SmartCrowd: A Workflow Framework for Complex Crowdsourcing Tasks

Tianhong Xiong[1], Yang Yu[1], Maolin Pan[1], Jing Yang[1]

[1] Sun Yat-sen University, Guangzhou 510006, China
{yuy,panml}@mail.sysu.edu.cn
tianhongxiong@qq.com
yangj357@mail2.sysu.edu.cn

Abstract. Over the past decade, a number of frameworks have been introduced to support different crowdsourcing tasks. However, complex creative tasks have remained out of reach for workflow modeling. Unlike typical tasks, creative tasks are often interdependent, requiring human cognitive ability and team collaboration. The crowd workers are required not only to perform typical tasks, but also to participate in the analysis and manipulation of complex tasks, hence the number and execution order of tasks are unknown until runtime. Thus, it is difficult to model this kind of complex tasks by using existing workflow approaches. Therefore, we propose a workflow modeling approach based on state machine to design crowdsourcing model that can be translated into SCXML code and executed by an open source engine. This approach and engine are embodied in SmartCrowd. Through two evaluations, we found that SmartCrowd can provide support for complex crowdsourcing tasks, especially on creative tasks. Moreover, we introduce a set of basic design patterns, and by employing them to compose complex patterns, our framework can support more crowdsourcing research.

Keywords: Crowdsourcing, State Machine, Workflow, Complex Tasks, Creative Tasks, Design Patterns.

1 Introduction

Crowdsourcing can be defined as an emerging computing paradigm that uses advanced internet technologies to harness the efforts of a virtual crowd to perform specific organizational tasks [1]. The requesters (or employers) publish tasks through crowdsourcing marketplaces (such as Amazon Mechanical Turk [2]), which are completed by crowd workers (or employees). The typical tasks are self-contained, simple and repetitive, crowd workers can directly complete work without worrying about how their contributions affect others [3], such as identifying objects in a photo or video, de-duplicating data, transcribing audio recordings, or researching data details. Conversely, the complex tasks, especially about creative tasks [4, 5], are often interdependent, requiring human cognitive ability and team collaboration [6]. Such tasks can not be solved directly and need to be decomposed into subtasks. And the workers

are required not only to perform typical tasks, but also to participate in the analysis and manipulation of complex tasks, including judgment and decomposition of tasks. In this situation, the number and execution order of tasks are unknown until runtime. Consider for example the task of writing a short article, this is a creative task that involves many subtasks and crowd workers, such as deciding structure of the article, deciding how other parts of the article need to write, deciding which sections need to be decomposed further, taking pictures and laying out the document, and so forth. Furthermore, changing one part of the article may trigger changes to the overall plot and vice versa, hence each subtask also needs to coordinate in order to avoid redundant work and to make the final version of the article coherent. However, this kind of complex creative tasks has remained out of reach for workflow modeling, and it will be resisted by the features of these tasks.

Therefore, we propose a workflow modeling approach to support complex crowdsourcing tasks. More specifically, our approach extends statecharts [7] based on state machine to model crowdsourcing tasks, and the model can be translated into State Chart XML (SCXML) which is proposed by World Wide Web Consortium (W3C) to combine statecharts semantics with XML syntax [8]. And we integrate the modeling approach and an open source Apache engine called Commons SCXML [9] to build a framework called SmartCrowd.

In summary, we make the following contributions:

(1) We present a visual modeling approach, which combines graphical symbols of state machine with SCXML. Moreover, by introducing the concept of task instance tree, we can monitor the running process of crowdsourcing.

(2) We provide the SmartCrowd to support model design and implementation of crowdsourcing tasks.

(3) We introduce a group of basic design patterns based on existing crowdsourcing literatures. We can employ them to compose complex design patterns that can be supported by SmartCrowd.

The remainder of the paper is organized as follows. In section 2 we compare our work with related work. Section 3 introduces the core conception for our approach, and describes our modeling approach in detail. The structure of SmartCrowd will be shown in Section 4. And Section 5 reports two evaluations about our approach. Section 6 concludes with a brief description of future work.

2 Related Work

Over the past decade, crowdsourcing has received a lot of attention, and the approaches and frameworks that support crowdsourcing have attracted the interest of many researchers.

[10] describes a new toolkit Turkit that executes JavaScript files with APIs, the programmer need to write JavaScript code for deploying iterative tasks to Amazon Mechanical Turk (MTurk). [11] introduces AutoMan that is a programming system based on the Scala programming language. By automatically managing quality control and budgeting, it can drive the tasks to continue to do the computation until a

desired confidence level is achieved. However, similar to Turkit, complex task must be manually decomposed into smaller tasks by programmers. [12] proposes CrowdDB that provides a declarative approach to solve problems by using an extension of SQL called CrowdSQL. It is good at dealing with tasks that are of pure data processing nature, such as subjective comparisons and ordering of datasets. However, it is limited to pure data processing problems.

[13] presents Crowdforge that is inspired by the MapReduce distributed computing approach, and it decompose the complex task into three types of subtasks, including partition, mapping and reducing. Yet it cannot support recursion of whole task [15]. [14] presents Jabberwocky, a crowd computing framework that contains three components: the Dormouse is a human and machine resource management system, the ManReduce is a parallel programming framework, and the Dog is a high-level programming language. The programmer can handle tasks with a support of cross-platform programming languages, routing tasks from one platform to another one. However, it is assumed that task requesters (not workers) will determine how tasks are broken down in all cases, hence it is not suitable for creative tasks. [15] introduces a tool called Turkomatic, which emphasizes the decomposition, resolution and merging of solutions. It enables the requesters and workers to collaborate on execution of tasks. Specifically, the workers execute tasks following the instructions of requesters. However, this approach forces the workers must follow the instructions of requesters, and it lacks support for creative tasks that are determined by workers.

These approaches mentioned above are dependent on different programming languages, different types of tasks require different programs to match, involving a lot of hard code. Recently, some workflow-based approaches have been introduced for crowdsourcing research. [17] describes CrowdLang, a programming framework that defines a set of operators, including Reduce, Aggregate, Multiply, and so on, and it can employs these operators to compose complex patterns. [18] presents CrowdSearcher, a search paradigm that defines a query language as a bridge between input and output, which can improve the quality of search results in complex seeking tasks. And the most recent version of CrowdSearcher [19] introduces the modeling concept, and defines a set of task types (eg. labeling, liking, sorting, classifying, grouping) to solve different problems. Moreover, it can support some specific crowdsourcing patterns. However, it restricts crowdsourcing tasks to simple operations, and the workers are limited to simple and repetitive tasks. [20] provides Crowd Computer that adopt a business process modeling approach based on BPMN[21] to support crowdsourcing workflow. Concretely, it introduces BPMN4Crowd, an extension of BPMN that can model crowdsourcing process. And it extends the standard workflow to relax the constraints imposed on the task assignment, so that it can assign tasks to workers through different tactics (e.g., marketplace, contest, auction, mailing list). However, these approaches mentioned above are rigid that all of tasks still must be decomposed by requesters in advance and executed in a given order, thus, they are also not suitable for creative tasks described previously.

3 Modeling Approach

3.1 Statechart

As we know, the traditional finite state machine cannot model large and complex problems because of three main reasons. Firstly, it has no hierarchy or module concepts. Secondly, there may be a "state explosion" problem. Thirdly, it cannot describe concurrency. Therefore, David Harel presented statecharts, an extension of finite state machine that provides a broadcast mechanism for communication between concurrent components [7]. It conquers the limitations of traditional finite state machine while inherits its main strengths.

In general, a statechart (state machine) contains State, Event, Condition, Action and Transition. The state is a description of the status of a system that can execute action. Transition is a kind of relationship between two states, when a condition is fulfilled or an event is received, the action of transition will be executed, and the source state will transfer to the target state. A transition is often expressed like this: event [condition]/action. Fig. 1 shows an example of statecharts. And we invite the reader who wants to get more information about statechart to read the literatures [7, 8].

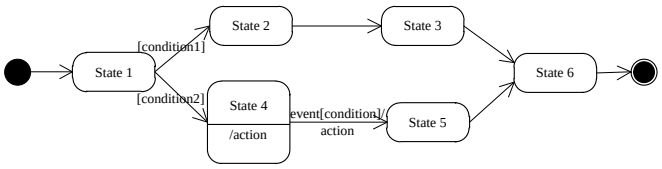


Fig. 1. An example of statecharts.

3.2 Approach Features

In essence, a complex crowdsourcing task often involves one or more subtasks, some of which may continue to be decomposed into subtasks, this may lead to a large number of tasks in the crowdsourcing solution. So how to model these tasks, how to coordinate the interrelated subtasks, these are the key questions for crowdsourcing. In our approach, each task or subtask includes multiple steps from start to end, and each step is treated as a certain state. Hence, all steps of a task can be mapped to different states, which form a complete lifecycle from the start state to the final state. Clearly, lifecycles of different task types are different. Therefore, we consider a crowdsourcing task as a process, which consist of several subprocesses (subtasks). And we employ state machines to model task types as the appropriate building blocks of crowdsourcing model, and then we aggregate them to yield the crowdsourcing result. The approach is described in detail below.

Firstly, we extend action of state machine to meet the requirements of task. Specifically, we add multiple structured elements into action, such as name, attribute, event, data model, and so on, so that we can configure our crowdsourcing solution in these elements, including task decomposition, number of workers, subtasks model, condition of next step, evaluation of results, and so on. In this way, we can not only assign

simple tasks to workers, but also arrange workers to participate in the execution of complex tasks. When the task is in different states, the workers are required to participate in different work items, including voting for task decomposition, deciding the next step, processing and evaluating the results of other workers. These structured elements will be embedded into the executable content (EC) of state machine in SCXML. The Fig. 2 indicates the EC can be executed by state and transition of state machine. When the state machine takes a transition, it executes the EC of exit action in the states it is leaving, followed by the EC in the transition, followed by the EC of entry action in the states it is entering.

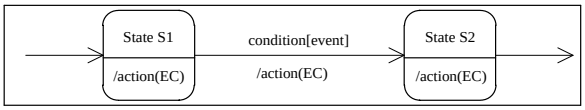


Fig. 2. The EC of state machine.

Secondly, the events of state machine are usually used for internal communication and triggering state transition. Since the statecharts and SCXML provide a flexible mechanism for communication between state machines, the events are used not only for a single state machine, but also for coordination between multiple state machines, this means that different tasks can communicate with each other by sending and receiving events. Therefore, our approach can support interdependent tasks, not just independent tasks.

Lastly, we introduce the task instance tree to monitor the running of crowdsourcing in our framework. As we mentioned previously, a crowdsourcing solution often involves one or more tasks. When it is running, some tasks (called parent tasks) can be broken down into several subtasks (called child tasks), and all tasks will be instantiated in chronological order. A parent task will trigger its child tasks, and the child tasks will continue to trigger their child tasks, and go on. Finally all task instances will form a tree structure called task instance tree. The Fig. 3 shows some examples of task instance trees, Fig. 3 (a) shows that a task is complex, while it is non-decomposable, such as Macro Tasks [16]. Thus, only one node in task instance tree. Fig. 3(b) shows a complex task that can be decomposed into subtasks. When a task node is finished, it sends an event to its parent node as the signal to finish. It is worth noting that our approach also supports a crowdsourcing model similar to tournament form, Fig. 3(c) shows it as below. Note that the task instance tree of a crowdsourcing solution may not be fixed, when workers are asked to participate in the task decomposition, different decomposition schemes will lead to different instance trees.

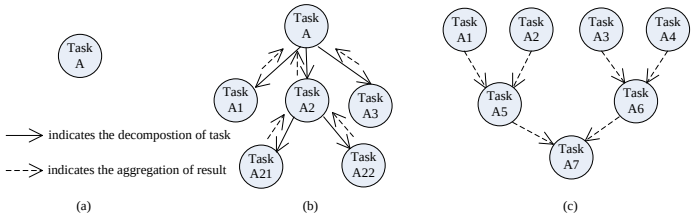


Fig. 3. Some examples of task instance trees.

4 Framework

We integrate our modeling approach and Commons SCXML to build SmartCrowd. The Commons SCXML is an implementation aimed at creating and maintaining a Java SCXML engine that is capable of executing a state machine defined using a SCXML document, while abstracting out the environment interfaces [9]. The SmartCrowd mainly includes design, compilation, execution and management & maintenance. Fig. 4 indicates the architecture of SmartCrowd.

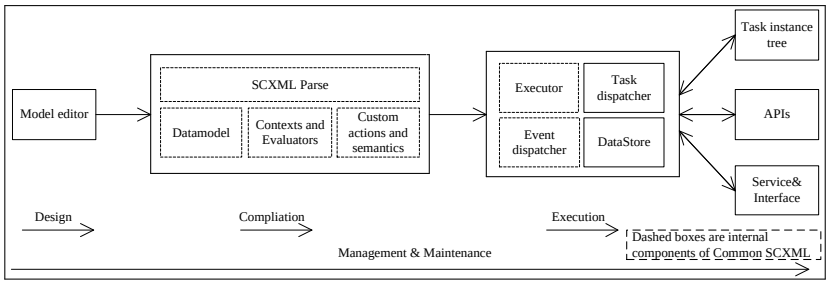


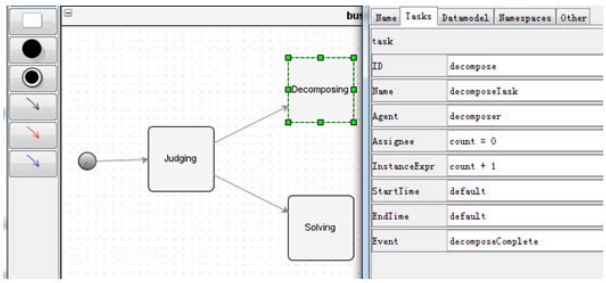
Fig. 4. The high level architecture of SmartCrowd.

Design. We provide a model editor for users to design their crowdsourcing solution. As we mentioned previously, we add several structured elements to support crowdsourcing solution, such as name, attribute, event, data model, and so on. And the model editor can verify whether the model conforms to the syntax rules of the XML specification, and export the SCXML code document to compilation. Fig. 5 shows one part of crowdsourcing design process and corresponding SCXML code.

Compilation. In this stage, the SCXML code document that defines the state machines will be parsed into Commons SCXML Java object model. More specifically, the elements of data and temporary variables will be defined in the Data model. The Contexts and Evaluators can support expression evaluation and context explanation. In addition, the Custom actions and semantics provide an extension of the Commons SCXML for specialized uses, such as supporting custom elements and custom processing logics.

Execution. The Executor, a SCXML engine that can drive the crowdsourcing model to run. And the Task dispatcher can support the task assignment. The Triggering events (event dispatcher) will deliver events for state machines. The DataStore is a data repository that provides the required data information for other parts of the framework.

Management & Maintenance. When a task is in a certain state that needs workers to participate in a work item, such as voting, the work item will be instantiated as a micro task that can be published to traditional crowdsourcing markets (such as Amazon Mechanical Turk) by APIs and task templates. The task instance tree provides a visual UI to show the running process of crowdsourcing. And the Services&Interfaces will be used for custom page, functional interface and web services, etc. Note that the two-way arrows indicate these parts can interaction with Execution.




```
<task id="decompose" name="decomposeTask" brole="decomposer">
    <principle method="Allocate" distributor="Round"/>
    <callback on="complete" event="decomposeComplete"/>
    <documentation>Decompose task let decomposers decompose crowdsourcing task
    <param name="taskName"/>
    <param name="taskDescription"/>
</task>
```

Fig. 5. One part of crowdsourcing design process and corresponding SCXML code.

5 Evaluation

The SmartCrowd places an emphasis on how to support different crowdsourcing tasks. On the one hand, it has a powerful flexibility, allowing the users to focus on the design and optimization of solution, and do not need to develop an additional, dedicated prototype, especially without hard coding. On the other hand, it has enough adaptability to support complex crowdsourcing tasks which are composed by several basic design patterns. In this section, we report two evaluations on exploring whether or not this framework has achieved the desired goals.

More specifically, the first evaluation describes the design and implementation of complex creative tasks. It demonstrates the capability of SmartCrowd for crowdsourcing. In the second evaluation, we show a set of basic design patterns, which can be used to compose more complex design patterns to accommodate different types of crowdsourcing problems, and our approach can also support these complex patterns. This evaluation suggests that SmartCrowd has a great adaptability.

5.1 Crowdsourcing writing

Here, we employ the crowdsourcing writing task to illustrate the capability of our framework. Crowdsourcing writing is chosen as a test domain because it is a complex creative task [3], and it can further exploit human cognitive ability and collaborative innovation. For these reasons, we chose the MapReduce crowdsourcing writing process from [13] as example to test our approach. Here, we show how our framework supports it without hard coding.

The MapReduce method builds on the general approach to distributed computing, it brokes down a complex problem into a sequence of simpler subtasks, as shown in Fig. 6.

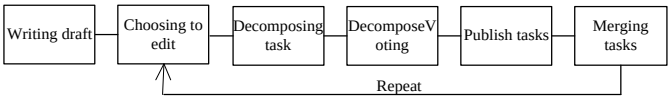


Fig. 6. The MapReduce write processes.

For a better readability, the design model only show the key actions and conditions, and "[]" means the condition, "/" represents the executable content of action in state and transition.

The MapReduce model will judge (vote) for drafts or outline (section headings) in state "Judging", as shown in Fig. 7(a), if the outline is complex, and can not be done by workers directly, it enters the state "Decomposing". According to the decomposition strategy, several workers independently decompose the task into subtasks, and then there will be multiple alternative decomposition schemes. Thus in state "DecomposeVoting", the workers vote for the best decomposition scheme, and the framework assigns the number of corresponding subtasks to variable "DecTaskCount". Different decomposition schemes have different number of subtasks, so the number of subtasks here is not known until runtime. And then the SmartCrowd instantiates new state machines for subtasks by executing the "NewSubtasks(n)" in action, the parameter n represents the number of subtasks. Note that since all tasks, including subtasks, must first be judged whether they need to be broken down in this scenario, we employ the same model for these tasks, namely the state machines of subtasks are the same as their parent state machine, it makes our design more succinct and effective. After that, the parent task enters the state "Waiting" to wait for results of its subtasks, when all subtasks are finished, the condition "DecTaskCount==SubtaskFinishCount" is true, it enters the state "Merging" and aggregates the results to produce a final result, and sends an event as a finish signal to parent task. When the task is determined as simple, it do not need to decompose, and is directly allocated to multiple workers to solve in state "Solving". After all workers finish their jobs, the condition "TCount==WorkerFinishCount" is true, a group of workers vote for the best answer in state "SolveVoting", and finally send an event to inform the parent task. The Fig. 7(b) shows one of the task instance trees of MapReduce model, states of state machine are color-coded to indicate their status: in progress (red, underlined), finished (green). Different outlines of article yield different task instance trees, and even each execution might generate a different task instance tree since the task decomposition is uncertain until runtime. For example, the task A is decomposed into 3 subtasks, while its subtask A2 is decomposed into 2 subtasks (Fig. 7(b)), the different decomposition schemes voted by workers result in this result.

Through this case, SmartCrowd allows workers to participate in the task decomposition, supporting the collaboration between requesters and workers. Through the task instance tree, we can easily monitor the running process of crowdsourcing, including tasks types, the status of tasks, the number of task instances.

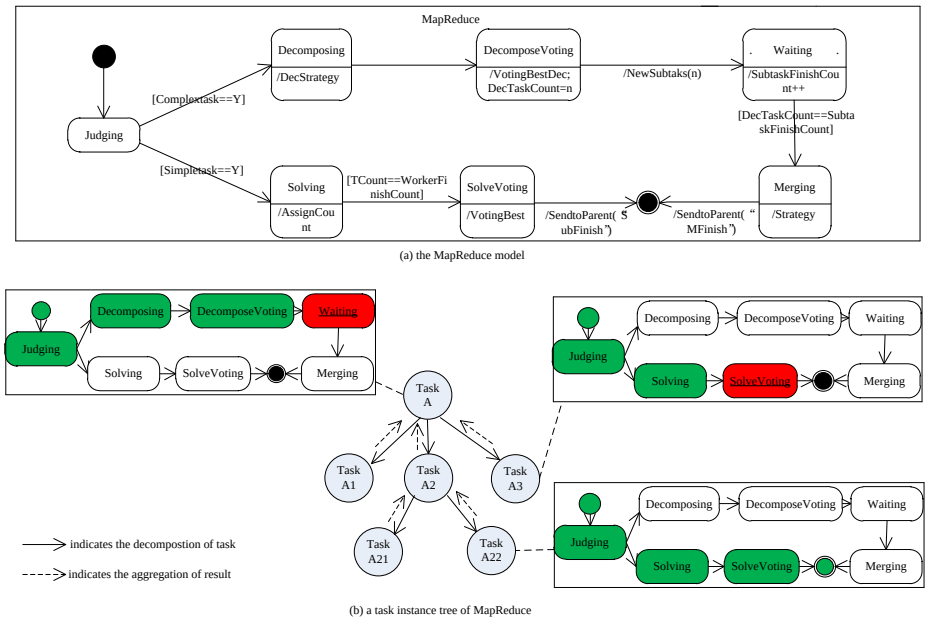


Fig. 7. The MapReduce model and one of its task instance trees

It is worth noting that, when workers are required to determine the task decomposition with no supervision, the task was likely to be constantly decomposed and iterated, resulting in excessive task instances. It may cause tasks to get out of control, thus SmartCrowd provides three ways to avoid this kind of situation, and the first one is to allow the requester to end task directly, which is the highest priority, the second allows the workers to decide when to end the task, within a set period of time. And the last one is to stop task after a predefined number of task decomposition rounds, namely preset depth of the task instance tree. And limited to the pages of paper, detailed SCXML code exported by our framework for this crowdsourcing writing case is displayed on the web site [22].

5.2 Adaptability Study

The Amazon Mechanical Turk team has indicated that the investigations on crowdsourcing should be designed, conducted, and published in a manner such that the research experiments can be repeated, potentially yielding standard design patterns and methods to achieve high quality, consistent results for a variety of human computation tasks [23]. So far, crowdsourcing design pattern has been explored by previous work [19, 24], but still has remained out of reach. We reviewed the existing literatures, including many empirical studies and reports [6, 25], and introduce four basic crowdsourcing design patterns based on them. Furthermore, we employ SmartCrowd to express these patterns, and for a better readability, we report them using task instance trees. The Fig. 8 shows these basic design patterns.

(1) The collection pattern, as shown in Fig. 8(a), in which the number of subtasks decomposed is fixed, and each task is independent of each other, with no interaction. It meets the requirements of a typical task.

(2) The contest pattern will choose a best answer for task, similar to tournament form, see Fig. 8(b).

(3) The collaboration pattern usually appears in creative crowdsourcing work. This pattern can arrange workers to participate in the execution of complex tasks. Especially in task decomposition, the number of subtasks is unknown until runtime, hence different tasks may have different number of subtasks.

(4) The interaction pattern is often used for creative tasks. In our approach, the parent task can communicate with its child tasks by sending and receiving events. However, the interaction pattern has a powerful feature that subtasks with the same parent task can communicate with each other by sending and receiving events, as shown in Fig. 8(d).

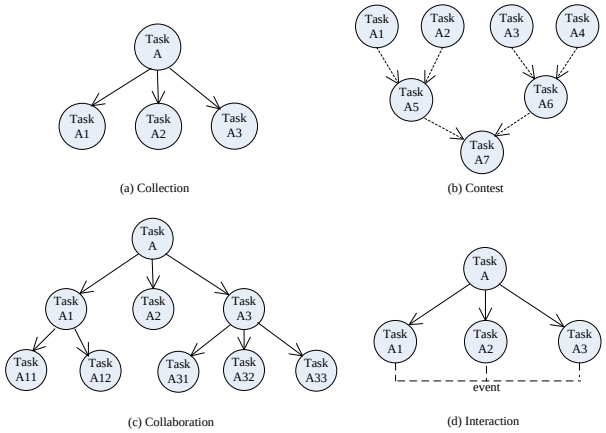


Fig. 8. The basic design patterns

In general, the complex design model is composed by different basic design patterns, such as crowdsourcing writing model is shown in section 5.1, which deployed some basic patterns, including collaboration and contest (voting for the best scheme). This evaluation shows that our framework can support complex patterns that are composed by basic design patterns. Note that we do not think these basic design patterns to be complete. In addition, we have tested a variety of crowdsourcing cases, such as purchasing decisions, evaluating a design and so on, and more detailed information can be seen in [22].

6 Conclusion

In this paper, we present SmartCrowd based on state machine and Common SCXML to support crowdsourcing research, especially in complex creative tasks. It allows researchers to put more energy on the solution formulation rather than on the devel-

opment and maintenance of system. Due to the use of the state machine and Common SCXML, the users of our approach also need to understand the state machine and the specification of SCXML.

There are a number of directions we are exploring for future work. We will continue to exploit new design patterns of crowdsourcing, and the performance analysis of crowdsourcing model is also a part of the follow-up work.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No.2017YFB0202200; the National Natural Science Foundation of China under Grant No.61572539; the Research Foundation of Science and Technology Major Project in Guangdong Province under Grant Nos.2015B010106007,2016B010110003; the Research Foundation of Science and Technology Plan Project in Guangdong Province under Grant No.2016B050502006;

REFERENCES

1. Saxton, G. D., Oh, O., & Kishore, R. (2013). Rules of crowdsourcing: Models, issues, and systems of control. Information Systems Management, 30(1), 2-20.
2. Amazon Mechanical Turk. https://www.mturk.com/, last accessed 2018/5/10.
3. Kim, J., Sterman, S., Cohen, A. A. B., & Bernstein, M. S. (2017, February). Mechanical Novel: Crowdsourcing Complex Work through Reflection and Revision. In Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (pp. 233-245). ACM.
4. Orlowska, M. E. (2015). Challenges for Workflows Technology to Support Crowdsourcing Activities. International Conference on Business Intelligence and Technology, Bustech (pp.88-92).
5. Kim, J., Cheng, J., and Bernstein, M. S. Ensemble:Exploring complementary strengths of leaders and crowds in creative collaboration. In Proc. CSCW, ACM (New York, NY, USA, 2014), 745–755.
6. Malone, T. W., Laubacher, R., & Dellarocas, C. (2010). The collective intelligence genome. MIT Sloan Management Review, 51(3), 21.
7. Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of computer programming, 8(3), 231-274.
8. SCXML. https://www.w3.org/TR/scxml/, last accessed 2018/6/10.
9. Apache Commons SCXML. http://commons.apache.org/proper/commons-scxml, last accessed 2018/6/15.
10. Little, G., Chilton, L. B., Goldman, M., & Miller, R. C. (2009, June). Turkit: tools for iterative tasks on mechanical turk. In Proceedings of the ACM SIGKDD workshop on human computation (pp. 29-30). ACM.
11. Barowy, D. W., Curtsinger, C., Berger, E. D., & McGregor, A. (2012). Automan: A platform for integrating human-based and digital computation. Acm Sigplan Notices, 47(10), 639-654.
12. Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., & Xin, R. (2011, June). CrowdDB: answering queries with crowdsourcing. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 61-72). ACM.
13. Kittur, A., Smus, B., Khamkar, S., & Kraut, R. E. (2011, October). Crowdforge: Crowdsourcing complex work. In Proceedings of the 24th annual ACM symposium on User interface software and technology (pp. 43-52). ACM.
14. Ahmad, S., Battle, A., Malkani, Z., & Kamvar, S. (2011, October). The jabberwocky programming environment for structured social computing. In Proceedings of the 24th annual ACM symposium on User interface software and technology (pp. 53-64). ACM.

15. Kulkarni, A., Can, M., and Hartmann, B.Collaboratively crowdsourcing workflows with turkomatic. In Proc. CSCW, CSCW '12, ACM (New York, NY, USA, 2012), 1003–1012.
16. Chittilappilly, A. I., Chen, L., & Amer-Yahia, S. (2016). A survey of general-purpose crowdsourcing techniques. IEEE Transactions on Knowledge and Data Engineering, 28(9), 2246-2266.
17. Minder, P., & Bernstein, A. (2012, December). Crowdlang: A programming language for the systematic exploration of human computation systems. In International Conference on Social Informatics (pp. 124-137). Springer, Berlin, Heidelberg.
18. Bozzon, A., Brambilla, M., & Ceri, S. (2012, April). Answering search queries with crowdsearcher. In Proceedings of the 21st international conference on World Wide Web (pp. 1009-1018). ACM.
19. Bozzon, A., Brambilla, M., Ceri, S., Mauri, A., & Volonterio, R. (2015). Designing complex crowdsourcing applications covering multiple platforms and tasks. Journal of Web Engineering, 14(5-6), 443-473.
20. Tranquillini, S., Daniel, F., Kucherbaev, P., & Casati, F. (2015). Modeling, enacting, and integrating custom crowdsourcing processes. ACM Transactions on the Web (TWEB), 9(2), 7.
21. Object Management Group. 2011. Business Process Model and Notation (BPMN) Version 2.0. Available at http://www.omg.org
22. SmartCrowdCode. https://github.com/rinkako/RenWFMS/tree/master/exampleProcess/Crowdsourcing, last accessed 2018/6/10.
23. Chen, J. J., Menezes, N. J., Bradley, A. D., & North, T. A. (2011). Opportunities for crowdsourcing research on amazon mechanical turk. Interfaces, 5(3).
24. Lofi, C., & El Maarry, K. (2014, July). Design patterns for hybrid algorithmic-crowdsourcing workflows. In Business Informatics (CBI), 2014 IEEE 16th Conference on (Vol. 1, pp. 1-8). IEEE.
25. Quinn, A. J., & Bederson, B. B. (2011, May). Human computation: a survey and taxonomy of a growing field. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 1403-1412).