Journal of Computers Vol. 32 No. 6, 2021, pp. 1-14
doi:10.53106/199115992021123206001

# Complex Task Design Based on Crowdsourcing Using Petri Net

Donglai Fu[1,2*], Yanhua Liu[3]

[1] Software School, North University of China, Taiyuan 030051, Shanxi, China
fudonglai@nuc.edu.cn

[2] Shanxi Province Military-Civilian Integration Software Engineering Technology Research Center, Taiyuan 030051, Shanxi, China

[3] Affiliated Hospital, North University of China, Taiyuan 030051, Shanxi, China
zbliuyh@163.com

Received 12 October 2020; Revised 1 February 2021; Accepted 15 March 2021

Abstract. Design of complex crowdsourcing tasks is a challenging problem. This study presented an alternative method called *CRVA* to address the challenge. The *CRVA* used hierarchical Petri nets to model the issue. The requester first constructed a root net, and skilled workers then refined composite tasks in the root net by sub-nets. The final outputs were composed of a root net and one or more sub-nets. When a composite task was activated, the control turned to the corresponding sub-net; when the sub-net completed its operation, the control passed back to the parent net. The new method improves task design by enabling ordinary requesters to delegate complex design tasks to specialized crowds, and automatically checks the output quality based on Petri nets with precise formal semantics. Results showed the proposed method is not only feasible but also advantageous compared to other decomposition-based methods.

Keywords: crowdsourcing, task design, complex task, Petri net

## 1 Introduction

Since its advent, crowdsourcing has enabled humans to accomplish many tedious tasks, ranging from the SETI@home project which uses internet-connected computers of more than 5.2 million volunteers in the search for extraterrestrial intelligence to reCAPTCHA which collects valuable information by internet users and Foldit which is a revolutionary crowdsourcing computer game enabling you to contribute to important scientific research, in a fast, convenient, and cost-effective manner. Amazon Mechanical Turk (MTurk) [1] is a successful crowdsourcing system that enables individuals or companies to harness collective intelligence from a global workforce to accomplish various tasks, e.g., Human Intelligence Tasks (HITs). Employers (known as requesters) recruit employees (known as workers) for the execution of HITs and reward for their labor. However, MTurk is well-suited to only simple and repetitive tasks.

Recent years, we have witnessed the emergence of a trend toward solving complex tasks via crowdsourcing systems. Clearly, designing complex tasks is far more difficult than designing simple tasks. In general, a complex task implicitly or explicitly consists of multiple subtasks. Thus, it is not easy to decompose a complex task and rearrange its subtasks. In many cases, requesters themselves have no prior knowledge of how to decompose such tasks. Moreover, there is significant inter-dependence among the subtasks; hence, requesters require a complex process such as a workflow for their management. Furthermore, requesters must analyze the workflow specification to avoid errors.

Our hypothesis is that the participation of skilled workers in the design process can guide requesters toward better task design. However, most existing crowdsourcing systems fall severely short in this regard. Although they often include best-practice workflows to guide requesters toward accomplishing

* Corresponding Author

the job, they do not provide mechanisms that can enable requesters and workers to collaboratively define the workflow model, and the model cannot be analyzed by the requesters. In addition, such systems are developed to solve some specific problems, and it is difficult to adapt them to diverse crowdsourcing tasks.

To address the above-mentioned problems, we present a novel and general method that enables both requesters and workers to work collaboratively in order to improve the task design. *CRVA*, our initial prototype, focuses on collaboratively designing the workflow model and analyzing its correctness, which are difficult and important aspects of task design. To this end, Petri Nets are used to model the workflow and qualitatively analyze its logic correctness.

Compared with most reported methods, our method is based on a unique concept whereby the design of complex tasks should be outsourced to specialized workers. Note that this method fully exploits collective intelligence and the idea is consistent with the crowdsourcing objective.

To evaluate our method, we integrated an open-source workflow engine called YAWL and an open-source distributed version control system called Git to build a prototype. Then, we performed a crowdsourcing task related to Web application security testing using the prototype. Thus, we concluded that our approach is not only feasible but also superior to previous methods in certain aspects.

The contributions of this study can be summarized as follows:

To the best of our knowledge, our method is based on a novel concept of outsourcing tedious task designs to skilled workers. Further, the final model is described as a Petri net, which enables requesters to analyze the results. Thus, this method not only benefits unskilled requesters but also enables them to analyze the design quality.

From the view of task design, the entire formal model is defined. The formal model, *CRVA*, describes the workflow and its refinement. This study only focuses on correctness semantics of the workflow. Based on these formal semantics, the specific algorithm called *CRVA-I*, which enables requesters and workers to collaboratively design tasks, is developed.

A case related to crowdsourcing writing is demonstrated. On the one hand, the example shows the entire process of task design. On the other hand, its feasibility is verified and its advantages are illustrated as well. Using the new method, requesters can produce a verifiable workflow specification, which can be checked for its correctness.

The prototype is built to investigate the applicability, advantages and disadvantages of the proposed method in practice. We provide a simple implementation of the current method by combining the workflow engine based on Petri nets for modeling the process of task execution and a distribution version control system for managing the workflow specification. The source code related to the prototype has been released to facilitate future research[1]. Based on the prototype, a complex task related to Web application security testing is performed.

The remainder of this paper is organized as follows. Section II surveys related studies. Section III provides a formal description of the crowdsourcing task design problem. Section IV describes the algorithm for collaborative task design by workers and requesters. Section V presents a case of a crowdsourcing writing task. Section VI discusses the prototype construction based on the proposed method as well as an experiment on Web application security testing. In addition, it compares the current method with previous methods. Finally, Section VII concludes the paper.

## 2 Related Work

Combining the strengths of humans and computers to accomplish tasks that neither can do alone has attracted considerable attention from academia and industry [2]. The idea dates back to the 1960s, with the publication of "Man-Computer Symbiosis" by J. C. R. Licklider [3]. Tim Berners-Lee proposed the concept of a social machine in 2009 and regarded the cooperation between machines and humans as the next direction of Web application development [4]. The term crowdsourcing was coined by Jeff Howe in 2006 [5]. MTurk is a pioneering crowdsourcing system. It has been successfully used to solve simple tasks.

The overall process of completing a crowdsourcing task may be coarsely divided into three phases: the design phase, the online phase, and the conclusion phase [6]. Task design is considerably important for

[1] https://github.com/hhluci/collaboration

ensuring the quality of the result. However, it is not easy to complete the design of a task, especially in the case of a complex task. Decomposition is a natural and popular strategy in task design. Kittur et al. proposed a solution to decompose a complex task on the basis of Map-Reduce mechanisms [7]. In their method, iteration or recursion is not supported and a task designer must specify the execution sequence of the subtasks. Furthermore, task design requires the support of the procedure to solve subtasks rather than decomposition. Little et al. explored an iterative workflow paradigm for solving complex tasks, including image description, copy editing, handwriting recognition, and sorting [8-9]. They improved the quality of the results using an iterative algorithm in which the number of iterations was determined by the budget. However, requesters are required to divide each task by hard code before the task is posted on the third-party crowdsourcing system. MTurk. Dai et al. improved the iterative workflow model from the aspect of workflow control [10]. They could autonomously control the workflows without human intervention and yield better results. Lin et al. proposed the idea of multiple workflows based on the probabilistic graphical model, and dynamically implemented switches between these workflows. Their experiments demonstrated that their method yields better results for named-entity recognition [11]. Bernstein et al. introduced the novel idea of multiple-phase workflow and designed the find-fix-verify crowd programming pattern, which split tasks into a series of generation and review stages for complex crowdsourcing writing [12]. Kulkarni et al. designed a new algorithm, which guides workers by converting large and complex tasks into micro-tasks that are appropriate for crowd markets [13-14].

The above-mentioned solutions aim to solve specific problems and hence cannot meet the requirement of solving diverse tasks. Although Crowd Computer [15] provides extensibility, it is not sufficiently friendly for less skilled requesters. Zheng et al. proposed a general workflow technology using a state machine based on recursive decomposition approaches; by means of this platform, one can develop many types of crowdsourcing applications [16]. Xiong et al. [17] extended Zheng's work by proposing a workflow framework called SmartCrowd for complex crowdsourcing tasks. Wu et al. presented Service4Crowd, a highly flexible and extensible process management platform for crowdsourcing based on service-oriented architectures [18]. They claimed that the platform could provide a one-stop solution for requesters. Inspired by these developments, the present study designs an alternative method for task design. The main difference is that the proposed method does not require requesters to complete tedious design tasks. Instead, we outsource such tasks to skilled workers, which is consistent with the crowdsourcing concept. In addition, a workflow technique base on Petri nets [19] is adopted instead of BPMN [20] or the state machine [21], which enables requesters to check the design quality. To the best of our knowledge, this study is the first to use Petri nets to model the design of crowdsourcing tasks.

## 3 Problem Formulation

Here, we formalize some concepts and processes related to the crowdsourcing task design. In this study, we consider a crowdsourcing system in which employees (called workers) are recruited by employers (called requesters) for the execution of tasks in exchange for a wage (called a reward). The context of the task design is as follows. A requester submits a task to a crowdsourcing system and obtains a workflow model described by Petri nets for the process definition of solving the task with the assistance of workers. Each worker earns money as a reward.

### 3.1 Participants

In this study, three types of participants are considered:

- Requesters consist of a group of persons who post tasks on the crowdsourcing system. Thus, they are modeled as a set $R = \{r_i \mid 1 < i < n\}$, where $r_i$ denotes the i-th requester.
- Workers consist of a group of persons who receive simple or complex tasks from the crowdsourcing system, solve them, and return the results to the system. Thus, they are also modeled as a set $W = \{w_i \mid 1 < i < n\}$, where $w_i$ denotes the i-th worker.
- Machines, which are non-human, usually consist of a series of computing services, such as Web services, that only complete automated tasks. Thus, they are also modeled as a set $M = \{m_i \mid 1 < i < n\}$, where $m_i$ denotes the i-th machine.

### 3.2 Task

A task is a unit of work designed to transform an input into the corresponding output. The type of task may be simple, complex, or automated. A simple task (called an atomic task) can be easily accomplished by a single worker, but a complex task (called a composite task) needs to complete a series of subtasks to achieve its objective. An automated task can be performed only by machines.

Tasks can be defined as a set $T = \{t_i \mid 1 < i < n\}$, where $t_i$ denotes the i-th task. Further, $T = T_{at} \uplus T_{ct} \uplus T_{au}$, where $T_{at}, T_{ct}$ and $T_{au}$ are sets of atomic tasks, composite tasks, and automated tasks, respectively.

### 3.3 Condition

In this study, we assume that there is a priori finite set of competences $C = \{c_i \mid 1 < i < n\}$, where $c_i$ denotes the i-th condition. The map $g_1(x) \subseteq C$ for $\forall x \in W$ or $\forall x \in M$ is used to obtain the conditions of each worker or machine, i.e., their capabilities. Similarly, the map $g_2(x) \subseteq C$ for $\forall x \in T$ describes the conditions for executing a task. In other words, a worker $x$ wants to complete a task $y$ iff $g_1(x) \supseteq g_1(y)$. In addition, preconditions of a task are denoted by the function $g_{pre}(x) \subseteq C$ for $\forall x \in T$, while $g_{post}(x) \subseteq C$ for $\forall x \in T$ denotes its post-conditions.

### 3.4 Incentive

A crowdsourcing system always considers some incentives that usually can be divided into two categories, i.e., intrinsic and extrinsic. In this study, all incentives are converted into money, i.e., workers earn money by accomplishing tasks, and requesters pay money for the completion of their tasks. The former amount is called the salary, while the latter amount is called the budget. The function $g_1(t)$ for $\forall t \in T$ is used to describe the budget of a task. The function $g_2(t)$ for $\forall t \in T$ is used to calculate the salary of a worker for the completion of a task.

### 3.5 Task Partitioning

Here, we assume that all composite tasks are partition-able. Each composite task can always be broke down into a series of subtasks including atomic, automated, and small-sized composite tasks. Therefore, given a composite task $t_i \in T_{ct}$, its partitioning is a set of subtasks $P = \{p_i \mid 1 < i < n\}$ satisfying the following conditions:

- Validness $\forall p_j \in P$, $\forall p_j \in t_i$.
- Completeness $\bigcup_{\forall p_j \in P} p_j = t_i$.
- Disjointness $\forall i, j, p_i \cap p_j = \phi$.

An atomic or automated task is the smallest task. Therefore, it should be further noted that there is no way to divide it.

### 3.6 Workflow

Petri nets have many advantages, including the graphical representation, a strong mathematical basis, various analytical techniques, and other extensions. Owing to these characteristics, it is possible to model complex situations in a structured and accessible manner. Further details, which are omitted here owing to space constraints, can be found in the literature [22].

The workflow can be denoted by the following tuple:

$\Sigma = \{P, p_i, p_o, T; F, M; \lambda_1, \lambda_2, \lambda_3, \lambda_4; T, g_{pre}, g_{post}; \lambda_5, \lambda_6, \lambda_7\}$ such that

- P is a set of places.
- $p_i$ is a source place.

- $p_o$ is a sink place.
- T is a set of transitions.
- For $\forall x \in (P \bigcup T)$, x is always contained in a path from $p_i$ to $p_o$.
- $F \subseteq (P \setminus \{p_i\} \times T) \bigcup (T \times P \setminus \{p_o\}) \bigcup (T \times T)$ is a flow relation.
- *M* is a map $P \to \mathbb{N}$ that represents the state of the workflow. It is the distribution of tokens over places.
- $\lambda_1 : T \to \{AND, XOR, OR\}$ specifies the split behavior of each transition.
- $\lambda_2 : T \to \{AND, XOR, OR\}$ specifies the join behavior of each transition.
- $\lambda_3 : T \nrightarrow \mathbb{P}(T \bigcup P \setminus \{p_i, p_o\})$ specifies the additional tokens to be removed by emptying a part of the workflow. $\mathbb{P}(T \bigcup P \setminus \{p_i, p_o\})$ is the set of all sets including places in $P \setminus \{p_i, p_o\}$ and transitions in T.
- $\lambda_4 : T \nrightarrow \mathbb{N} \times \mathbb{N}^{\inf} \times \mathbb{N}^{\inf} \{\text{dynamic}, \text{static}\}$ specifies the multiplicity of each transition, for instance, the minimum, maximum, threshold for continuation, and dynamic or static creation of instances.
- *T* is a set of tasks.
- $g_{pre}$ is used to obtain the preconditions of a task.
- $g_{post}$ is used to obtain the post-conditions of a task.
- $\lambda_5 : T \to T$ associates a task with a transition.
- $\lambda_6 : g_{pre}(t) \to P$ associates each precondition of a task with an element of the place set P.
- $\lambda_7 : g_{post}(t) \to P$ associates each post-condition of a task with an element of the place set P.

## 3.7 Task Design

In this study, the problem of task design is regarded as the iterative process of refinement, i.e., each composite task included in the parent workflow net requires to be replaced with a workflow subnet. The process is called *Crva* and formalized as follows:

Let $\Sigma = \{P, p_i, p_o, T; F, M; \lambda_1, \lambda_2, \lambda_3, \lambda_4; T, g_{pre}, g_{post}; \lambda_5, \lambda_6, \lambda_7\}$ be a workflow, then

$\Sigma' = \{P', p_i', p_o', T'; F', M'; \lambda_1', \lambda_2', \lambda_3', \lambda_4'; P, g_{pre}', g_{post}'; \lambda_5', \lambda_6', \lambda_7'\}$ is a workflow such that:

- *P* is the partition of a composite task $t \in T_{ct} \subseteq T$.
- $p_i' = g_{pre}(t)$ and $p_o' = g_{post}(t)$.
- $(P' \bigcup T') \bigcap (P \bigcup T) = \phi$.

Therefore, the replacement of the transition $t \in T$ by $\Sigma'$ in $\Sigma$ is

$\Sigma_{[t/\Sigma']} = \{P_{[t/\Sigma']}, p_{i[t/\Sigma']}, p_{o[t/\Sigma']}, T_{[t/\Sigma']}; F_{[t/\Sigma']}, M_{[t/\Sigma']}; \lambda_{1[t/\Sigma']}, \lambda_{2[t/\Sigma']}, \lambda_{3[t/\Sigma']}, \lambda_{4[t/\Sigma']}; T_{[t/\Sigma']}, g_{pre[t/\Sigma']}, g_{post[t/\Sigma']};$ $\lambda_{5[t/\Sigma']}, \lambda_{6[t/\Sigma']}, \lambda_{7[t/\Sigma']}\}$, where:

- $P_{[t/\Sigma']} = P \bigcup (P' \setminus \{p_i, p_o\})$.
- $P_{[t/\Sigma']} = p_i$.
- $P_{[t/\Sigma']} = p_o$.
- $P_{[t/\Sigma']} = (T \setminus \{\lambda_5(T)\}) \bigcup T'$.
- $F_{[t/\Sigma']} = (F \setminus \{(\bigcup\{(p,t) \mid \forall p \in P\}) \bigcup (\bigcup\{(t,p) \mid \forall p \in P\}) \bigcup F'$.
- $M_{[t/\Sigma']} = M(P) \bigcup M'(P')$.
- $\lambda_{1[t/\Sigma']}(t') = \lambda_1(t')$ if $t' \in (T \setminus \{\lambda_5(t))$ and $\lambda_1'(t')$ otherwise.
- $\lambda_{2[t/\Sigma']}(t') = \lambda_2(t')$ if $t' \in (T \setminus \{\lambda_5(t))$ and $\lambda_2'(t')$ otherwise.
- $\lambda_{3[t/\Sigma']}(t') = \lambda_2(t')$ if $t' \in (T \setminus \{\lambda_5(t))$ and $\lambda_3'(t')$ otherwise.
- $\lambda_{4[t/\Sigma']}(t') = \lambda_2(t')$ if $t' \in (T \setminus \{\lambda_5(t))$ and $\lambda_4'(t')$ otherwise.
- $T_{[t/\Sigma']} = (T \setminus \{t\}) \bigcup P$.
- $f_{pre[t/\Sigma']}(t') = g_{pre}(t')$ if $t' \in (T \setminus \{t\})$ and $g_{pre}'(t')$ otherwise.

- $f_{post[t/\Sigma']}(t') = g_{post}(t')$ if $t' \in (T \setminus \{t\})$ and $g'_{post}(t')$ otherwise.
- $\lambda_{5[t/\Sigma']}(t') = \lambda_5(t')$ if $t' \in (T \setminus \{t\}$ and $\lambda'_5(t')$ otherwise.
- $\lambda_{6[t/\Sigma']}(t') = \lambda_6(t')$ if $t' \in (T \setminus \{t\}$ and $\lambda'_6(t')$ otherwise.
- $\lambda_{7[t/\Sigma']}(t') = \lambda_7(t')$ if $t' \in (T \setminus \{t\}$ and $\lambda'_7(t')$ otherwise.

Fig. 1 shows the iterative process of refinement. The composite task $t_2$, whose transition is $t_2 = \lambda_5(t_2)$, is replaced with $\Sigma'$.

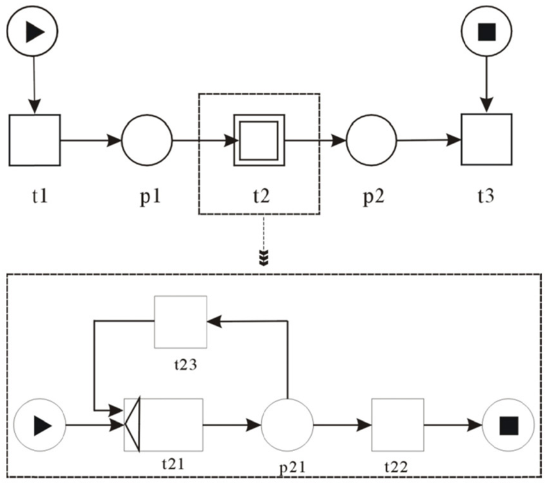


**Fig. 1.** Illustrating the process of refinement. The transition $t_2$ corresponding to the composite task $t_2$ is replaced by the lower workflow

### 3.8 Soundness

After the task design is accomplished, the workflow specification includes a set of workflows that resembles an inverted tree. The workflow specification may face a deadlock, be unable to terminate, or have dead parts. Therefore, we need to analyze its soundness properties to ensure that it is correct before running it.

Let $\Sigma$ be sound iff

- for $\forall M$, $(M_{\text{start}} \rightarrow M) \Rightarrow (M \rightarrow M_{\text{end}})$, i.e., there is always a path from $M_{\text{start}}$ to $M_{\text{end}}$.
- for $\forall M$, $(M_{\text{start}} \xrightarrow{*} M) \wedge (M \geq M_{\text{end}}) \Rightarrow (M = M_{\text{end}})$, i.e., the end state $M_{\text{end}}$ is the only reachable state from $M_{\text{start}}$ and it has at least one token in place.
- for $\forall t \in T$, $\exists M, M'$ such that $M_{\text{start}} \xrightarrow{*} M \xrightarrow{t} M'$, i.e., each transition may be fired.

## 4 Crowdsourcing System

Here, accomplishing a crowdsourcing task involves four stages: design, allocation, inference, and fulfillment. A crowdsourcing system is formalized as a tuple $\Gamma = (R, T, M, t; h_1, h_2, h_3, h_4)$, where *R*, *W*, and *M* are requesters, workers, and machines respectively. Further, $h_1, h_2, h_3$, and $h_4$ are responsible for task design, task allocation, inferring the correct answer, and fulfilling the promise respectively.

In this context, the algorithm $h_1$ is mainly used to complete the conversion from the task set *T* to the workflow specification $\Sigma$. By the algorithm $h_2$, all tasks of the workflow $\Sigma$ are allocated to workers who are chosen under constraints, namely the budget of a requester, the conditions of a task, the conditions of a worker, and the salary of a worker. In other words, the total cost of completing all tasks should be less than the total budget, i.e., $\Sigma_T g_1(t) \geq \Sigma_T g_2(t)$, and the condition, i.e., $g_1(t) \subseteq g_2(y)$ for $\forall t \in T$ and $(\forall y \in W$ or $\forall y \in M)$, should also hold for each worker that is chosen to complete the task *t*.

After the execution of tasks, the algorithm $h_3$ collects diverse results and infers the trusted answer from them. The algorithm $h_4$ pays the salaries to the workers according to their contributions.

In this study, we completely focus on the problem to exploit the collective intelligence in order to complete the task design and obtain a verifiable result.

To achieve the above-mentioned goals, we design the Create-Refine-Vote-Analyze pattern, CRVA-I, inspired by the literature [12]. CRVA-I separates the task design into four phases, as shown in Fig. 2. The first stage, Create, asks a requester to submit an initial workflow called the root workflow, which is highly abstract and only includes a source place, a sink place, and a composite task that gives general statements for the problem.

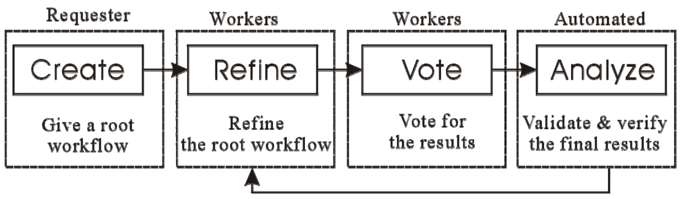


**Fig. 2.** Create-Refine-Vote-Analyze pattern. A requester gives an initial workflow and the refinement process is then completed by workers. The number of loops can be set by the requester

The second stage, Refine, recruits workers to refine the root workflow. Specifically, the requester, first, needs to choose workers according to their skill and salary for the execution of the task. Second, each worker refines the initial root workflow. We present the refinement process using the pseudo-code in Algorithm 1. Finally, the workers submit the design results.

```
Algorithm 1. RPW()
Input: a given workflow specification $\Sigma$
Output: the refinement of $\Sigma$
1. for i = 1 to $|T_{ct}|$ do
2.     $t = t_i \in T_{ct}$
3.     $p_i' = g_{pre}(t)$ and $p_o' = g_{post}(t)$
4.     Divide $t$ into $P = \{p_j \mid 1 < j < n\}$
5.     Let $P = \{p_i', p_o'\}$ and $T' = \{\}$
6.     for j=1 to $|P|$ do
7.         $t' = p_j$
8.         $T' = T' \cup \{\lambda_5(t')\}$
9.         $P' = P' \cup \lambda_6'(g_{pre}(t')) \cup \lambda_7'(g_{post}(t'))$
10.    end for
11.    $F' \subseteq (P' \setminus \{p_i'\} \times T) \cup (T' \times P' \setminus \{p_o'\}) \cup (T' \times T')$
12.    According to $P', T', F'$, set $M', \lambda_1', \lambda_2', \lambda_3', \lambda_4', g_{post}', \lambda_5', \lambda_6', \lambda_7'\}$
13.    Let $\Sigma' = \{P', p_i', p_o', T'; F', M'; \lambda_1', \lambda_2', \lambda_3', \lambda_4'; P, g_{pre}', g_{post}'; \lambda_5', \lambda_6', \lambda_7'\}$
14.    Do $\Sigma = \Sigma_{[t/\Sigma']}$
15. end for
16. return $\Sigma$
```

The last stage, Analyze, requires the requester to complete the analysis of candidates and make final decisions. More specifically, the requester needs to analyze the soundness of each option. Of course, the task can be performed automatically because the workflow specified in terms of a Petri net has clear and precise definitions. There are three workflows: $\Sigma_1$, $\Sigma_2$, $\Sigma_3$. If $\Sigma_3 = \Sigma_1 \uplus \Sigma_2$, then $\Sigma_1$ and $\Sigma_2$ are sound if $\Sigma_3$ is sound. The proof of the statement can be found in the literature [23]. Therefore, the soundness of candidates can be analyzed hierarchically because the final workflow specifications have a tree-like hierarchy. Algorithm 2 gives the pseudo-code for the analysis process.

**Algorithm 2.** SAH()

```
Input: a tree-like workflow specification $\Sigma_{tree}$
Output: 0 if sound, 1 otherwise
1.  Construct a queue q
2.  q.push( $\Sigma_{tree}$ )
3.  n=q.pop()
4.  while n do
5.      Construct the reachable graph
6.      if n is not sound then
7.          return 0
8.      end if
9.      if n->left then
10.         q.push(n->left)
11.     end if
12.     if n->right then
13.         q.push(n->right)
14.     end if
15.     n= q.pop()
16. end while
17. return 1
```

## 5 Case Study

In this section, a crowdsourcing writing task is considered to illustrate the feasibility of the proposed method. The case study involves the writing of a review article about crowdsourcing. Crowdsourcing writing is chosen as a test domain because it is a complex task.

First, a requester posts a task of writing a crowdsourcing review on the crowdsourcing system in the Create stage. Furthermore, he gives a root workflow shown in Fig. 3. It is very easy to design the root workflow for a requester because it is only an abstract task.

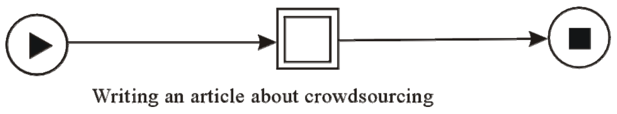


**Fig. 3.** A requester gives the above workflow. In the workflow, the requester only gives a composite task to disclose his/her intention

In the Refine stage, the requester recruits some workers according to his/her requirements and allocates refinement tasks to them. Here, we assume that the requester hires two workers. After accomplishing the task, the workers post their respective answers, i.e., the refinement of the root workflow, as shown in Fig. 4 and Fig. 5.

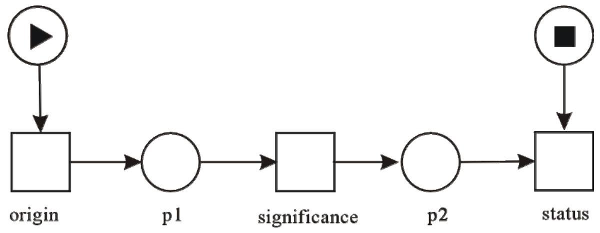


**Fig. 4.** The answer is given by one worker. The worker splits the above composite task into three sub-tasks: origin, significance, and status, i.e., the origin, significance and status of crowdsourcing. Furthermore, he/she believes that these tasks should be completed in order

In the next stage, i.e., the Vote stage, the requester recruits some workers according to his/her requirements and allocates vote tasks to them. After voting, the requester chooses the top result. Here, we assume that Fig. 5 is chosen.

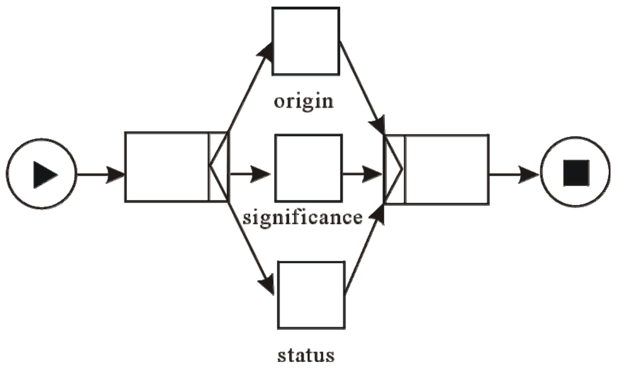


**Fig. 5.** The answer is given by another worker. The worker also splits the above composite task into three sub-tasks: origin, significance and status, i.e. the origin, significance, and status of crowdsourcing. However, he/she believes that these tasks should be completed simultaneously

In the last stage, the requester needs to analyze the final workflow specifications using **Algorithm 2**. The reachable graph is shown in Fig. 6.

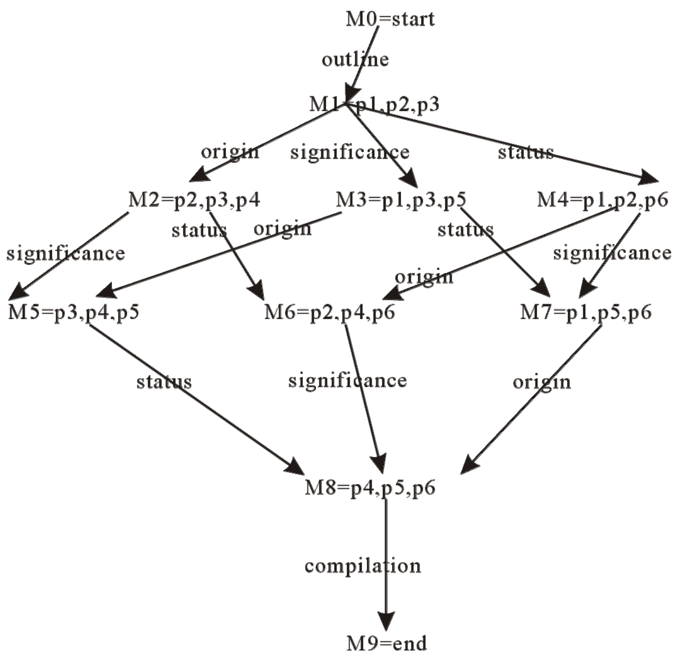


**Fig. 6.** Reachable graph of Fig. 5. The requester chooses the second answer and analyzes it. For simplicity, the above workflow does not appear in the reachable graph

In summary, tedious task designs are delegated to skilled workers. For a requester, what he/she needs to do is to post an initial and easy root workflow. As a result, he/she obtain a formal workflow model described by Petri nets. Therefore, many automated tools can be used to analyze the model.

## 6 Performance Evaluation

### 6.1 Experiment Setup

To evaluate the feasibility of the proposed method, a prototype was developed by assembling open-source core components including Spring Boot, Activity, and YAWL editor. Spring Boot was considered as the main framework of the current prototype. Activity, a workflow engine, was used to implement the

Create-Refine-Vote-Analyze pattern. The YAWL editor, a Petri net workflow design tool, was provided to workers to accomplish refinement of the workflow. In addition, we revised the editor to enable it to support collaboration based on the GitHub repository. The source code for the revised editor can be found on the Web[2].

## 6.2 Experimental Results

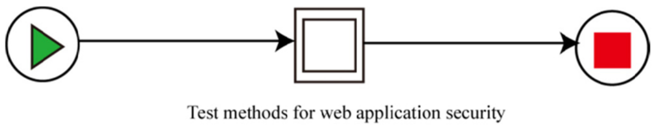
Our initial evaluation sought to establish evidence for feasibility of the proposed method, as well as to understand the characteristics of the Create-Refine-Vote-Analyze task design pattern.

Our goal was to determine the extent to which the proposed method enables unskilled requesters to obtain a high-quality workflow model. To simulate the entire procedure, we forged a complex task related to Web application security testing and started a workflow instance by imitating a real requester.

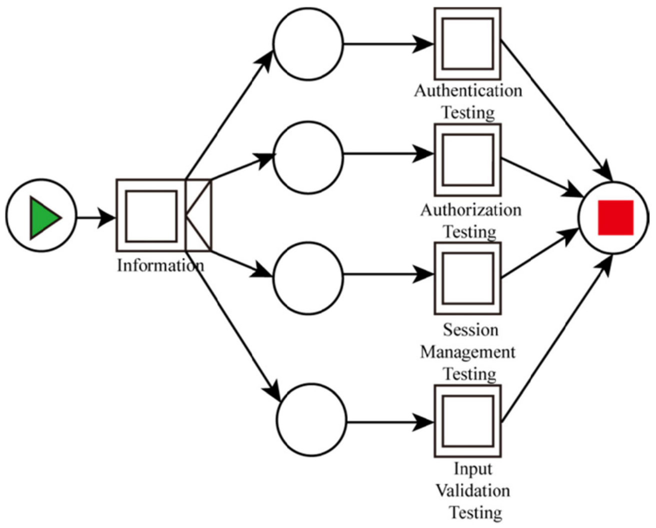
Five workers were required to complete the Refine tasks. To simulate the scenario, we singled out five excellent students who have some knowledge of Web penetration testing to imitate five workers. We required three workers to vote for the candidates. In this stage, we singled out three outstanding students to imitate three common workers. Finally, the imitated requester analyzed the soundness of the top candidate. After accomplishing the job, the requester could either stop the case or go back to the Refine stage and repeat the above-mentioned procedures until he/she was satisfied with the result. Here, the procedure was repeated three times.

First, the impersonation of a requester gave the init workflow. There was only a composite task that described the requirements of the requester to get test methods for Web application security, as shown in Fig. 7(a).

Second, five workers accomplished refinement tasks and submitted their answers. After voting, the requester made the decision, as shown in Fig. 7(b). After the first refinement, the abstract task was split into four subtasks: Authentication, Authorization, Session Management and Input Validation. These subtasks could be executed simultaneously.



(a) Init workflow

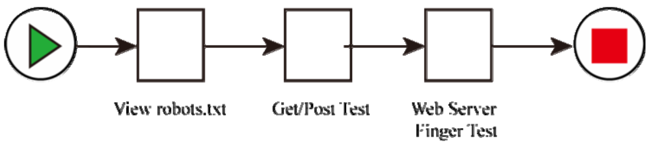


(b) Final selection

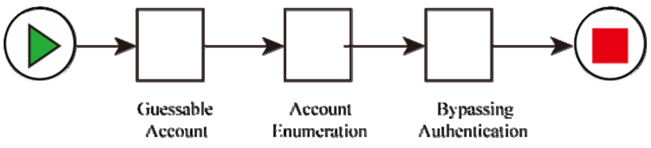
**Fig. 7.** Initial

[2] https://github.com/hhluci/yawl42

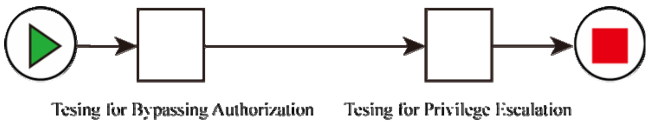
Next, the requester did not stop the iteration. Instead, he started the second refinement. Finally, four subtasks, namely Authentication, Authorization, Session Management, and Input Validation were refined respectively, as shown in Fig. 8. These subtasks need to be completed in order.



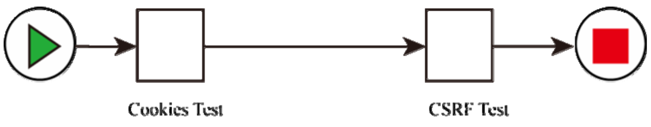
(a) Refinement of the task: Information Gathering



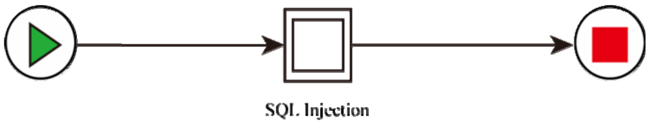
(b) Refinement of the task: Authentication Test


Tesing for Bypassing Authorization
Tesing for Privilege Escalation

(c) Refinement of the task: Authorization Test



(d) Refinement of task: Session Management Test



(e) Refinement of task: Input Validation Test

**Fig. 8.** Second refinement

Finally, the requester asked the workers to refine the subtask SQL injection. The last selection for this refinement can be seen in Fig. 9. For SQL injection, the tester first needs to find the injection entrance. Then, he/she can retrieve all users by SQL injections. The two subtasks need to be executed in order. Next, he/she can execute three subtasks, namely Get Databases, Get Tables, and Get Columns, simultaneously. Finally, the test results are submitted.

Through this case, the proposed method enables requesters to delegate difficult task design to workers and supports collaboration between requesters and workers. Moreover, requesters can control the entire process.

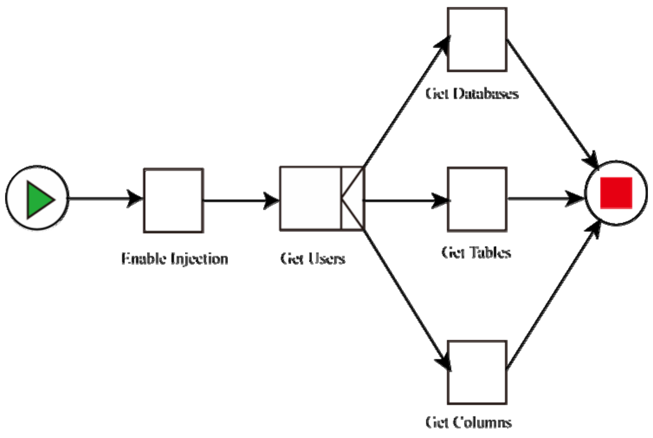


**Fig. 9.** Third refinement: the refinement for SQL injection

### 6.3 Comparative Analysis

Modeling of task design in the crowdsourcing system is the main concern of the current study. Table 1 summarizes the differences between the current study and other related studies. The "Relies on" column indicates whether the method relays the crowdsourcing process to the third-party platforms. The "Complex Tasks" column describes, if available, how to handle complex tasks. The fourth column, "Decomposition Strategy", describes the method adopted in the process of task design. The "Formal Model" and "Formal Analysis" columns indicate whether the final results are formal and whether formal analysis is performed, respectively. The last column, "Task Designer", indicates who designs the task.

**Table 1.** Comparison with other methods

| Method | Relies on | Complex Tasks | Decomposition Strategy | Formal Model | Formal Analysis | Task Designer |
|---|---|---|---|---|---|---|
| CrowdForge [7] | MTurk | Yes (Workflows) | Map Reduce | No | No | Requester |
| Turkit [8] | MTurk | Yes (Workflows) | Crash and Rerun | No | No | Requester |
| Turkomatic [14] | MTurk | Yes (Workflows) | Divide and Conquer | No | No | Worker |
| CrowdComputer [15] | Self | Yes (Workflows) | Customer API | No | No | Requester |
| BOWorkflow [16] | Self | Yes (Workflows) | Decompose Merge | No | No | Worker |
| SmartCrowd [17] | Self | Yes (Workflows) | Decompose Merge | No | No | Worker |
| Service4Crowd [18] | Self | Yes (Workflows) | Decompose Merge | No | No | Requester |
| Jabberworcky [24] | Self | Yes (Workflows) | Map Reduce | No | No | Requester |
| Current Study | Self | Yes (Workflows) | Create-Refine-Vote-Analyze | Yes | Yes | Worker |

It is obvious that the trend is toward decomposing complex tasks into simple subtasks in dealing with crowdsourcing complex tasks. As shown in Table 1, all of methods adopted decomposition strategies based on workflows to accomplish complex tasks. In the current paper, the workflow was also adopted to model the crowdsourcing process. Authors customized a simple workflow based on the MapReduce programming idea in [7]. The method is limited to some particular scenarios. In contrast, our method is more general. Authors explored an iterative paradigm which had a lack of flexible controls in [8]. The literature [14] is similar to our work in that sense. Authors of two literatures think that the task design is so difficulty that requesters have to invest substantial effort to complete it. In addition, we think that it is necessary to delegate the responsibility for designing workflows to skilled workers. The main difference is that our method places more stress on collaboration and verifiability. Compared to the literatures [1, 7-8], [15-18, 24] try to design a new mechanism instead of improving existing crowd platforms. We also appreciate this idea and put it into practice. The literature [15] modeled the crowdsourcing process using the BPMN technology. The method provides more flexibility than other methods but more complicated. In contrast, [16] seems simpler than [15] in practice, which uses the state machine workflow technology to model the crowdsourcing process. The literature [17] improved the automation of the total process compared with [16]. Authors provided a web-based distributed application that supports the crowdsourcing process management in [18]. The downside of this approach is that the requester is still sole responsibility for the design work. In other words, it does not support that requesters and workers do the design work together, which is different from the current study. The advantage of [24] is that human and machine resources are managed together. However, it mainly provides the tools available to programmers for crowd computing.

In summary, the differences are mainly reflected in several aspects. On the one hand, the current study enables workers to complete difficult design jobs. However, other studies require requesters to do so, which is difficult for unskilled requesters in practice. On the other hand, the current study enables a requester to obtain a formal workflow model, but other studies have no such function. The formal results are extremely important for make automatic analysis. Furthermore, this aspect is of particular importance for quality control in crowdsourcing environments.

## 7 Conclusion

Although popular crowd platforms always ask requesters to accomplish the crowd design, we argue that the work is difficult for requesters and sometimes it does not work in practice. To solve the issue, we explored a novel idea whether requesters and workers can collaborate on the work. We found an alternative method that enables workers to share the stress of requesters via Petri nets. The new method enables requesters to achieve high-quality design of complex tasks. We formalized the process of task design on the basis of Petri nets and described the Create-Refine-Vote-Analyze task design pattern in detail. Furthermore, we introduced a case on crowdsourcing writing to further elucidate our ideas. In addition, we implemented a system prototype to evaluate the advantages and disadvantages of the proposed method.

In addition, we argue that this study would contribute to existing literature in several ways. First, it is the first attempt to commission the design of complex tasks to workers to improve the quality of task design. Second, we employ Petri nets to crowd computing, which enriches the application scenarios of the old technology. Third, we give an alternative formal model of the crowd task design. In addition, the new method enables requesters to detect the quality of the design.

In the future, we plan to simplify the Petri nets editor to lower the demands for workers by exploring man-machine task design methods. We also plan to perform additional crowdsourcing tasks to identify other weaknesses of the proposed method and improve upon them.

## Acknowledgments

We would like to thank everyone who contributed suggestions and ideas to this work, including Liu Zhongbao, Cai Xingwang, and other members of the research group. This work was supported by the Natural Science Foundation of Shanxi Province of China under Grant 201801D121151. Any opinions, Findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

## References

[1] T.P. Robinson, M.E. Kelley, Renewal and resurgence phenomena generalize to Amazon's Mechanical Turk, Journal of the experimental analysis of behavior 11(1)(2020) 206-213.

[2] A. Ghezzi, D. Gabelloni, A. Martini, A. Natalicchio, Crowdsourcing: A Review and Suggestions for Future Research, International Journal of Management Reviews 20(2)(2018) 343-363.

[3] J.C.R. Licklider, Man-Computer Symbiosis, IEEE Transactions on Human Factors in Electronics 1(1)(1960) 4-11.

[4] J. Hendler, T. Berners-Lee, From the Semantic Web to social machines: A research challenge for AI on the World Wide Web, Artificial Intelligence 174(2)(2010) 156-161.

[5] J. Howe, M. Tech, P. Reviews, The Rise of Crowd sourcing, Wired Magazine 14(2006) 1-6.

[6] N. Luz, N. Silva, P. Novais, A survey of task-oriented crowdsourcing, Artificial Intelligence Review 44(2)(2015) 187-213.

[7] A. Kittur, B. Smus, R. Kraut, CrowdForge: crowdsourcing complex work, in: Proceedings of Annual Acm Symposium on User Interface Software & Technology, 2011.

[8] G. Little, L.B. Chilton, M. Goldman, TurKit: Tools for Iterative Tasks on Mechanical Turk, in: Proceedings of the ACM SIGKDD Workshop on Human Computation, 2009.

[9] G. Little, L.B. Chilton, M. Goldman, Turkit: human computation algorithms on mechanical turk, in: Proceedings of 23rd Annual ACM Symposium on User Interface Software and Technology, 2010.

[10] P. Dai, Decision—theoretic control of crowdsourced workflows, in: Proceedings of AAAI, 2010.

[11] C.H. Lin, Dynamically Switching between Synergistic Workflows for Crowdsourcing, in: Proceedings of 26th AAAI, 2012.

[12] M.S. Bernstein, G. Little, R.C. Miller, Soylent: a word processor with a crowd inside, in: Proceedings of ACM Symposium on User Interface Software & Technology, 2010.

[13] A. Kulkarni, M. Can, B. Hartmann, Collaboratively crowdsourcing workflows with turkomatic, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work, 2012.

[14] A. Kulkarni, B. Hartmann, Turkomatic: automatic recursive task and workflow design for mechanical turk, in: Proceedings of Human Factors in Computing Systems, 2011.

[15] S. Tranquillini, F. Daniel, Modeling Enacting and Integrating Custom Crowdsourcing Processes, ACM Transactions on the Web 9(2)(2015) 1-43.

[16] Z. Qiang, W. Wei, Y. Yang, Crowdsourcing Complex Task Automatically by Workflow Technology, in: Proceedings of Management of Information, Process and Cooperation, 2017.

[17] T. Xiong, Y. Yu, M. Pan, SmartCrowd: A Workflow Framework for Complex Crowdsourcing Tasks, in: Business Process Management Workshops, 2019.

[18] W. Shujie, S. Hailong, C. Pengpeng, Service4Crowd: A Service Oriented Process Management Platform for Crowdsourcing, in: Proceedings of Computer Supported Cooperative Work and Social Computing, 2018.

[19] T. Ranra, L. Faming, Z. Xueping, An introduction and review of petri net unfolding technology, in: Proceedings of the 4th International Conference on Communication and Information Processing, 2018.

[20] Microsoft, State Machine Workflow in Windows Workflow Foundation. <https://msdn.microsoft.com/enus/lib rary/ee264171(v=vs.110).aspx>, 2017 (accessed 21.01.30).

[21] Activiti, activiti-7-developers-guide. <https://www.activiti.org/>, 2021 (accessed 21.01.30).

[22] K. Salimifard, M. Wright, Petri net-based modeling of workflow systems: An overview, European Journal of Operational Research 134(3)(2001) 664-676.

[23] W.M. Aalst, Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques, Business Process Management Models Techniques & Empirical Studies, 2000.

[24] S. Ahmad, A. Battle, Z. Malkani, The jabberwocky programming environment for structured social computing, in: Proceedings of 24th Annu. ACM Symp. User Interface Software Technology, 2011.