24 hours to complete the process, then only the final edition will be performed and a notification will be sent to the administrator of the process.

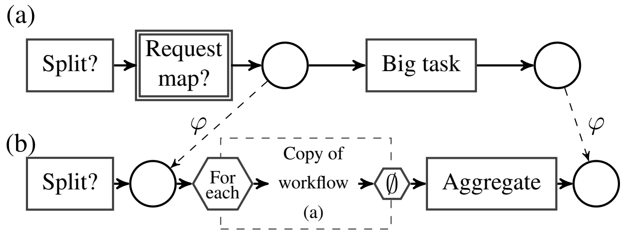


Figure 6: As in Turkomatic (Kulkarni, Can, and Hartmann 2012), workers may request to split a task in easier chunks.

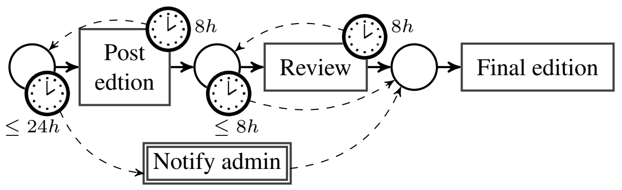


Figure 7: Deadline management in a post edition process. Individuals in the post edition and review have 8 hours to submit a task. Tasks may be skipped depending on the remaining time.

5 Examples

To show the potential of our model, we are going to describe two complex crowdsourcing processes. We chose one of the

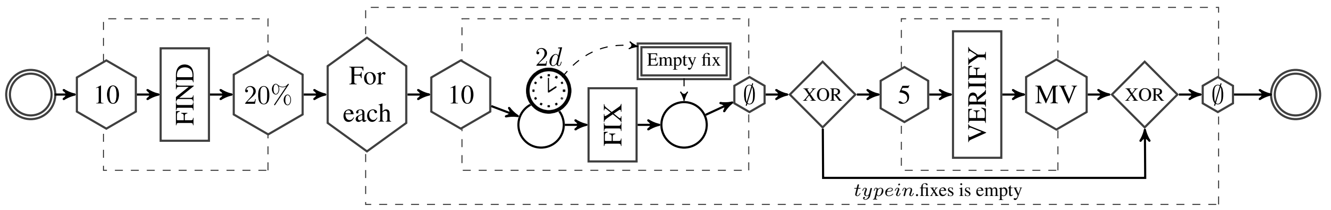


Figure 8: Crowdsourcing process used in Soylent (Bernstein et al. 2010). Given a text, the crowd find sentences that can be shortened. The group also proposes 10 shorter versions for every sentence. Consensus is reached by a simple voting system.

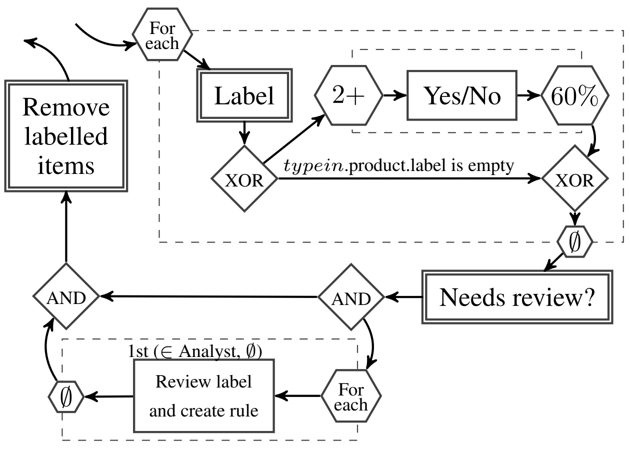


Figure 9: Process used by Chimera to label a list of products using machine learning, crowd workers and analysts. In the model, *Analyst* is a fixed group of individuals hired by the company.

first collaboration patterns present in crowdsourcing as an example of marketplace-like tasks. Then, we describe an industrial process that combines the power of machine learning algorithms, in-house analysts and crowd workers.

The FIND-FIX-VERIFY is a pattern designed by Soylent (Bernstein et al. 2010) in order to reduce the length of a text. It consists of three phases: In the FIND phase, 10 individuals flag sentences or paragraphs as *potentially reducible*. If more than 20% of individuals send the same flag, then it is considered in the following phases; The FIX and VERIFY phases are executed for every discovered flag in the FIND phase. First, 10 users propose an alternative text with the same meaning. Finally, 5 individuals vote for the most accurate alternative. The original text is replaced by the most voted. We assume there is a time-constraint in the resolution of the process. In order to ensure finishing on time, we enforce the FIX phase to end in at most 2 days. After that period, the number of proposed fixes might be less than 10 or even none. In the latter, we assume that the sentence is already correct or too complex for the available crowd. This process is described in Figure 8 using our model.

Chimera is a crowdsourcing process designed by Walmart Labs (Sun et al. 2014) to classify a large set of products. Given the size of the set, using hired analysts to label products, or review the quality of machine learning algorithms, is unfeasible. But the crowd can help to reduce the workload of analysts. In Figure 9, one can find a graphical representation of the process. The whole process is repeated until all the products are labeled. In the first phase, all products are classified by a combination of machine-learning algorithms and a set of handmade rules. If the algorithm is not sure about the label of a product, it remains unlabeled and passes directly to the next phase. If the algorithm recommends a label, then it is reviewed by the crowd. The total number of reviewers depends on initial consensus: only three crowd workers are involved if the two first reviewers do not agree on the correctness of the classification. After that, some of the products will be examined in a second review. This second phase is done by hired professional analysts: They check the quality of the classification and create new rules for the machine-learning algorithm if needed.

6 Conclusions and Future Work

The research on crowdsourcing has been mostly focused on study individual tasks properties, and the application of crowdsourcing to the resolution of real complex problems has been limited to a few examples. However, we have observed a real industrial interest in crowdsourcing for solving complex real world processes. To cover this industrial need, the design of tasks takes a global perspective: deadline management does not only affect particular tasks, but the whole process; Individuals can participate in multiple tasks of the same process, but it may be need to limit those contributions, etc. CrowdWON offers to the industry a graphical language able to represent a possible workflow for any complex industrial process.

For future work, we want to further investigate the best procedure to describe reward mechanisms in our model. At the moment, it is easy to implement usual financial incentives, but we still need to study how our workflow transformations can affect rewards. In the near future, we will set up an scenario where processes can adapt to the current expertise of the individuals–instead of only allowing experts.

It still remains to study if our modelling language can also help in the design of mechanisms to measure the skill acquisition of crowd workers. Our new contribution-based policy in the selection of workers allows for communication between individuals contributing to the process. Therefore, it is possible to give feedback to individuals, so they can improve their contribution. Moreover, this also sets a first step to group-based crowdsourcing. Groups could claim complex tasks, collaboratively create sub-processes to solve the problem and assign sub-tasks to their colleagues or the crowd–if the team does not have expertise in an area.

Acknowledgments This work is partially supported by Secretaria de Universitats i Recerca of Generalitat de Catalunya (2013DI062), by the Ministry of Science and Technology of Spain under contract TIN2012-34557 and by the BSC-CNS Severo Ochoa program (SEV-2011-00067).

References

Ahmad, S.; Battle, A.; Malkani, Z.; and Kamvar, S. 2011. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, 53–64. New York, NY, USA: ACM.

Bernstein, M. S.; Little, G.; Miller, R. C.; Hartmann, B.; Ackerman, M. S.; Karger, D. R.; Crowell, D.; and Panovich, K. 2010. Soylent: A word processor with a crowd inside. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, 313–322. New York, NY, USA: ACM.

Dawis, E.; Dawis, J.; and Koo, W. P. 2001. Architecture of computer-based systems using dualistic petri nets. In *In proceeding of 2001 IEEE International Conference on Systems, Man, and Cybernetics*.

Hassan, U. U.; O'Riain, S.; and Curry, E. 2013. Effects of expertise assessment on the quality of task routing in human computation. In *2nd International Workshop on Social Media for Crowdsourcing and Human Computation*, 1–10.

Kittur, A.; Smus, B.; Khamkar, S.; and Kraut, R. 2011. Crowdforge: crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, 43–52. New York, NY, USA: ACM.

Kittur, A.; Khamkar, S.; André, P.; and Kraut, R. 2012. Crowdweaver: Visually managing complex crowd work. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, 1033–1036. New York, NY, USA: ACM.

Kulkarni, A.; Can, M.; and Hartmann, B. 2012. Collaboratively crowdsourcing workflows with turkomatic. In *In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12)*, 1003–1012. New York, NY, USA: ACM.

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2010. Turkit: Human computation algorithms on mechanical turk. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, 57–66. New York, NY, USA: ACM.

Mason, W., and Watts, D. J. 2009. Financial incentives and the "performance of crowds". In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, 77–85. New York, NY, USA: ACM.

Massolution. 2013. The crowd in the cloud: Exploring the future of outsourcing. Technical report, massolution.com.

Muntés-Mulero, V.; Paladini, P.; Manzoor, J.; Gritti, A.; Larriba-Pey, J.-L.; and Mijnhardt, F. 2013. Crowdsourcing for industrial problems. In Nin, J., and Villatoro, D., eds., *Citizen in Sensor Networks*, volume 7685 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 6–18.

Petri, C. A. 1966. *Communication with automata*. Ph.D. Dissertation, Universitt Hamburg.

Sun, C.; Rampalli, N.; Yang, F.; and Doan, A. 2014. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB* 7(13):1529–1540.

Thorpe, D. 2013. Crowdfunding entrepreneurs predict more good in the world in 2014. *Forbes*.

van der Aalst, W. M. P.; Rosemann, M.; and Dumas, M. 2007. Deadline-based escalation in process-aware information systems. *Decis. Support Syst.* 43(2):492–511.

van der Aalst, W. M. P. 1998. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers* 8(1):21–66.