A Context-Driven Worker Selection Framework for Crowd-Sensing

Wang, J., Wang, Y., Helal, S. & Zhang, D.

Original citation:
Wang, J, Wang, Y, Helal, S & Zhang, D 2016, 'A Context-Driven Worker Selection Framework for Crowd-Sensing', International Journal of Distributed Sensor Networks, vol. 12, no. 3, 6958710. https://dx.doi.org/10.1155/2016/6958710

DOI 10.1155/2016/6958710
ISSN 1550-1329
ESSN 1550-1477

Publisher: SAGE Publications

Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2016, Article ID 6958710, 16 pages
http://dx.doi.org/10.1155/2016/6958710

Research Article

A Context-Driven Worker Selection Framework for Crowd-Sensing

Jiangtao Wang,[1,2] Yasha Wang,[1,3] Sumi Helal,[4] and Daqing Zhang[1,2]

[1]*Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China*
[2]*School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China*
[3]*National Engineering Research Center of Software Engineering, Peking University, Beijing 100871, China*
[4]*Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 116120, USA*

Correspondence should be addressed to Yasha Wang; wangys@sei.pku.edu.cn

Received 1 September 2015; Accepted 27 December 2015

Academic Editor: Diego López-de-Ipiña

Worker selection for many crowd-sensing tasks must consider various complex contexts to ensure high quality of data. Existing platforms and frameworks take only specific contexts into account to demonstrate motivating scenarios but do not provide general context models or frameworks in support of crowd-sensing at large. This paper proposes a novel worker selection framework, named WSelector, to more precisely select appropriate workers by taking various contexts into account. To achieve this goal, it first provides programming time support to help task creator define constraints. Then its runtime system adopts a two-phase process to select workers who are not only qualified but also more likely to undertake a crowd-sensing task. In the first phase, it selects workers who satisfy predefined constraints. In the second phase, by leveraging the worker's past participation history, it further selects those who are more likely to undertake a crowd-sensing task based on a case-based reasoning algorithm. We demonstrate the expressiveness of the framework by implementing multiple crowd-sensing tasks and evaluate the effectiveness of the case-based reasoning algorithm for willingness-based selection by using a questionnaire-generated dataset. Results show that our case-based reasoning algorithm outperforms the currently practiced baseline method.

1. Introduction

Recent years have seen rapid improvements in the capabilities of mobile phones, such as processing power, embedded sensors, storage capacities, and network data rates. These technology advances coupled with the sheer number of user-companioned mobile phones enable a new and fast-growing sensing paradigm, which is referred to as crowd-sensing. Crowd-sensing is a capability by which application developers can create tasks and recruit smartphone users to provide sensor data to be used towards a specific goal. In this paper, developers who create the crowd-sensing task are referred to as task creators, while mobile users who fulfill the task by contributing data are referred to as workers.

To support crowd-sensing at large, many mediation platforms and frameworks have been proposed recently to connect the task and worker, which can be divided into two modes. The first is pull mode. For many existing mediation platforms, it is the worker's responsibility to search for the task he/she wants to undertake. In these platforms, the task creators post tasks on the platform by describing what should be done, defining the layout of the user interface, and specifying the reward. Then workers actively login to the platform and search for tasks that they are qualified for and interested in by providing some keywords. Then a worker will get a list of tasks and decides which one to complete. Typical systems adopting the pull mode include Amazon Mechanical Turk [1], Medusa [2], CrowdDB [3], CrowdSearch [4], and TurKit [5]. The second is push mode, in which the mediation platform or framework must be able to select appropriate workers automatically based on certain constraints. Typical systems or frameworks using the push mode include [6], PRISM [7], and Anonysense [8]. With the popularity of crowd-sensing paradigm, the number of both tasks and workers has been increasing rapidly. As a result, both the pull mode and push mode connection

between task and worker are important, since they either help workers find their preferred tasks or deliver a certain task to the most appropriate workers. The pull/push modes are analogous to the search-based and recommendation-based approaches, respectively, utilized for dealing with the information overload problem on the Web.

There are several research works focusing on the worker selecting for crowd-sensing. However, they have the following limitations.

First, many platforms do not provide general support for managing various and complex contexts. This is a significant deficit given that one of the biggest challenges for worker selection is the ability to utilize complex and generalized contexts in many cases. In this paper, factors that need to be considered in identifying appropriate workers are supported by a variety of contexts to be utilized by the crowd-sensing worker selection process. Existing mediation platforms or frameworks also take contexts into account to some extent. However, they only consider specific or sample contexts to support their own motivating scenarios but do not provide general context models or frameworks in support of crowd-sensing at large. For example, [6] points out that people's availability in terms of space and time, transportation mode, and the coverage must meet certain requirements in the selection phase. PRISM [7] takes the device's sensing capabilities and worker's location as the contexts. Anonysense [8] takes time, worker's location, and privacy setup into account. The authors in [9] regard the location, battery level, and coverage as contexts. In [10, 11], the budget and coverage are the contexts for selection.

Second, the selection is not precise enough, because existing platforms exploit contexts only in terms of the requirements defined by the task creator without considering other worker-required contexts that could highly decide whether the worker would accept the task or not. The task creator is either a skilled software developer or people with very basic programming language skills. According to their programming capabilities, different levels of language support are provided to help them define the contexts for worker selection. For example, [2] provides XML-based language for nonprofessional programmers, while PRISM [7] provides higher level programming support for professional programmers. No matter what level of programming support is provided, existing systems select the workers based on the constraints predefined merely by the task creator. However, this is not precise enough if too many workers meet the constraints. With the increasing popularity of crowd-sensing, such imprecise task pushing may overwhelm the workers given the large number of recommended tasks. In fact, workers decide whether to accept and actually undertake a recommended task based on many other contexts, for example, whether the reward is attractive enough, whether the worker is interested in the task domain, and if accomplishing the task may not cause too much privacy violation. These contexts are not well exploited in existing systems.

To overcome abovementioned limitations, this paper proposes a novel worker selection framework, named WSelector, to select appropriate workers by taking various contexts into account. To achieve this goal, WSelector first provides programming time support, which is based on context modeling technique, to help task creators define all constraints for worker selection. Then a two-phase selection process is adopted at runtime to identify workers who not only are qualified but also would be more willing to undertake a crowd-sensing task. In the first phase, it selects workers satisfying predefined constraints. In the second phase, by leveraging the worker's past participation history, it uses a case-based algorithm to further select workers who are more likely to accept to undertake the task. WSelector is not a mediation platform and does not handle many of the complex issues found in many of the existing platforms. However, it is intended to be integrated into existing or future mediation platforms to better support their worker selection features.

The main contributions of this paper are as follows:

(1) We propose a core context model to semantically express the general context for worker selection in crowd-sensing systems and to provide a mechanism for task creators to utilize context modeling in their applications.

(2) We propose a two-phase worker selection framework to identify workers who not only are qualified but are more likely to be willing to undertake a crowd-sensing task by taking various worker-side contexts into account.

(3) We demonstrate the expressiveness of the framework based on various crowd-sensing tasks and evaluate the effectiveness of a case-based reasoning algorithm based on a dataset collected from an online questionnaire.

2. Related Work

In this section, we review the related literature from two perspectives. The first is the worker selection capabilities of related platforms/frameworks for crowd-sensing, which is the main problem we are addressing in this paper. The second is the ontology-based context modeling, which is related to a key technique we use as a solution to the problem in our framework.

2.1. Worker Selection Framework. Many crowd-sensing mediation platforms, including Amazon Mechanical Turk [1], Medusa [2], CrowdDB [3], CrowdSearch [4], TurKit [5], and mCrowd [12], adopt the pull mode to connect task and worker. In these platforms, workers search the task and the platform does not need to select workers. Compared with these frameworks, ours can identify appropriate workers based on various contexts information and push tasks to them.

There are many other frameworks that are based on the push mode. In this mode, the mediation platform must be able to identify appropriate workers automatically based on certain factors. References [11, 13] develop a selection framework to enable organizers to identify well-suited participants for data collections based on geographic and temporal availability, transportation mode, and the coverage. PRISM [7]

Table 1: Worker selection of typical mediation platforms or framework: a comparison.

Platforms/frameworks	Comparison aspect		
	Push/pull	Consideration for contexts	Whether to consider worker-side contexts
Amazon Mechanical Turk [1]	Pull	No context	No
Medusa [2]	Pull	No context	No
CrowdDB [3]	Pull	No context	No
CrowdSearch [4]	Pull	No context	No
TurKit [5]	Pull	No context	No
mCrowd [12]	Pull	No context	No
[6, 11]	Push	Geographic and temporal availability, transportation mode, and the coverage	No
PRISM [7]	Push	Device's sensing capabilities and worker's location	No
Anonysense [8]	Push	Time, worker's location, and privacy setup	No
[9]	Push	Device's location, battery level, and the overall spatial coverage	No
CrowdRecruiter [10]	Push	Incentive payments and coverage constraint	No
Crowdlab [14]	Push	Location and battery budget	Yes
[15]	Push	Location	No
[16]	Push	Energy consumption and data quality	No
[17]	Push	data quality requirements, location, and budget constraints	No
WSelector	Push	Providing general context model	Yes

takes the device's sensing capabilities and worker's location as the constraints to select appropriate workers. Anonysense [8] takes time, worker's location, and privacy setup into account when identifying workers. Reference [9] proposes an assignment policy to identify suitable workers based on their device's location, battery level, and the overall spatial coverage. CrowdRecruiter [10] aims at minimizing incentive payments by selecting a small number of participants while still satisfying probabilistic coverage constraint. Crowdlab [14] schedules the task based on location and battery resource budget. Reference [15] takes location as the only context for worker selection. A QoI-Aware energy-efficient worker selection framework has recently been proposed [16], which considers the QoI requirements, the energy consumption index, and the estimation of the collected amount of data. The literature [17] selects workers based on the data quality requirements, location, and budget constraints. Although these works also take many contexts into account for worker selection, they only consider demonstrative contexts to support their motivating scenarios but do not provide general context models in support of crowd-sensing at large. Our paper proposes a core context model for worker selection, in which the general concepts for crowd-sensing are defined and characterized. All the demonstrative contexts in above frameworks can be modeled by extending our core context model. Besides, worker selection in the most of existing frameworks is merely based on the constraint defined by the task creator, while WSelector takes the worker-side contexts into account.

A comparison of the platforms or frameworks in terms of worker selection is summarized in Table 1.

2.2. Ontology-Based Context Modeling. Ontology-based context modeling techniques are widely used in pervasive computing systems, especially for modeling heterogeneous contexts in smart pervasive spaces. One of the first approaches of modeling the context with ontologies has been proposed by [18], which analyzed psychological studies on the difference between recall and recognition of several issues in combination with contextual information. Another approach has been proposed as the Aspect-Scale-Context Information (ASC) model [19]. Ontologies provide a uniform way for specifying the model's core concepts as well as an arbitrary amount of subconcepts and facts, altogether enabling contextual knowledge sharing and reuse [20]. The model has been implemented by applying selected ontology languages. These implementations build up the core of a nonmonolithic Context Ontology Language (CoOL), which is supplemented by integration elements such as scheme extensions for Web Services and others [21, 22]. The CONON context modeling approach [23, 24] created an upper ontology which captures general features of basic contextual entities and a collection of domain-specific ontologies and their features in each subdomain. Above works inspire our work to some extent, especially the idea of building domain-specific ontology based on an upper-level ontology proposed by [23, 24]. However, to the best of our knowledge, our paper is a

first work using ontology-based context modeling in crowd-sensing.

There are also other literatures inspiring our work, in that they also use the crowd-sourced way for ontology-based knowledge modeling. Reference [25] presented a hybrid metamodel that combines features from key-value, markup, object oriented, and ontology-based context modeling approaches. The architecture is also introduced to allow the dynamic collaborative extension and crowd-sourced convergence of context models. In [26], the authors presented different concepts to create context models, including a collaborative one. In [27], the authors proposed consensus-building mechanisms for collaborative ontology building, which is based on offline discussions. Reference [28] presented an entirely online-based process for ontology building and convergence that is implemented on an extended Wiki.

3. Framework Design Overview

3.1. Challenges. The goal of our framework is to perform a precise worker selection for crowd-sensing. It should be able to identify workers who not only are qualified but also willing to undertake a specific task. In order to achieve the goals, at least the following challenges exist.

First, it is difficult to model various contexts for worker selection. A key challenge for worker selection is that in many cases complex and various contexts must be considered, and they are all factors that need to be considered for selecting appropriate workers. To the best of our knowledge, though context modeling is a widely used technique in mobile and pervasive systems, there are no existing context models for characterizing the concepts and their relationships in crowd-sensing applications.

Second, it is difficult for the task creator to define all contexts reasonably or properly. A naive idea for enabling the worker selection is to require the task creator to define all the contexts which can be realized by task specification languages such as two-level predicate in [7], AnonyTL [8], and Medscript [2]. Although these languages are useful in defining the contexts to some extent, there are limitations in many scenarios. It is not at all clear if certain combinations of contexts may be unnecessarily too restrictive, or to the contrary, adequate to achieve successful selection. Task creator may not be thoughtful enough to define all needed contexts that some workers are likely to be missed and the task is likely to be pushed to inappropriate workers. Sometimes they may define a context that is unnecessarily too restrictive, thus excluding many appropriate workers.

Third, selecting workers who are willing to undertake a task is not easy. On the one hand, different contexts could have different influence in deciding whether the worker is willing to undertake a task. It is very difficult for task creator to predict or learn these differences. On the other hand, even the same context may have different impact on different workers, which is subjective and varies from one worker to another. For example, privacy preserving is more important than earning money for some workers, but it may be the opposite case for others.

3.2. Insights. The design of our framework is based on the following insights.

3.2.1. Insight 1: Context Classification. Context is an abstract concept. According to the abovementioned definition, it can be any factor that needs to be considered in the crowd-sensing worker selection. However, these contexts are not semantically at the same level and can be divided into the following categories.

Constraints. Constraints are the minimum and basic requirements that can be set on the workers (e.g., spatial constraint, temporal constraint, and sensing capability constraint). They are the objective requirements from the task itself.

Worker-Side Factors. They are subjective factors considered when the worker decides whether to undertake a recommended task, for example, whether the reward (incentive) is attractive, whether the task falls into his interested domain, whether his privacy is well protected, and whether the workload is too heavy.

Task Properties. Task properties characterize basic information of a crowd-sensing task (e.g., title, description, reward, keywords, and assignment). They are commonly key-value pairs whose value can be assigned in the mediation platform by either the form-like user interface or the task specification language.

3.2.2. Insight 2: The Opportunity of Leveraging Worker's Participation History. When a crowd-sensing task is pushed to a worker, it may be undertaken or declined, which is referred to as the *outcome* in this paper. Over a period of time, the outcome, together with the corresponding worker-side factors and task properties, constitutes a worker's participation history.

Workers' participation history contains knowledge about how the worker-side factors and task properties affect the workers' decisions. Therefore, one basic idea in this paper is to leverage the participation history to improve the precision of future worker selection, making the task recommend to those who are more likely to accept it. Since WSelector must be integrated into an existing mediation platform (e.g., Amazon Mechanical Turk) in practical usage, we assume that our framework can access the participation history from the mediation platform.

3.3. Design Overview. With abovementioned challenges and insights, we design a novel worker selection framework, named WSelector. The basic idea for the design of WSelector is as follows: (1) since it is difficult for the task creator to define all contexts reasonably or properly, WSelector first provides a programming framework to help task creators define constraints for the worker selection; (2) WSelector adopts a two-phase process at runtime to select workers who are not only qualified but also more likely to undertake a crowd-sensing task. We demonstrate the system design in terms of programming time and runtime support as follows.

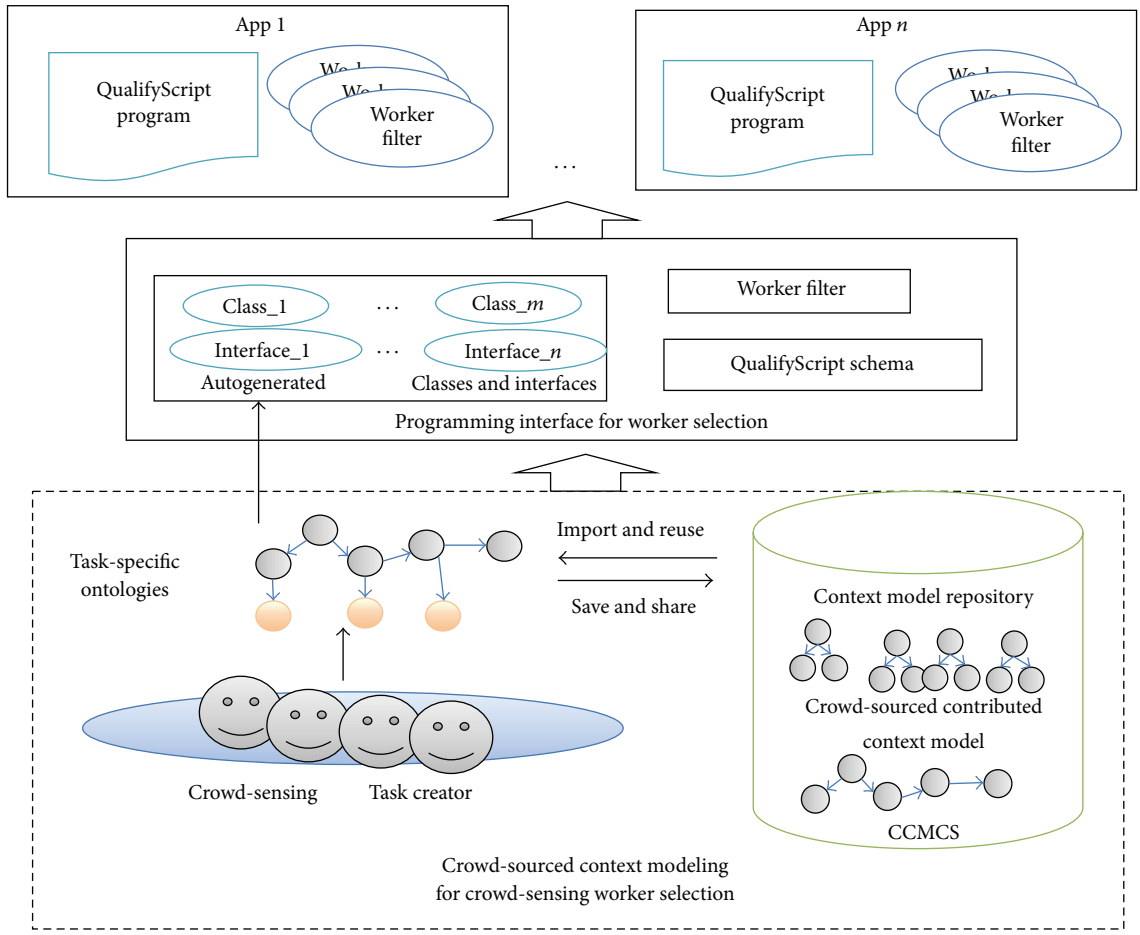

Figure 1: Programming time support.

3.3.1. Programming Time Support. A context-driven programming framework (see Figure 1) is used to assist the task creator define task properties and constraints.

First, this programming framework is based on a core context model and crowd-sourced context modeling mechanism. (1) We propose an ontology-based context model, named CCMCS (core context model for crowd-sensing), to define the most fundamental concepts for crowd-sensing. The objectives of the CCMCS include modeling a set of upper-level concepts and providing flexible extensibility to add specific concepts for different crowd-sensing tasks. In realistic crowd-sensing tasks, there are other task-dependent concepts needed to be characterized, which share common concepts that are modeled in CCMCS and differ significantly in detailed features. Our framework enables the task creators to build their task-specific ontologies by extending CCMCS. (2) To enable the reuse of context models, we also provide a crowd-sourced mechanism. When a task creator wants to create a context model, he can either extend the CCMCS or import existing models that are established and saved in the context model repository by other task creators. Since the creation and management of ontologies are a mature technology and there are many existing tools, our framework directly exploits the Protégé (http://protege.stanford.edu/), a free open-source Java tool, to support the creation and management of ontologies.

Second, the framework provides interfaces for task creators to define constraints for worker selection. (1) Our framework provides an abstract interface WorkerFilter(), and the task creator can create task-specific filters by extending this interface. To make the filter creation more efficient, the framework generates some JAVA components automatically, which can be directly imported to realizing task-specific filters. Task-specific ontology classes and their data properties defined by the task creator are automatically transformed into corresponding Java components including classes and interfaces. Our framework adopts the approach in [29] to implement the transformation from ontologies to Java components. (2) We propose QualifyScript, an XML-based language, to define the references of all the worker filters. The worker selection reference is defined between the <filter> and </filter> tags and refers to an executable filter implemented by the task creator. QualifyScript also provides other predefined tags, including title, description, reward, and assignment, to define task properties. It also allows task creator to define new tags.

Box 1 shows a QualifyScript program for the air quality report application.

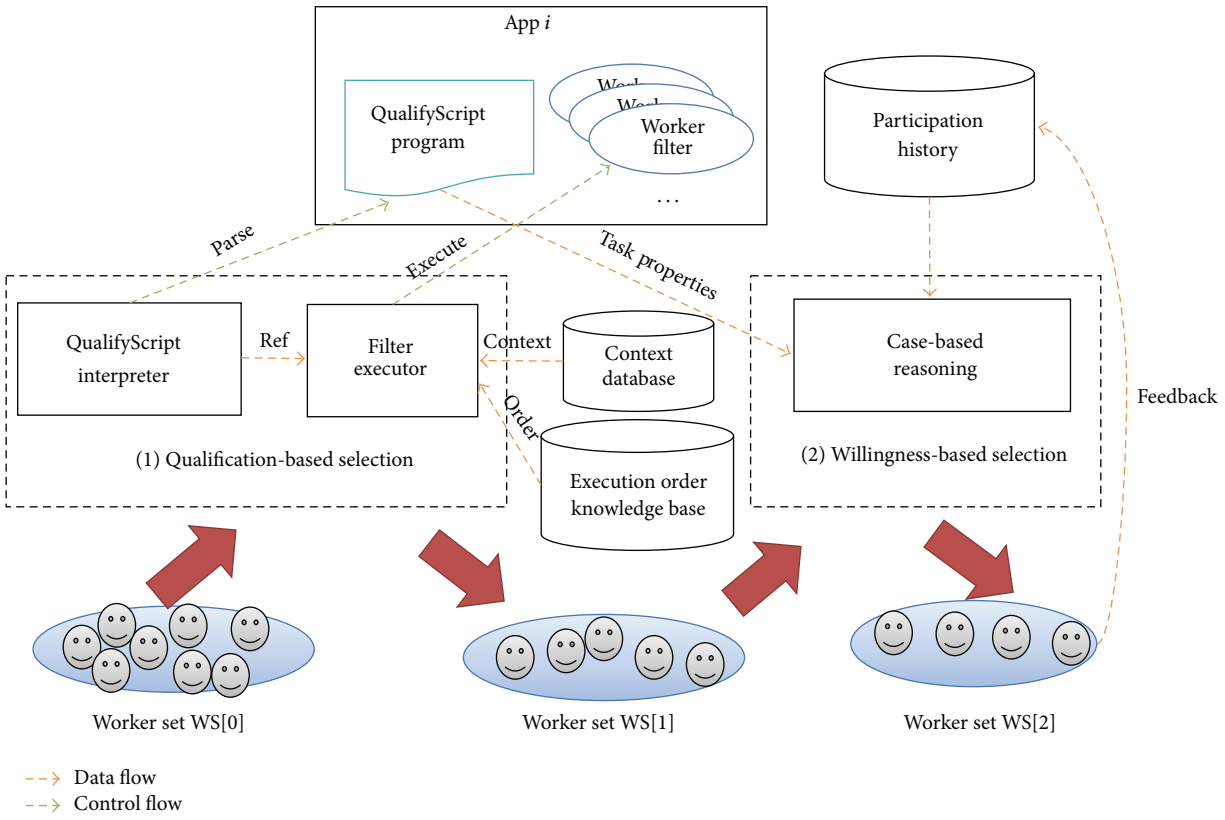

Figure 2: Runtime support.

```
<xml>
  <task_properties>
    <title> Air Quality Report </title>
    <descript>
        report PM 2.5 measurement in the scenic spot.
    </descript>
    <reward> 0.5 USD </reward>
    <assignments> 5 </assignments>
  </task_properties>
  <worker_filters>
    <filter> LocationFilter </filter>
    <filter> SenseCapFilter </filter>
  <worker_filters>
</xml>
```

Box 1: QualifyScript for air quality report task.

The advantage of our design is that it better supports software reuse and maintenance. First, existing filters created by others can be imported into a QualifyScript program very easily when defining a new task. Second, when a new constraint emerges for a predefined task, the task creator only has to implement a new filter and configure it in the QualifyScript without modifying and recompiling others.

3.3.2. Runtime Support. WSelector adopts a two-phase process at runtime for worker selection by taking both the predefined constraints and worker-side factors into account (see Figure 2).

Phase 1—Qualification-Based Selection. In this first phase, we propose a pipe filtering model to select workers who satisfy the predefined constraints. This is done through executing a flow of predefined filters one by one through a virtual pipeline at runtime.

The input for this stage is all registered users in the crowdsensing mediation platform, which is denoted as a worker set WS[0] in Figure 2, together with a predefined QualifyScript program and workers' filters of a certain task.

The qualification-based selection mainly consists of two steps: (1) QualifyScript interpreter parses the QualifyScript program into several references of worker filters and passes them to the filter executer. (2) The worker filters are then executed one by one by the executer (the filters are JARs (JAVA executable packet), while the executor is a component to invoke each JAR). The execution of the worker filters needs the current contexts of workers. For example, a worker filter f_i is to select workers who are currently in a certain place, and the runtime execution of f_i needs the location information of each candidate worker. The acquisition of contexts is the mediation platform's responsibility. We assume that real-time contexts are managed by the mediation platform in

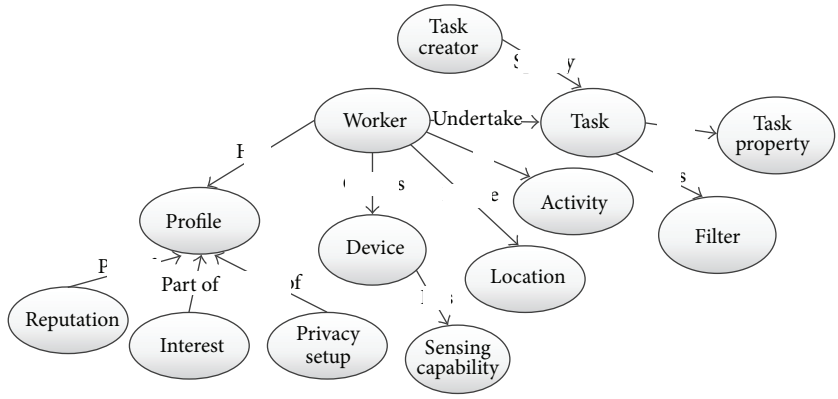

Figure 3: CCMCS: Core Context Model for Crowd-Sensing.

the context database, which our framework can access (see Figure 2).

Ultimately, at the end of the pipeline, we get the selected workers who are qualified for all predefined constraints, which is denoted as the worker set WS[1].

In the crowd-sensing paradigm, energy consumption and privacy violation are two key concerns for workers. It is more satisfactory if the mediation platform updates worker's dynamic or private-sensitive contexts in a lower frequency. For example, updating location information uses more smartphone battery and could lead to violation of privacy, thus workers may not want it to be collected too frequently. The optimized execution orders of filters can make the crowd-sensing mediation platform update the contexts in a more energy-saving and privacy-preserving way. We preestablish an execution order knowledge base (see Figure 2), in which the execution orders of some predefined contexts are specified. At runtime, worker filters will be executed one by one in the filter executer based on the predefined orders in the knowledge base. Filters relevant to contexts that are more static (e.g., those determined when the worker registered to the mediation platform or those which change very slowly) will be executed in front of those filters relevant to contexts that are more dynamic or privacy-sensitive. Take the air quality report task in Box 1 as an example. If the *LocationFilter* is executed at first, then the location information of all workers in WS[0] must be updated. However, if the sensing capability filter (*SenseCapFilter*) is executed before the *LocationFilter*, only workers whose devices are embedded with air quality sensors need to collect and upload their location information to the mediation platform. Therefore, in the execution order knowledge base, we specify that sensing capability filter is executed prior to location filter.

Phase 2—Willingness-Based Selection. After the qualification-based selection phase, if the number of candidates (i.e., |WS[1]|) is still larger than the number of assignments specified in the QualifyScript, the second phase (i.e., the willingness-based selection) will be started.

In this phase, WSelector further selects workers who are more likely to undertake the task. The input of this stage is the worker set WS[1], and the output is k workers in WS[1] who have the highest likelihood of actually undertaking a task if the system pushes it to them (denoted as WS[2]). The parameter k is the number of assignments defined in the QualifyScript program.

We achieve such a goal based on our second insight, which is leveraging the candidate worker's participation history over time to perform a more precise selection. To implement this idea, we propose a case-based reasoning approach to select those k workers, which will be introduced in great detail in Section 5. We assume that the mediation platform has been recording participation history and storing them in the case database, to which our framework can get access (see Figure 2).

4. Context Modeling for Crowd-Sensing

4.1. Core Context Model and Extensibility. We propose an ontology-based context model, named CCMCS (Core Context Model for Crowd-Sensing), to define basic concepts and their relationships in crowd-sensing worker selection (see Figure 3). Each entity in the context model is associated with its attributes (represented in owl:DatatypeProperty) and relations with other entities (represented in owl:ObjectProperty).

Due to space limitation, Figure 3 only shows owl:ObjectProperty. *Task*, *task creator*, and *worker* are the most basic concepts, to which other concepts are related. Generally speaking, each task has its *filters* defining constraints, *task property* including name, description, reward, and assignments. Each worker also has his/her *profile*, *location*, and carried *device*. In the CCMCS, we establish *reputation*, *interest,* and *privacy setup* as part of predefined profiles and *sensing capability* as predefined object property of carried device.

WSelector also provides extension ability for adding task-specific concepts based on the CCMCS. First, the built-in OWL property (such as owl:subClassOf and owl:partOf) allows for most of the extensions. Second, the task creator can

Table 2: Measurable worker-side factor.

Worker-side factor	How it is measured	Possible values	
		Values	Meaning
Incentive (Reward)	Measured by the reward specified in the QualifyScript program	**1**	0~10 US cents
		2	10~50 US cents
		3	50~100 US cents
		4	1~3 USDs
		5	more than 3 USDs
Interest	Measured by how many tasks a worker has undertaken in the same domain	**1**	0 tasks
		2	1~3 tasks
		3	3~5 tasks
		4	5~10 tasks
		5	More than 10 tasks
Data privacy sensitivity	Measured by the sensitivity of the data the worker must provide when to fulfill the task	**1**	Do not provide private data
		2	Only provide coarse-grained location data (e.g., @University of Florida)
		3	Provides very sensitive private data (e.g., sound data via microphone)

define its own OWL property (both the owl:ObjectProperty and owl:dataTypeproperty).

4.2. Crowd-Sourced Context Modeling. Our framework supports crowd-sourced context modeling. It allows a collaborative creation of context models and provides automated mechanisms for validation.

As Figure 1 shows, the context model repository is open to the public, into which everyone can contribute new context models. The repository automatically validates new models for using a unique name and in terms of the OWL (Web Ontology Language) grammar requirements. The automatic validity check ensures the integrity of all models that are published. After acceptance, all context models in the repository can directly be used by other task creators.

Even though they provide better support for context models reuse and for accelerating the task creation, crowd-sourced context modeling mechanisms come at a cost and overhead. Allowing everyone to submit models might lead to the fast growth in the size of repository, because each task creator creates the model that best fits his/her purpose. As an example it can be expected that there are many different models for "smartphone." When searching a specific model (e.g., smartphone), the task creator might be overwhelmed by getting numerous alternatives. Therefore, this paper proposes a mechanism to rank the context models based on their popularity. WSelector collects two types of different data as the metric to rate the popularity of context models. One is the number of downloads of a context model, and the other is the task creator's manual scoring for the model (from 0~10). When a task creator wants to search a model, the framework can sort the alternatives by either the number of downloads or the average scoring.

5. Willingness-Based Selection

In the willingness-based selection phase, we aim to select workers who are more likely to accept to undertake a certain task by leveraging worker's participation history over a period of time.

5.1. Measurable Worker-Side Factors. There are many worker-side factors that may affect a worker's decision to undertake or decline a certain task. However, the current implementation of WSelector takes only those measurable worker-side factors (in Table 2) into consideration. Here the term "measurable" means these factors can be measured by either the task property in the QualifyScript program or the worker behavior learned over time from the mediation platform.

5.2. Case-Based Reasoning Algorithm. We propose a case-based reasoning algorithm to fulfill the willingness-based selection. Although current implementation of the algorithm takes only those worker-side factors in Table 2 into account, the algorithm itself is not limited to these three factors and can be extended easily if additional measurable factors are used in the future.

5.2.1. Problem Formulation. The problem is formulated as follows.

Consider that $\mathrm{WS}_1(T_h) = \{\mathrm{wk}_1, \mathrm{wk}_2, \ldots, \mathrm{wk}_N\}$ is the output of the qualification-based selection phase and the input of willingness-based selection, where wk_i $(i = 1, 2, \ldots, N)$ is a worker satisfying the constraints of task T_h.

The goal is to find K $(K \leq N)$ workers from $\mathrm{WS}_1(T_h)$ who have the highest likelihood of accepting the task T_h. The output is $\mathrm{WS}_2(T_h) = \{\mathrm{wk}_{p_1}, \mathrm{wk}_{p_2}, \ldots, \mathrm{wk}_{p_k}\}$, where K is specified in the property "assignments" of the QualifyScript program by the task creators. For example, in the example of Box 1, $K = 5$.

5.2.2. The Algorithm. We first define the concept of historical case.

Consider that $\overrightarrow{\mathrm{WF}}_{i\theta} = (c_{1\theta}, c_{2\theta}, \ldots, c_{m\theta})$ is a *worker-side factor vector* of task T_θ on worker i. For example, in the current implementation of the framework, $m = 3$, and

$(1, 2, 3)$ means that a certain task provides the reward of 0~10 US cents; the worker i has undertaken 1~3 tasks in the same domain; and the task requires the worker to provide very sensitive private data. A *historical case* is defined as $(i, \theta, \overrightarrow{\mathrm{WF}}_{i\theta}, \lambda)$, where i and θ are the identity of a worker and a task, respectively, $\overrightarrow{\mathrm{WF}}_{i\theta}$ is a m-dimensional worker-side factor vector, and λ is the two-valued outcome (accept/decline).

Our case-based reasoning algorithm consists of the following two steps.

Step 1 (case selection). Since the influence of different worker-side factors varies from one worker to another, the algorithm first selects historical cases of worker i to compute the likelihood of a worker i to undertake the task T_h. Using selected cases, the following two reference matrices are established, which are the positive reference matrices $(\overrightarrow{\mathrm{RP}}_1, \overrightarrow{\mathrm{RP}}_2, \ldots, \overrightarrow{\mathrm{RP}}_{x(i)})^{\mathbf{T}}$ and the negative reference matrix $(\overrightarrow{\mathrm{RN}}_1, \overrightarrow{\mathrm{RN}}_2, \ldots, \overrightarrow{\mathrm{RN}}_{y(i)})^{\mathbf{T}}$, where $x(i)$ and $y(i)$ are the number of "accept" (positive) and "decline" (negative) cases, respectively. $\overrightarrow{\mathrm{RP}}_j = (a_{j1}, a_{j2}, \ldots, a_{jm})$ is a worker-side factor vector whose corresponding outcome λ is "accept." $\overrightarrow{\mathrm{RN}}_j = (b_{j1}, b_{j2}, \ldots, b_{jm})$ is a worker-side factor vector whose corresponding feedback λ is "decline":

$$\begin{aligned}
&\left(\overrightarrow{\mathrm{RP}}_1, \overrightarrow{\mathrm{RP}}_2, \ldots, \overrightarrow{\mathrm{RP}}_{x(i)}\right)^{\mathbf{T}} \\
&\quad = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ a_{x(i)1} & a_{x(i)2} & \cdots & a_{x(i)m} \end{pmatrix}, \\
&\left(\overrightarrow{\mathrm{RN}}_1, \overrightarrow{\mathrm{RN}}_2, \ldots, \overrightarrow{\mathrm{RN}}_{y(i)}\right)^{\mathbf{T}} \\
&\quad = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{y(i)1} & b_{y(i)2} & \cdots & b_{y(i)m} \end{pmatrix}.
\end{aligned} \tag{1}$$

Step 2 (likelihood measurement and worker selection). We use $U(i, h)$ to measure the likelihood of worker i to accept task T_h (see (2)), where $\mathrm{Dist}(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RN}}_j)$ is the Euclidean distance between $\overrightarrow{\mathrm{WF}}_{ih}$ and $\overrightarrow{\mathrm{RN}}_j$ and $\mathrm{Dist}(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RP}}_j)$ is the Euclidean distance between $\overrightarrow{\mathrm{WF}}_{ih}$ and $\overrightarrow{\mathrm{RP}}_j$. K workers with maximum $U(i, h)$ are what we want to select:

$$\begin{aligned}
U(i, h) = &\sum_{j=1}^{y(i)} \mathrm{Dist}\left(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RN}}_j\right) \\
&- \sum_{j=1}^{x(i)} \mathrm{Dist}\left(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RP}}_j\right).
\end{aligned} \tag{2}$$

The intuition behind this function is that the more likely a worker i is to accept task T_h, the closer $\overrightarrow{\mathrm{WF}}_{ih}$ is to positive cases and the more distant it is from negative cases.

Note that different worker-side factors could have different impact on deciding the outcome; the weights of different factors have to be considered (see (3)) when calculating the distance, where $w_i \in (0, 1)$ is the weight of ith worker-side factor:

$$\begin{aligned}
\mathrm{Dist}\left(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RN}}_j\right) &= \sqrt{\sum_{p=1}^{m} \left(c_{ph} - b_{jp}\right)^2 * w_p}, \\
\mathrm{Dist}\left(\overrightarrow{\mathrm{WF}}_{ih}, \overrightarrow{\mathrm{RP}}_j\right) &= \sqrt{\sum_{p=1}^{m} \left(c_{ph} - a_{jp}\right)^2 * w_p}.
\end{aligned} \tag{3}$$

Now the key problem is how to precompute the weight w_p $(p = 1, 2, \ldots, m)$. We propose a weight calculation algorithm based on the *sensitivity analysis principle* [37]. This principle is to identify the key variables for a certain target, by calculating the effect of variable's change on the target's change. This principle mainly consists of the following steps:

(a) Determine the variables and target to be analyzed according to the problem.

(b) Vary only one variable, observing and measuring the change of target.

(c) Repeat the analysis in (b) for each variable repetitively for calculating the corresponding effect on the target.

Based on the above sensitivity analysis principle, we propose a *personalized weight calculation* algorithm according to our problem. Some key points of this algorithm are explained as follows.

(1) The variables are worker-side factors, and the target is the outcome (accept/decline). Matrix $\mathbf{A} = (\vec{Q}_1, \vec{Q}_2, \ldots, \vec{Q}_{[x(i)+y(i)]})^{\mathbf{T}}$ is all cases of worker i, which is the mergences of positive reference matrix $(\overrightarrow{\mathrm{RP}}_1, \overrightarrow{\mathrm{RP}}_2, \ldots, \overrightarrow{\mathrm{RP}}_{x(i)})^{\mathbf{T}}$ and the negative reference matrix $(\overrightarrow{\mathrm{RN}}_1, \overrightarrow{\mathrm{RN}}_2, \ldots, \overrightarrow{\mathrm{RN}}_{y(i)})^{\mathbf{T}}$.

(2) Lines 3~19 calculate the weight of a certain worker-side factor x_k by changing x_k and remaining others unchanged.

(3) In lines 11~12, $L = \mathrm{Max}\{q_{1k}, q_{2k}, \ldots, q_{(x(i)+y(i))k}\} - \mathrm{Min}\{q_{1k}, q_{2k}, \ldots, q_{(x(i)+y(i))k}\}$ is the maximum difference among the possible values of the kth variable, and $L/|q_{jk} - q_{lk}|$ measures the effect of variable x_k's change on the target's change. It is reasonable, because when $|q_{jk} - q_{lk}|$ is smaller and the target has changed, it means that even the slightest change of x_k is enough to leading to the change of target (indicating that this variable's weight is heavier).

(4) We adjust the weight to make sure that their sum is equal to 1 (in line 20).

Input: Matrix **A** is the merging of positive and negative reference matrix of the worker i, and vector **B** is the corresponding feedbacks ($\vec{Q}_j$ corresponds to λ_j).

$$\mathbf{A} = (\vec{Q}_1, \vec{Q}_2, \ldots, \vec{Q}_{x(i)+y(i)})^{\mathbf{T}} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ q_{(x(i)+y(i))1} & q_{(x(i)+y(i))2} & \cdots & q_{x((i)+y(i))m} \end{pmatrix}$$

$\mathbf{B} = (\lambda_1, \lambda_2, \ldots, \lambda_{(x((i)+y(i))})^{\mathbf{T}}$ $\lambda_i \in \{\text{accept, decline}\}$.

Output: w_i $(i = 1, 2, \ldots, m)$

(1) **ALGORITHM BEGIN**
(2) float total = 0, effect = 0;
(3) **FOR** ($k = 1$ to $k = m$){
(4) /* Calculate the weight w_k. */
(5) **FOR** (from $j = 1$ to $j = x(i) + y(i)$)
(6) **FOR** (from $l = j + 1$ to $l = x(i) + y(i)$)
(7) **IF** (& $q_{jk} \neq q_{lk}$ &
(8) other components of $\vec{Q}_j$ and $\vec{Q}_l$ are identical)
(9) total ++;
(10) **IF** ($\lambda_j \neq \lambda_l$)
(11) $L = \text{Max}\{q_{1k}, q_{2k}, \ldots, q_{(x(i)+y(i))k}\}-$
(12) $\text{Min}\{q_{1k}, q_{2k}, \ldots, q_{(x(i)+y(i))k}\}$.
(13) effect = effect + $L/|q_{jk} - q_{lk}|$;
(14) **END FOR**
(15) **END FOR**
(16) w_i = effect/total.
(17) /* clear the parameters for calculating next weight. */
(18) total = 0, effect = 0.
(19) **END FOR**
(20) **FOR** ($j = 1$ to $j = m$) $w_j = w_j / \sum_1^m w_i$
(21) //Adjust the weight to make their sum to be 1
(22) **ALGORITHM END**

Algorithm 1: Personalized weight calculation algorithm.

5.2.3. Strategy for Cold-Start Problem. We may encounter the cold-start problem when a worker just registered in a crowd-sensing mediation platform. In this situation, the number of one's historical cases is zero and our framework knows nothing about the new worker.

We deal with the cold-start problem by leveraging the cases of other workers, because users tend to make similar decisions under similar contexts. The approach is almost similar to Algorithm 1, but is different in the following two aspects.

(1) Case Selection. In the case selection step, we select historical cases of other workers whose worker-side factor vector is identical to the current context vector $\overrightarrow{\text{WF}}_{ih}$ of worker i and task h.

(2) General Weight Calculation. The weight calculation algorithm is almost the same as the personalized weight calculation algorithm, but the input is changed to the entire datasets (all cases from all workers). The calculated weights are referred to as the *general weight*, which will be used in the likelihood measurement and worker selection.

5.3. Iterative Selection Strategy

5.3.1. Problem Statement. After the execution of willingness-based worker selection algorithm, K workers will be selected and the task is pushed to them. However, in real-world circumstances, the selected workers may not be able to accomplish the task due to some unexpected reasons.

For example, a candidate worker, named Tom, has a very high probability of accepting a certain crowd-sensing task according to his historical participation records and the attributes of the task. Thus, our willingness-based worker selection algorithm selects him, and the platform pushes the task to him. Nevertheless, Tom has something urgent to do all of a sudden, so he declines the pushed task.

Therefore, sometimes the number of workers who actually accept and accomplish the task may be less than K, which does not satisfy the requirement of the task creator.

5.3.2. Strategy Description. To deal with above-stated problem, this paper proposes an iterative worker selection strategy. The main idea of the strategy is to divide the willingness-based selection phase into the iterative execution of several

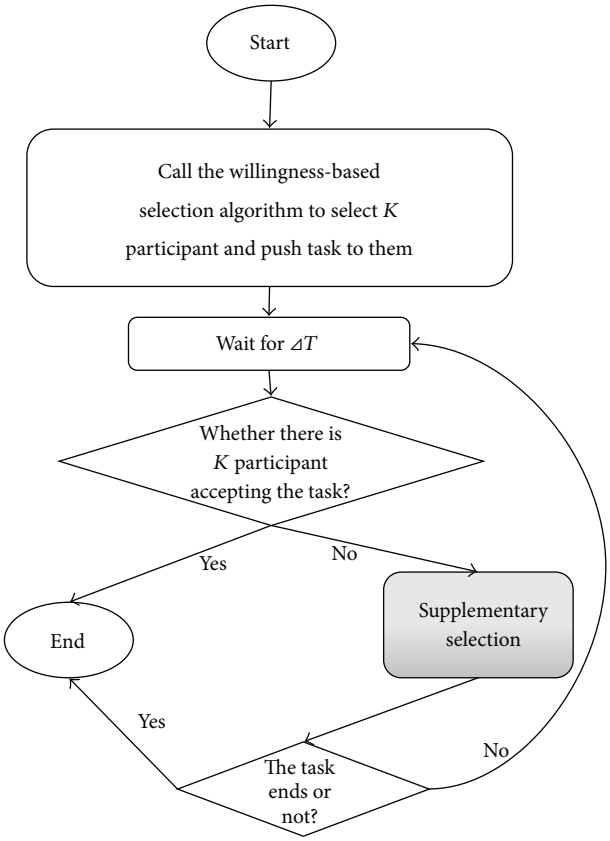

Figure 4: Iterative selection process.

substeps, by which it can achieve a high probability that there are K workers actually accepting the task.

Figure 4 shows the workflow of the strategy. Before the execution of workflow, willingness-based worker selection algorithm will calculate the all qualified candidate workers' probability of accepting a task named tsk. Then the workflow is executed as follows:

(1) Select K workers who are more likely to accept the task.

(2) Observe the result after a short period of time (denoted as ΔT). If the total number of workers who accept the task reaches K or the task is out of time, then the entire process comes to an end. For a certain task, we denote its entire duration as $[\theta_{\text{start}}, \theta_{\text{end}}]$, where θ_{start} and θ_{end} are the start and end time point. We assume that $\theta_{\text{start}} - \theta_{\text{end}} > 2\Delta T$, ensuring that our iterative process has at least one chance to reselect workers.

(3) Reselect certain number of workers (supplementary selection) and then go to (2). Here, the reselected workers are those who have not been selected in previous iterations and with highest likelihood of accepting the task.

Now the key problem is how many workers should be selected in each of the iterations.

We denote the number of selected workers in each of the iterations as x_i ($i = 1, 2, \ldots,$ max_round), where max_round is the maximum number of iterations decided by $(\theta_{\text{start}} - \theta_{\text{end}})/\Delta T$.

For the ith iteration, the number of reselected workers λ_i should satisfy the following formula:

$$\frac{\sum_{j=1}^{j=i-1} x_j}{\alpha * \sum_{j=1}^{j=i-1} \lambda_j} = \frac{E(x_i)}{\lambda_i}, \tag{4}$$

where $\sum_{j=1}^{j=i-1} \lambda_j$ is the total number of selected workers in the previous $i-1$ iterations, $\sum_{j=1}^{j=i-1} x_j$ is the total number of workers who have accepted the task, $E(x_i)$ is the expectation of the number of workers who will accept the task in the ith iteration. Here we use the concept of expectation, because at this time we do not know how many reselected workers in the ith iteration will eventually accept the task. In each of the iterations, we use the greedy strategy hoping that after the iteration the total number of workers accepting the task reaches K. Thus, $E(x_i) = K - \sum_{j=1}^{j=i-1} x_j$. The parameter α is used to characterize the probability of accepting the task by analyzing the historical participation record, and it is calculated as follows:

$$\alpha = \frac{\sum_{j=1}^{j=\text{Sum}} U(w_j, \text{tsk})}{\sum_{j=\text{Sum}+1}^{j=\text{Sum}+E(x_i)+1} U(w_j, \text{tsk})}, \quad \text{where Sum} = \sum_{j=1}^{j=i-1} x_j. \tag{5}$$

Therefore, the number of workers that should be selected in each of the iterations is calculated as

$$\lambda_i = \frac{\alpha * \left(K - \sum_{j=1}^{j=i-1} x_j\right) * \sum_{j=1}^{j=i-1} \lambda_j}{\sum_{j=1}^{j=i-1} x_j}. \tag{6}$$

The probability that the total number of workers accepting the task reaches K after the ith iteration is denoted as p_i ($i = 1, 2, \ldots,$ max_round). Then after max_round iterations, the probability that total number of workers is still less than K is Prob (failure) $= (1 - p_1) * (1 - p_2) * \cdots * (1 - p_{\text{max\_round}})$.

Thus after max_round iterations, the probability that total number of workers reaches K is Prob (success) $= 1 - (1 - p_1) * (1 - p_2) * \cdots * (1 - p_{\text{max\_round}})$.

As $p_i \in (0, 1)$, $1 - p_i \in (0, 1)$, Prob (success) will get close to 1 with the increase of the number of iterations. On the other hand, max_round is decided by $(\theta_{\text{start}} - \theta_{\text{end}})/\Delta T$. Therefore, the iterative selection strategy described above can make Prob (success) get close to 1 by reducing ΔT.

6. Evaluation

In this section, we evaluate the qualification-based and willingness-based selection of WSelector. The goals of our evaluation are as follows. (1) For the qualification-based selection, we first demonstrate the extensibility of the core

Table 3: Constraint for worker selection.

Crowd-sensing task	Constraint for worker selection
Petrol Watch [30]	Location, sensing capability (camera)
Haze Watch [31]	Location, sensing capability (air quality sensor)
Citizen Journalist [32]	Location, sensing capability (camera)
EarPhone [33]	Location, sensing capability (microphone), privacy setup (microphone can be accessed)
Nericell [34]	Location, activity (driving), sensing capability (accelerometer), privacy setup (accelerometer can be accessed)
Bus Waiting Time Prediction [35]	Activity (riding bus), sensing capability (accelerometer)
QTime [36]	Location, activity (queuing), sensing capability (accelerometer)

context model and the expressiveness of the programming interfaces. Then we evaluate the correctness of the runtime selection. (2) For the willingness-based selection, we evaluate the validity of our case-based reasoning algorithm.

Our experiments are conducted on a prototype of WSelector. The core context model is created using the Protégé [38], a free and open-source ontology editor. The management of the crowd-sourced context modeling process is implemented with the help of OWL API [39]. The programming interfaces and the runtime environments (including the QualifyScript interpreter, filter executor, and case-based reasoning module) are implemented using Java. The context database and case database are established and managed by the MySQL system, and Java SDK is imported for accessing the MySQL database.

6.1. Qualification-Based Selection. We have implemented worker selection module of 10 crowd-sensing tasks, among which 7 tasks are from literature review and other three are borrowed from the examples we mentioned in the introduction section. The constraints for worker selection of these tasks are listed in Table 3.

In Table 3, we assume that all mobile devices are all embedded with modules for localization, so that we do not include it in the sensing capability. Since this paper does not focus on the context collection from mobile nodes and assumes this to be the crowd-sensing mediation platform's duty, the client side of these crowd-sensing tasks ranges from mobile phone to wearable devices.

By implementing the worker selection of all these tasks, we can report the results as follows.

(1) All these constraints can be modeled by our framework successfully, by either directly importing the classes and properties in the core context model or extending by the user-defined ones.

(2) Our programming interfaces are expressive enough for supporting the task creator to define relevant worker filters. The ontology classes and properties in the task-specific ontology model are transformed into Java classes and interfaces, which are used to develop relevant worker filters. Finally, we define the reference of worker filters in the QualifyScript program. Then we randomly generated 100 testing cases for each task to evaluate the result of qualification-based selection at runtime. Each testing case corresponds to one candidate worker and contains the values for different constraints. We manually label each worker as "selected" or "not selected" based on the constraints in Table 3, which are taken as the ground truth. By running the implemented worker selection module in the runtime environments of WSelector, we demonstrate that all the $10 * 100 = 1000$ selection results obey the predefined constraints.

6.2. Willingness-Based Selection

6.2.1. Data Collection. We post a questionnaire on an online platform to generate the dataset [40]. First, we assume that there is a crowd-sensing task with basic description. Then we design 15 questions, each of which ask for the outcome under different combination of worker-side factors. Second, we invite 26 volunteers (including the undergraduate, graduate students, and faculties in our institute) to respond to this questionnaire. Third, we collect all these questionnaires and generated the dataset. The generated dataset consists of $75 * 26 = 1950$ historical cases (each question and its answer can generate 5 cases of a worker, so that each worker has $15 * 5 = 75$ cases and the overall dataset consists of $75 * 26$ cases).

6.2.2. Experimental Methodology. After the dataset has been generated, we design a 6-round experiment to evaluate the case-based reasoning algorithm for willingness-based worker selection. The generated datasets are used to establish both the case database, which is the input of our algorithm, and testing cases containing the ground truth. Our 6-round experiment is summarized in Table 4.

(1) The First Round Is for the Baseline Method. It is assumed that none of the cases have been preacquired, so that we can only randomly select k workers from 26 candidates. This round of experiment consists of the following steps: (a) randomly assign value to the components of worker-side factor vector $\overrightarrow{\mathrm{WF}}_{ih}$. (b) Perform the baseline method, that is, guessing k workers randomly. (c) Check how many of those guessed workers will undertake the task based on the ground truth in the datasets. (d) Perform (a)~(c) for 10 times and calculate the average number of right-selected workers.

(2) The 2nd~6th Round of the Experiments Is to Test Our Case-Based Reasoning Algorithm When the Number of Cases in the Case Database Changes. (1) The 2nd~4th round of

Table 4: Summary of 6-round experiment.

Round	The number of cases as historical data	Selection approach
1st round	Zero	Baseline method (random selection)
2nd round	Each worker has 25 cases	Approach in Section 5.2.2
3rd round	Each worker has 50 cases	Approach in Section 5.2.2
4th round	Each worker has 75 cases	Approach in Section 5.2.2
5th round	Two workers have no historical cases Each of the other workers has 50 cases	Approach in Section 5.2.2 + Approach in Section 5.2.3
6th round	Five workers have no historical cases Each of the other workers has 50 cases	Approach in Section 5.2.2 + Approach in Section 5.2.3

experiments assumes that each of 26 candidate workers has preacquired cases, and the difference among these three rounds is the number of cases. In these three rounds, we calculate the personalized weight based on the approach in Section 5.2.2. (2) The 5th~6th round assumes that some workers do not have preacquired cases (3 and 5 workers, resp.). In these two rounds, we will use our mechanism in Section 5.2.3 for workers without historical cases. In both 2nd~6th rounds, we first randomly generated 100 worker-side factor vectors as testing cases. Then for each vector, we changed the second component (i.e., interest) randomly since each candidate worker's interest for the same task may be different and kept the other two components (i.e., reward and data privacy sensitivity) unchanged since they are the same among workers for a specific task.

6.2.3. Experimental Results. The willingness-based selection accuracy of each round is measured by Accuracy(i) $= T(i)/k$ ($i = 1, 2, \ldots, 6$), where $T(i)$ is the number of workers in the ith round who will undertake the task based on the ground truth and k is the number of selected workers.

Since the number of selected workers (i.e., k) has impact on the selection accuracy, we conduct our 6-round experiments for 3 times by changing k from 10 and 15 to 20, respectively. Hence we have completed $3 * 6 = 18$ rounds of experiments in total, whose selection accuracy is demonstrated in Figure 5.

According to the observation of the results, we can draw the following conclusions:

(1) Our case-based reasoning algorithm outperforms the baseline method no matter $k = 10$, 15, or 20 and no matter the number of cases as historical data is 25, 50, or 75. Therefore, it shows the effectiveness of our algorithm in different situations.

(2) The smaller k is, the bigger the increase in selection accuracy of our algorithm is compared to the baseline method.

(3) Compared to the 2nd, 3rd, and 4th rounds, we can see that the selection accuracy of our algorithm is improved with the increasing of the number of historical cases.

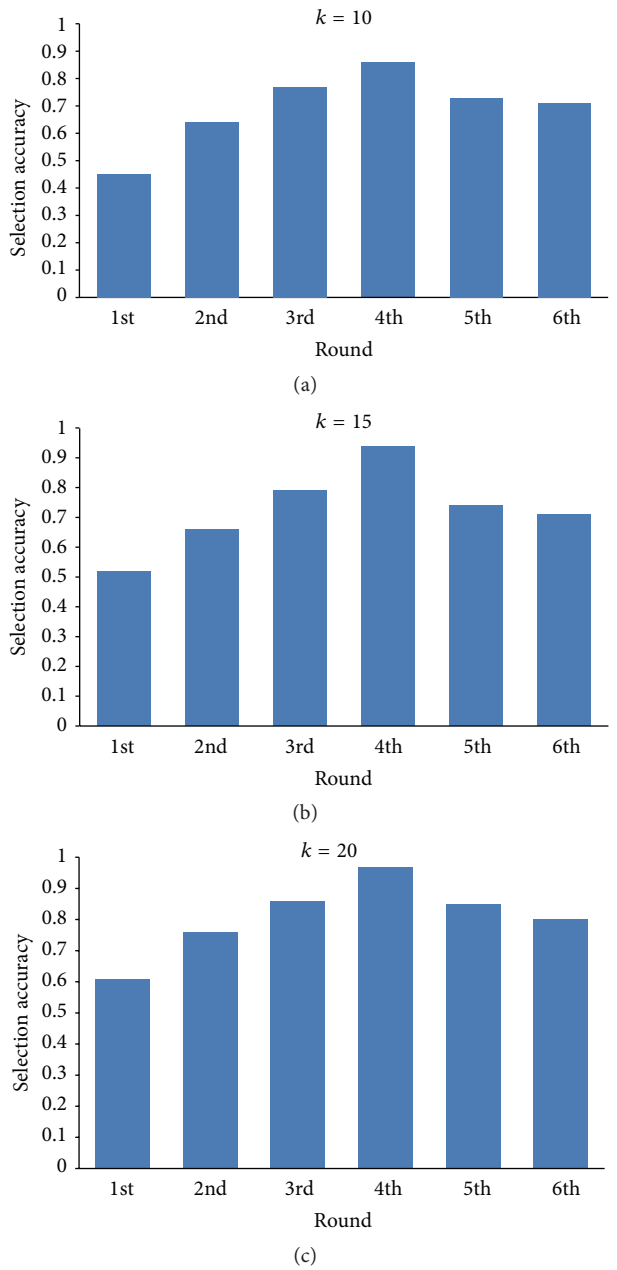

Figure 5: Selection accuracy of different rounds.

(4) The experimental result of 5th and 6th round shows that our mechanism in Section 5.2.3 is effective to solve the cold-start problem to some extent. Although the accuracy of 5th and 6th round is lower than the 3rd round, it still significantly outperforms the baseline method.

7. Discussion

In this section, we discuss some limitations of WSelector and additional important issues, some of which will be the subject of our future work.

7.1. Access to Mediation Platforms. A fundamental assumption in this paper is that WSelector can get access to the participation history in the mediation platform. In practical usage, WSelector is to be integrated into an existing crowd-sensing mediation platform. If our framework is effective enough in selecting appropriate workers, it is reasonable to assume that mediation platforms will provide participation history APIs, to enable a loose and inexpensive integration. At the same time, in our future work, we plan to propose a simple API of the same, which could guide and encourage existing mediation platforms to implement it. Besides, device's functionalities/sensing capabilities are determined when the worker registers to a certain crowd-sensing platform. When integrated to the platform, our framework can get the device's functionalities of each worker.

7.2. The Number of Worker-Side Factors. Although current implementation and evaluation of the case-based reasoning algorithm only take three worker-side factors into account, the algorithm itself is not limited to these factors only. It can be extended easily if we find new measurable ones in future work. For example, if a mediation platform requires task creator to assign value for a new task property (e.g., the approximate workload) when specifying the task, we can easily add it as the fourth factor for willingness-based selection. In our future work, we will analyze and propose a broader set of worker-side factors. We will also include additional factors that we may learn about from the literature. However, when more factors can be added, there is a new challenge, that is, how to determine which subset of factors is more significant in selecting workers? We plan to exploit feature selection mechanisms in the future work.

7.3. Dynamic Adjustment of the Number of Selected Workers. This paper assumes that all workers are selected before they start to undertake the task. However, not all crowd-sensing tasks obey this assumption. For example, a crowd-sensing task may require that selected workers must meet certain geographical coverage rather than reaching certain number in total. In this case, it is a better strategy to dynamically adjust the number of selected workers online according to the current distribution of workers who have already fulfilled the task. Therefore, we plan to propose a dynamic worker selection mechanism in the future work to make WSelector able to handle more crowd-sensing tasks.

7.4. Fine-Grained Privacy Measurement. In the current implementation of WSelector, we consider privacy as one of three contexts in the willingness-based selection. However, since our main contribution mainly lies in the case-based reasoning algorithm, we only measure privacy factor in a very simple way by giving them three possible values. However, the measurement of the privacy factor is much more complex in reality. We plan to leverage the model proposed in [41] to measure the privacy factor in a more fine-grained manner.

8. Conclusion

In this paper, we proposed a novel worker selection framework, named WSelector, to select appropriate workers for crowd-sensing more precisely by taking various contexts into account. It first provides programming time support to help task creator define all constraints. Then a two-phase process was adopted at runtime to select workers who not only are qualified but also have higher possibility to undertake a crowd-sensing task. Evaluations with 11 crowd-sensing tasks indicate the expressiveness of our core ontology model and programming interface. Besides, evaluations with a questionnaire-generated dataset show that our case-based reasoning algorithm outperforms the baseline method.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is funded by the National High Technology Research and Development Program of China (863) under Grant no. 2013AA01A605. Besides, this research is supported by NSFC Grant (no. 61572048) and Microsoft Collaboration Research Grant.

References

[1] Amazon mechanical turk, https://www.mturk.com/.

[2] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: a programming framework for crowd-sensing applications," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 337–350, ACM, Lake District, UK, June 2012.

[3] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: answering queries with crowdsourcing," in *Proceedings of the ACM SIGMOD International Conference on Management of data (SIGMOD '11)*, pp. 61–72, Athens, Greece, June 2011.

[4] T. Yan, V. Kumar, and D. Ganesan, "CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, pp. 77–90, ACM, San Francisco, Calif, USA, June 2010.

[5] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "TurKit: human computation algorithms on mechanical turk," in *Proceedings of the Proceedings of the 23nd Annual ACM Symposium*

on User Interface Software and Technology, pp. 57–66, ACM, October 2010.

[6] S. Reddy, D. Estrin, and M. Srivastava, "Recruitment framework for participatory sensing data collections," in *Pervasive Computing*, vol. 6030 of *Lecture Notes in Computer Science*, pp. 138–155, Springer, Berlin, Germany, 2010.

[7] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: platform for remote sensing using smartphones," in *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, pp. 63–76, ACM, June 2010.

[8] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: privacy-aware people-centric sensing," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pp. 211–224, ACM, usa, June 2008.

[9] G. Cardone, L. Foschini, P. Bellavista et al., "Fostering participaction in smart cities: a geo-social crowdsensing platform," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 112–119, 2013.

[10] D. Zhang, H. Xiong, L. Wang, and G. Chen, "CrowdRecruiter: Selecting participants for piggyback crowdsensing under probabilistic coverage constraint," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 703–714, ACM, September 2014.

[11] S. Reddy, K. Shilton, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using context annotated mobility profiles to recruit data collectors in participatory sensing," in *Location and Context Awareness*, pp. 52–69, Springer, Berlin, Germany, 2009.

[12] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner, "Demo abstract: mCrowd—a platform for mobile crowdsourcing," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 347–348, ACM, Berkeley, Calif, USA, November 2009.

[13] A. Joki, J. Burke, and D. Estrin, "Campaignr: a framework for participatory data collection on mobile phones," Tech. Rep., UCLA Center for Embedded Network Sensing, 2007.

[14] E. Cuervo, P. Gilbert, B. Wu, and L. P. Cox, "Crowdlab: an architecture for volunteer mobile testbeds," in *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS '11)*, Bangalore, India, January 2011.

[15] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan, "Lowering the barriers to large-scale mobile crowdsensing," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile '13)*, ACM, Jekyll Island, Georgia, February 2013.

[16] Z. Song, B. Zhang, C. H. Liu, A. V. Vasilakos, J. Ma, and W. Wang, "QoI-aware energy-efficient participant selection," in *Proceedings of the 11th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON '14)*, pp. 248–256, IEEE, Singapore, July 2014.

[17] S. He, D.-H. Shin, J. Zhang, and J. Chen, "Toward optimal allocation of location dependent tasks in crowdsensing," in *Proceedings of the 33rd IEEE Conference on Computer Communications (INFOCOM '14)*, pp. 745–753, IEEE, Toronto, Canada, May 2014.

[18] P. Öztürk and A. Aamodt, "Towards a model of context for case-based diagnostic problem solving," in *Proceedings of the 1st International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT '97)*, pp. 198–208, Rio de Janeiro, Brazil, February 1997.

[19] T. Strang, *Service interoperability in ubiquitous computing environments [Ph.D. thesis]*, Ludwig-Maximilians-University Munich, Munich, Germany, 2003.

[20] J. De Bruijn, "Using ontologies—enabling knowledge sharing and reuse on the semantic web," Tech. Rep. DERI-2003-10-29, Digital Enterprise Research Institute (DERI), Innsbruck, Austria, 2003.

[21] T. Strang, C. Linnhoff-Popien, and K. Frank, "CoOL: a context ontology language to enable contextual interoperability," in *Distributed Applications and Interoperable Systems*, vol. 2893 of *Lecture Notes in Computer Science*, pp. 236–247, Springer, Berlin, Germany, 2003.

[22] T. Strang, C. Linnhoff-Popien, and K. Frank, "Applications of a context ontology language," in *Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SoftCom '03)*, pp. 14–18, Split, Croatia, October 2003.

[23] T. Gu, X. H. Wang, H. K. Peng, and D. Q. Zhang, "Ontology based context modeling and reasoning using OWL," in *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS '04)*, San Diego, Calif, USA, January 2004.

[24] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology based context modeling and reasoning using OWL," in *Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops (PerCom '04)*, pp. 18–22, IEEE, Orlando, Fla, USA, March 2004.

[25] M.-O. Pahl and G. Carle, "Crowdsourced context-modeling as key to future smart spaces," in *Proceedings of the Network Operations and Management Symposium (NOMS' 14)*, pp. 1–8, IEEE, Krakow, Poland, May 2014.

[26] C. W. Holsapple and K. D. Joshi, "A collaborative approach to ontology design," *Communications of the ACM*, vol. 45, no. 2, pp. 42–47, 2002.

[27] S. Karapiperis and D. Apostolou, "Consensus building in collaborative ontology engineering processes," *Journal of Universal Knowledge Management*, vol. 1, no. 3, pp. 199–216, 2006.

[28] V. Ludovici, F. Smith, and F. Taglino, "Collaborative ontology building in virtual innovation factories," in *Proceedings of the International Conference on Collaboration Technologies and Systems (CTS '13)*, pp. 443–450, San Diego, Calif, USA, May 2013.

[29] A. Kalyanpur, D. J. Pastor, S. Battle, and J. A. Padget, "Automatic mapping of OWL ontologies into java," in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE '04)*, vol. 4, pp. 98–103, Alberta, Canada, June 2004.

[30] Y. F. Dong, S. S. Kanhere, C. T. Chou, and N. Bulusu, "Automatic collection of fuel prices from a network of mobile cameras," in *Distributed Computing in Sensor Systems: 4th IEEE International Conference, DCOSS 2008 Santorini Island, Greece, June 11–14, 2008 Proceedings*, vol. 5067 of *Lecture Notes in Computer Science*, pp. 140–156, Springer, Berlin, Germany, 2008.

[31] V. Sivaraman, J. Carrapetta, K. Hu, and B. G. Luxan, "HazeWatch: a participatory sensor system for monitoring air pollution in Sydney," in *Proceedings of the IEEE 38th Conference on Local Computer Networks Workshops (LCN '13)*, pp. 56–64, IEEE Computer Society, Sydney, Australia, October 2013.

[32] R. Sasank, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. Srivastava, "Biketastic: sensing and mapping for better biking," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, 1820, 1817 pages, Atlanta, Ga, USA, April 2010.

[33] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Ear-phone: an end-to-end participatory urban noise mapping system," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '10)*, pp. 105–116, ACM, Stockholm, Sweden, April 2010.

[34] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*, pp. 323–336, Raleigh, NC, USA, November 2008.

[35] P. Zhou, Y. Zheng, and M. Li, "How long to wait: predicting bus arrival time with mobile phone based participatory sensing," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 379–392, ACM, June 2012.

[36] Y. Wang, J. Wang, and X. Zhang, "QTime: a queuing-time notification system based on participatory sensing data," in *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference (COMPSAC '13)*, pp. 770–777, IEEE Computer Society, 2013.

[37] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*, John Wiley & Sons, 2004.

[38] http://protege.stanford.edu/.

[39] S. Bechhofer, R. Volz, and P. Lord, "Cooking the semantic web with the OWL API," in *The Semantic Web—ISWC 2003*, vol. 2870 of *Lecture Notes in Computer Science*, pp. 659–675, Springer, Berlin, Germany, 2003.

[40] (Questionnaire) http://www.sojump.com/jq/3999847.aspx.

[41] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the14th International Conference on Ubiquitous Computing (UbiComp '12)*, pp. 501–510, ACM, Pittsburgh, Pa, USA, September 2012.