Crowd Computing Framework for Geoinformation Tasks

Alexander Smirnov and Andrew Ponomarev

Abstract In the paper a general purpose crowd computing framework architecture is discussed. The proposed framework can be used to compose crowd computing workflows of different complexity. Its prominent features include ontological description of crowd members' competencies profiles; automatic assignment of tasks to crowd members; the support of both human and non-human computing units (hybrid crowd); and spatial features of crowd members which make way for employing the proposed framework for a variety of crowdsourced geoinformation tasks.

Keywords Crowd computing · Crowdsourcing · Ontology · Profiling

1 Introduction

Geoinformation technologies include a wide spectrum of tasks, related to collection, processing and presentation of geospatial information. Most of these tasks permit automation and, therefore, are solved mostly in automated way in modern geoinformation systems. However, some problems are hard to deal with solely by automated computer environments. The reasons for that may be of different nature. Probably, the most frequently discussed one is connected with the difficulty that typical information processing techniques have dealing with problems involving heavy usage of common sense knowledge and incomplete definitions. Another

A. Smirnov (✉) · A. Ponomarev
St.Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences, 39, 14th Liniya, St.Petersburg 199178, Russia
e-mail: smir@iias.spb.su

A. Ponomarev
e-mail: ponomarev@iias.spb.su

A. Smirnov · A. Ponomarev
ITMO University, 49, Kronverksky Pr., St.Petersburg 197101, Russia

V. Popovich et al. (eds.), *Information Fusion and Geographic Information Systems (IF&GIS' 2015)*, Lecture Notes in Geoinformation and Cartography,
DOI 10.1007/978-3-319-16667-4_7

reason, less important but still valid, is physical absence of autonomous sensing/ computing devices in places where they are needed to be.

This results in a new kind geoinformation technologies, where classical GIS approaches are amalgamated with social computing practices. Examples of these new kind of technologies are community sense and response systems (e.g., [5]) and crowdsourcing of GIS data. Crowdsourcing becomes more and more widely used source of geospatial information. Examples range from general collaborative mapping (e.g., OpenStreetMap, Google Map Maker, WikiMapia) to thematic projects dedicated to crisis mapping (e.g., [18, 26]), natural resources management [28], e-government [9] and other application areas. Active involvement of non-expert humans into geoinformation processing tasks resulted into emergence of several specific terms, such as *crowd sensing* and *neogeography*.

All these developments are closely related to even bigger research direction aimed on the creation of hybrid human-computer systems, where human does not always consumes the results that are provided by computer devices, but can also be a provider of some information or service that is consumed by other humans or computer devices. From the point of this research direction the classical dichotomy represented by a human that provides some aims and inputs and a machine that performs routine computations to achieve these goals is transformed into a more general perspective of a network of interconnected humans and machines, performing different functions that together achieve the predefined goal.

The special kind of services relying heavily on human-specific abilities are currently discussed under a set of names and in several closely related research areas. The most prominent are crowdsourcing, human computations and crowd computing. All these areas have much in common. The interconnection between them is discussed, for example, in [8]. In this paper, crowd computing is understood as a spectrum of methods and technologies to solve problems with the help of undefined and generally large group of people, communicated through the internet (crowd). This practical definition used by the authors matches well with the specific characteristics of crowd computing that were enumerated in [21] after literature analysis.

The contribution of this paper is twofold. First, it presents a general purpose crowd computing framework that can be used for variety of problems. Second, it shows how this framework can be used to employ crowd computing for problems involving the processing of spatially enabled data.

One of the tasks performed by the authors of this work is the decomposition of currently established practice of programming for hybrid computer-human environments to identify the set of primitive operations that can be used to construct any type of the information processing workflow. In some sense, this task is similar to designing a set of machine instructions, a composition of which allows building any computer program (for hardware computer).

This paper is organized as follows. Section 2 presents current developments in the area of crowd computing framework development. Section 3 discusses the specific features of crowd computing and describes the idea of crowd computing patterns, aimed to form tried-and-tested solutions for typical problems of crowd computing systems. Based on the literature review and specific features analysis, in

Sect. 4 essential requirements are presented that drive the design of the proposed framework. Section 5 presents the overall design of the framework. Section 6 describes how this framework can be used for problems, involving processing of spatially enabled data. Results achieved are summarized in the conclusion.

2 Related Work

This section contains the review of existing multipurpose crowd programming frameworks. The literature analysis has shown that currently there are three major approaches to programming crowd effort: MapReduce-based, workflow-based, database-based.

MapReduce [4] is a programming model for processing and generating large datasets. The primary intent of this model is to allow for programming data processing algorithms easily parallelizable to multiple machines of a cluster. Since its creation in 2004 this programming model has received a lot of attention and has become very popular in the world of distributed data processing.

There were proposed several crowd computing frameworks [1, 10], somehow based on (or inspired by) MapReduce model.

The Jabberwocky programming environment [1] is a set of technologies, including a human and machine resource management system, a parallel programming framework for human and machine computation and a high-level programming language. The programming framework is an adaptation of MapReduce, in which the execution of Map and Reduce functions may require some human actions and the whole process is suspended until these actions are performed. The problem-setter can also impose constraints on characteristics of people who will be asked to perform a task.

CrowdForge [10] is a general-purpose framework for accomplishing complex and interdependent tasks using microtask markets. As a computation model, authors of Crowdforge also adopted MapReduce model augmented by a Partition stage, which purpose is to refine the problem itself and its decomposition into subtasks during the solution process. On the Map stage the problem is decomposed into subtasks which are executed, and on the Reduce stage all the results are merged in a problem-specific way.

Another approach for crowd computing originated in workflow modeling/management systems. The analogy between workflow modeling/management and crowd computing is also quite noticeable: both kinds of systems deal with some process, which can include humans as executive elements. There are several techniques in "pure" business process workflow modeling where human operations are explicitly modeled (BPEL4People, WS-HumanTask). These techniques were generalized and elaborated to achieve a crowd computing programming model.

An example of workflow-based approach to crowd computing is presented, for example, in [11]. The authors propose a three-layered model to crowd computing process. The topmost process/program layer, showing the process context of particular crowd computing task. Crowdsourcing tactic layer, where crowd computing specific operations (such as task parallelizing, quality control) are described. And, the lower-most crowdsourcing operations layer where API calls to some crowdsourcing marketplace are programmed. As a formal scheme for the top two layers the authors use a Business Process Model and Notation 2.0 (BPMN 2.0), proposing an extension to it indicating that some part of the process must be crowdsourced.

In CrowdLang [19, 20] crowd computations are also described in a graphical workflow-based way. The authors propose a set of typical workflow patterns and a graphical notation to specify the computation process. This approach is verified by creating several crowd computing applications for translation from German to English. Another idea that plays an important role in that paper is the focus on reusable crowd programming patterns (represented in the form of workflows).

In papers [6, 16] a database metaphor is developed in the context of crowd programming.

Markus et al. proposed Quirk [16] system and Quirk UDF language. The data model of Quirk is close to relational model supplemented with user defined functions (UDF) for posting tasks and resolving the situation of multiple answers for the same question that can be produced by different crowd members. A whole "program" for crowd in Quirk is represented by a declarative statement very similar to an SQL statement.

Franklin et al. [6] designed a crowd computing system (CrowdDB) in a way inspired by RDBMS. They proposed CrowdSQL, an SQL extension that supports crowd computing. Crowd computing in CrowdSQL is supported both by data definition language (DDL) and data manipulation language (DML) syntaxes. In CrowdSQL DDL, it is possible to define a column or a whole table as crowdsourced, while in CrowdSQL DML a special value similar to SQL NULL value is proposed, meaning that the value should be crowdsourced when it is first used. Beside CrowdSQL language, the authors proposed a user interface generation facility and special query processing techniques for queries involving crowdsourced values.

There are also several *sui generis* crowd computing approaches, that do not fall into one of the categories above.

One of the first multipurpose frameworks for crowd computing is TurKit [13, 15]. It is based on Javascript language with additional library for posting tasks to Amazon Mechanical Turk platform and receiving answers. TurKit uses a "crash-and-rerun" (the name proposed by its authors) programming and execution model designed to suit to long running processes where local computation is cheap, and remote work is costly, which is the case for crowd computing. TurKit suite also includes a generic GUI for running and managing TurKit scripts.

In Turkomatic [12] one of the design goals is to obviate the need for requesters to plan thoroughly through the task decomposition and workflow design. The idea is to crowdsource this information either, i.e. crowd workers are asked to recursively divide complex tasks into simpler ones until they are appropriately short, then to solve them. The requester can also participate and manage the decomposition process and the authors have shown by a case study that requesters involvement usually results in more robust workflows and sane decompositions. Experiments with this approach applied to unstructured tasks (writing an article, vacations planning, composing of simple Java programs) showed that it is effective with respect to certain kinds of tasks.

AutoMan [2] is a domain-specific language and runtime for crowd computing programming. With the use of AutoMan, all the details of crowdsourcing are hidden from the programmer. The AutoMan runtime manages interfacing with the crowdsourcing platform, schedules and determines budgets (both cost and time), and automatically ensures the desired confidence level of the final result. The distinctive feature of this language and runtime is its approach to quality control, based not on the worker features, but on a statistical processing of the results, received from different workers—the system checks whether the results are consistent with the required confidence level and if not, reissues tasks.

3 Human Factors and Crowd Workflow Patterns

One of the most important issues that makes crowd programming significantly special compared with conventional programming for machines is presence of human in the information processing loop. Differences between human and machine in this regard are being analysed on philosophical and technological layers for many decades, but in the context of crowd computing these differences were summarized by Bernstein et al. [3] to the following list:

- Motivational diversity. People, unlike computational systems, require appropriate incentives.
- Cognitive diversity. Characteristics of computer systems—memory, speed, input/output throughput—vary in rather limited range. People, by contrast, vary across many dimensions this implies that we must match tasks to humans based on some expected human characteristics.
- Error diversity. People, unlike computers, are prone to make errors of different nature.

Each of the listed items represents not a particular problem of crowd computing system development, but rather a fundamental issue that results in a bunch of design obstacles and decisions to overcome those obstacles. For example, [2] identifies following challenges for human-based computation:

- Determination of pay and time for tasks. Employers must decide in advance the time allotted to a task and the payment for successful completion.
- Scheduling complexities. Employers must manage the trade-off between latency (humans are relatively slow) and cost (more workers means more money).
- Low quality responses. Human-based computations always need to be checked: worker skills and accuracy vary widely, and they have a financial incentive to minimize their effort. Manual checking does not scale, and simple majority voting is insufficient since workers might agree by random chance.

It is easy to see, that this challenges (addressed in the AutoMan system [2]) are the results of fundamental differences, namely, motivational and error diversity. By the way, cognitive diversity is not addressed in the design of AutoMan.

The first point, taken by the authors of this paper as the basis of our research in the area of crowd computing, is that every crowd computing framework should account for each of these fundamental differences. Moreover, these fundamental differences together provide a basis to describe crowd computing frameworks pointing out the methods and techniques employed to overcome each of these differences. Later, in the respective section the authors will show how the proposed framework addresses all of these differences.

In the rest of this section, the concept of crowd computing patterns is introduced and justified as one of the important concepts in the domain, aimed on dealing with the differences highlighted above.

Like in other branches of computer science and artificial intelligence [7, 24], crowd computing pattern represents a reusable solution to a commonly occurring problem. In crowd computing context this term was first used (to the best of authors' knowledge) by [27] to name various techniques for dealing with unreliability of human responses. Many of these techniques were analyzed and used before that (e.g., [14]), but in [27] there was an attempt performed to describe them as reusable patterns. Later, the concept of crowd computing patterns were investigated in [20] resulting in their own set of reusable workflow constructs. It was also paid some attention in [11], but under the name of "templates" and, moreover, it was stressed there, that crowd computing platform should support reusable process skeletons. The reasons why the idea of crowd computing patterns seems fruitful is twofold. First, these patterns provide tested and effective solutions to error management in human responses. As it was mentioned, error management in this kind of systems is a complex issue, and it does not have an all-fits-one solution: for different kinds of problems different techniques turn out to be most effective. Patterns in this regard help structuring research space of error management and form a body of knowledge about what pattern is favorable in which situations. With this information at hand, a programmer can pick a readily available pattern that shows good results for the concrete problem, or, there is even an opportunity for crowd programming automation. Second, the analysis of this patterns and their building blocks can help to determine the—following an analogy between hardware and crowd computers—instruction set of a crowd computer. A minimal and sufficient set of operations to form any program for a crowd computer.

In [27] three basic patterns were identified:

Divide-and-Conquer. A complex task can be too large for an individual crowd member and, therefore, should require contribution of multiple crowd members. Divide-and-conquer pattern means decomposition of a complex task to several subtasks assigned to different crowd members and followed by composition of subtask solutions received from crowd members into the solution of the initial task.

Redundancy-based quality control. This pattern directly deals with error management in crowd computing. To alleviate the impact of erroneous answers provided by an individual crowd member, one task is assigned to several members and then some quality control mechanism is applied to the answers received to select the most likely correct one. There are several quality control mechanisms developed: averaging, simple voting, weighted voting to name a few. The pattern itself doesn't answer to the question how much redundancy is enough for particular task or what quality control is the best for a certain application, it just forms a scheme of a typical workflow.

Iterative improvement. In this design pattern one task is also assigned to several crowd members, but they work on this task in a sequential manner, and each member is able to see the solution provided by the previous worker. Iterative improvement pattern was successfully applied, for example, for text transcription (e.g., [14]).

An application (for one problem) may be composed of several patterns, for example, the whole problem can be decomposed into several subtask, and each subtask can be distributed to several crowd members with majority voting as the employed quality control mechanism.

4 Requirements

The literature analysis lets to identify a set of requirements for crowd computing framework. This section discusses most important of these requirements, as it is seen by the authors. Hereafter in this section, crowd computing framework will be referred to as "framework" for simplicity.

Framework should provide support of various incentives. The most widely employed in crowd computing incentive schemas are money and reputation, however there are more rare and exotic ones [22].

Framework should provide explicit treatment of cognitive diversity in and between human actors (crowd members). This requirement is a response to one of the fundamental differences between purely machine programming and programming of human-machine systems. In practice, it means that there should be a possibility to describe a crowd member with a set of characteristics corresponding to performance of this member in some type of tasks. The word "possibility" in clarification above should be stressed, because it is an application-specific requirement, and in some applications cognitive diversity can be neglected, but anyway, a multipurpose framework should provide support of it.

Framework should provide abstractions for complex coordination patterns such as quality control or group decision procedures. The advantages of pattern-oriented crowd programming were already enumerated in the respective section of the paper. It is also worth noting, that the list of abstractions provided by the framework should be extensible; in this regard, this requirement is connected to the requirement of providing an abstraction mechanism.

Framework should provide adaptive workflows. Several researchers (e.g., [2, 10, 27]) argue that entire specification of crowd computing in advance is not always the best option. The work specification itself might be adjusted during the time of execution either by human coordinators or by some algorithm and it sometimes leads to better results. This argument is supported by the evidence, so the proposed framework should also support editable/transformable on the fly workflows. However, workflow adaptation requires not only the possibility to change workflow specification in run-time, but also the presence of some entity in the system that implements this adaptation based on the calculation process [27]. Perceived as a part of crowd computing framework, this entity imparts elements of self-organization to it.

Framework should maintain user identities, rich user profiles, and social relationships. All this user information should allow problem setter to select crowd members that should address a task.

Framework should provide an abstraction mechanism, so that users with different skills and with different programming background could organize crowd computing workflow. By no means this prefers simplistic solutions to more advanced, it only requires that there should be several ways to organize workflow ranging from coarse construction from predefined blocks, to fine programming of particular quality control procedures.

Framework should support flexible scheduling. Scheduling in crowd computing is related to the search of balance between time required to solve a problem, dealing with substantial human latency, and cost of using more actors. As it is pinpointed in [2], scheduling in programming systems involving humans represents a very important and difficult task, so the proposed framework should be flexible enough to adopt different scheduling policies. It would allow to experiment with different scheduling models pertaining other components of the systems and code developed for it.

There should be a possibility to add software services to the crowd, resulting in a hybrid crowd. Hybrid crowd [25] is a relatively new research area, which is also developing under different names (e.g., human-machine cloud [17, 23]). The idea behind hybrid crowd reveals conceptual similarity between cloud services and crowd computing, both of which are based on a kind of resource virtualization and pay-per-use basis. Convergence of this two technologies leads to creation of new generation computing environments constituted from humans and machines.

Framework should provide logging and offline analysis facilities. One of the specific features of crowd computing system employing monetary-based incentivization is that execution of some operations cost money. Therefore, it is rather straightforward to try to save as many intermediate results as possible to avoid rerunning expensive operations. On the other hand, the extensive logging of all the intermediate results can provide a basis for applying alternative methods of analysis and to compare different processing methods of one collected dataset.

Framework should be able to use existing crowdsourcing platforms (e.g., Amazon Mechanical Turk), as there these platforms provide not only low-level tools to post task and receive answers, but—the most valuable—these platforms have huge database of users, i.e. provide the access to crowd itself.

5 Framework Design

To meet the requirements identified during the analysis phase, the authors propose a crowd computing framework. Conceptually, the proposed framework consists of three layers:

- The upper layer is *self-organization layer*, which contains components, methods and algorithms for adjusting workflows based on execution process.
- The middle layer (*workflow layer*) performs coordination of crowd computing process based on various patterns of human information workflow. These patterns were identified as a result of literature review: task splitting, different techniques of matching results provided by crowd members, etc.
- The bottom layer (*crowd layer*) contains primitive operations of crowd computer, such as addressing a particular task to a particular crowd member, posting a task into a common task pool, receiving answer from a crowd member.

The two latter layers can be used to easily build various crowd programming workflows, the self-organization layer is in some sense optional, but facilities of this layer can be used to adjust predefined workflow design of a particular application.

Beside three enumerated layers of programmable process, the proposed framework defines metamodels for crowd member and tasks description. These metamodels have very general nature and are equivalent to OWL ontology metamodel.

Runtime infrastructure, provided by the framework, is shown in Fig. 1. Central component of this infrastructure is the *program interpreter* that takes declarative workflow specification, posts human task requests according to this workflow specification and processes the results.

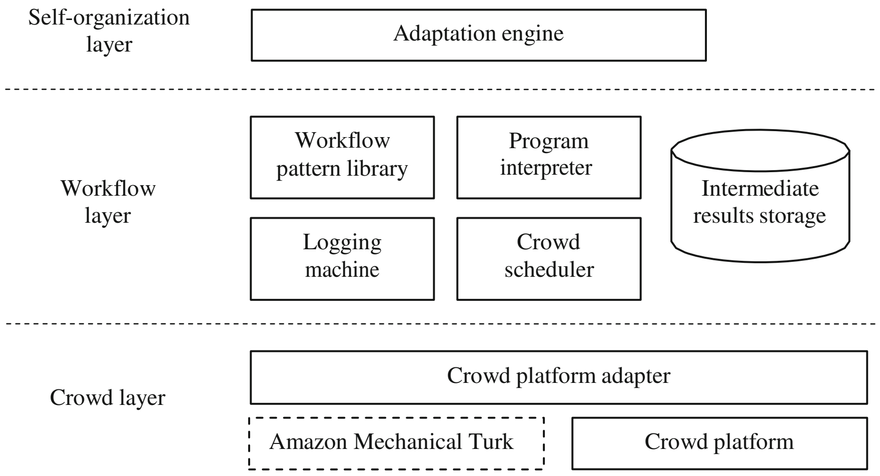

Fig. 1 General view of the framework

Human task requests are then processed by the *crowd scheduler*, which is responsible for assigning tasks to crowd members. Usually, this assignment depends on task attributes, underlying crowd platform features, crowd member declared skills and actual performance. When crowd scheduler finds appropriate members for task requests it posts human tasks to the *crowd platform*.

Crowd platform is the component of the framework responsible to actual communication with crowd members, i.e. sending tasks and receiving answers. One of the goals of this framework design was to adapt to existing crowdsourcing platforms (e.g., Amazon Mechanical Turk), so there is an adapter layer that implements crowd platform interface from one side and uses existing platforms' application programming interfaces from the other side.

Results received from crowd members are documented by the *logging machine,* and then stored in the *intermediate results storage* that holds the global state of all the crowdsourcing process.

The *adaptation engine* analyses the process of computation and transforms running program in order to optimize performance.

The framework is oriented on "top-to-bottom" rewarding calculation for abstract "cost" rewards, excluding other types of incentivization. Initially, a task is assigned some cost limit that the problem-setter can spend on this task. Crowd computing algorithm starts from the whole task and assigns it one unit of resource (corresponding to fund limit). On each task decomposition, resource is divided between subtasks by this algorithm in an application specific way, so that each particular human task is assigned certain cost based on resource value that was obtained after all separations. However, the proposed algorithm fails in situations of dynamic workflows, so there is some room for improvement.

The framework also includes *client module* that is able to communicate with crowd platform. This client module is deployed on crowd member's device and is used to deliver human task to people's devices and collect output. It should be stressed, that client module is able to communicate with original crowd platform only, in case some other crowd platforms are used (e.g. Amazon Mechanical Turk), tasks are assigned by client interfaces of the respective platforms.

Following distinction should be made. The proposed framework presents an infrastructure and generic components for crowd computing applications. An application is created for some particular purpose (collecting the data about traffic jams, prices in supermarkets etc.) with the help of the presented framework's infrastructure. On the other hand, application is to be deployed in a physical machine that also has to have some components (runtime) of the proposed framework, that enable the process of application execution.

An application developed in terms of the proposed framework is defined by the following set of components:

- user model, containing a set of application-specific skills and competencies, represented as an OWL ontology;
- task model, also represented as an OWL ontology;

- declarative code that defines crowd computing algorithm, including user-task matching and reward distribution;
- (optional) quality measures associated with different parts of the algorithm accompanied with signals that respective parts of the algorithm are subject to automatic improvement according to specified quality measure.

User model is defined for a particular application, so in the proposed approach there is no concept of an overall user model with all his/her characteristics. Instead, there is a set of application specific profiles expressed with a help of OWL ontologies. There is an interesting possibility to (partially) fill user model for a new application, based on user model from the applications the user already took part, it can potentially be done in the phase of application deployment and crowd forming, but this is a subject to further research.

Task model describes task attributes and task structure, if it is relevant. In geospatial context, task model usually contains attributes holding physical location of the place task refers to.

The whole computation workflow is defined in a declarative way, using attributes defined by the task and user ontologies.

The framework already contains a number of typical workflow patterns that can be easily reused either in their default form, or specialized in a way, supported by the pattern. For example, pattern *Divide-And-Conquer* should be specialized by providing specific procedures for task decomposition and subtask answers composition.

6 Examples of Geoinformation Tasks

The proposed framework can be used for geospatial information processing. In this section, it is shown how to build a crowd sensing application with the help of this framework.

The application described in this section used mostly for the demonstration purpose. Its main feature is to address a free-form request to people that are located in specific spatial area. An application like that can be used in variety of scenarios, for example, to query for a price (or availability) of some product in a shop, that doesn't expose its product listings to the internet, to find out traffic situation, etc. For the sake of simplicity, the expected answer should be in the form of one real number.

As it was discussed in Sect. 5, an application for the proposed framework consists of a user model, a task model and a workflow code (the forth component is optional, and is not covered in this example). The user model includes all the user features relevant for the application. In this particular case, no special competency modeling should be done, the only user property that is relevant is the user's ability to understand written language of the free-form question. Therefore, the user model in this case consists of the set of languages, which the user is able to understand, and the user's geographical coordinates.

The task model for this particular application is also simplistic, it is represented by three attributes—the text of the question, its language and the spatial area the request refers to.

To organize the application workflow it is convenient to use the *redundancy-based quality control* pattern. This pattern requires specialization by the redundancy number and the procedure of answers reconciliation. For example, it is possible to set the redundancy number to three and the reconciliation procedure as finding a mean value of answers, provided in redundancy branches of execution (remember, that answers for this question should be real values). Application workflow also specifies condition of crowd member selection for tasks, which is crowd member coordinates must be inside spatial area the question refers to and crowd member's list of languages must contain the language of the question.

Crowd members must join to this application to avail themselves as request answerers. When the problem setter wants to send a request, he/she provides input to the application, that is request text, its language, and reward. All this input values form a task instance. This instance is then processed by program interpreter and transformed into three human tasks. The task is sent through the crowd platform to crowd members satisfying the selection conditions, the answers are received, averaged according to workflow definition and presented to the problem setter (Fig. 2).

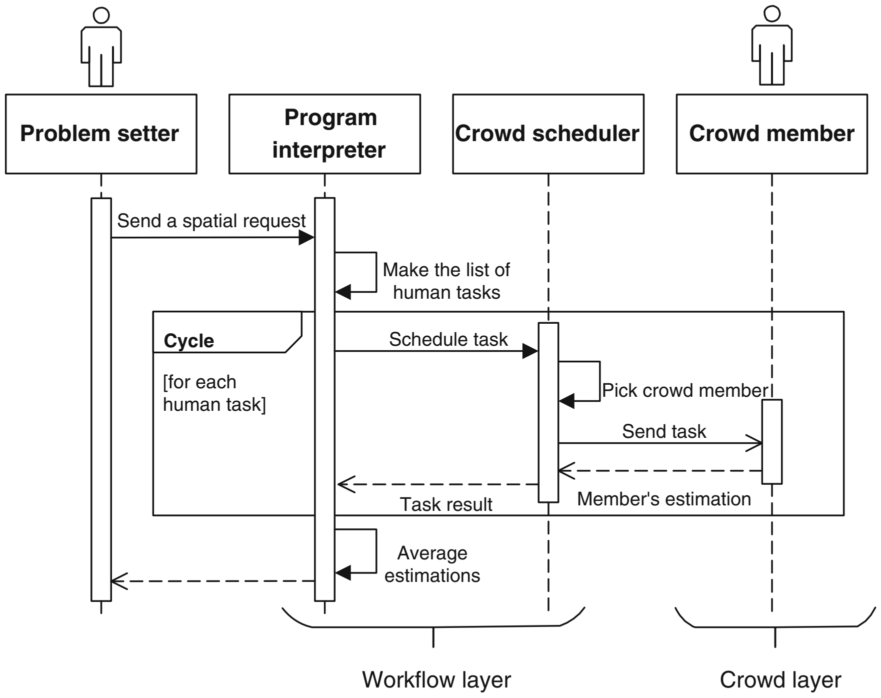

Fig. 2 Sequence diagram for request answering scenario

The provided example illustrates design of a simplified crowd computing application, but the flexibility of the proposed framework allows to define more complex workflows and models.

For instance, the provided example can be easily transformed into a crowd-sourced generation of a thematic map (relating to a free-form request). For this purpose, user model and task model remain unchanged, but the workflow description includes the step of task separation (*Divide-and-conquer* pattern) according to some level of spatial discretization. After that separation a bunch of subtask with the same structure but smaller spatial extent is generated. So, the area, that was provided by the problem-setter, is split into smaller areas. For each of this subtasks a *redundancy-based quality control* pattern is applied as in the original example. Resulting dataset will contain a set of human "measurements" that can be overlaid onto a geographic map forming a thematic layer related to the nature of the question explored by the problem-setter.

Another practical example of crowd computing with geospatial information can be assignment of physical area of competence to each member (e.g., ethnography, economics experts) and addressing questions to experts that have knowledge of particular area.

7 Conclusions

In the paper, a problem of crowd computing process organization was discussed with an application to geospatial information processing. The analysis of current developments revealed the set of specific issues that must be addressed in any crowd computing framework design and the set of crowd workflow patterns that are tried-and-tested ways to effectively address these issues. A set of universal requirements for a crowd programming environment was provided to generalize the work that has been performed so far in the area. Then, the original crowd computing framework design was described.

Although most of the stated requirements are addressed by the proposed crowd computing framework, the model to account for rewarding and incentivization still needs to be clarified and developed, especially for the case, when problem-setter and underlying crowd platform are using different rewarding mechanisms.

Another direction for further work is experimentation with different approaches and algorithms of self-organization and self-adjustment of the geoinformation crowd computing workflow.

Acknowledgments The research was supported partly by projects funded by grants # 13-07-00271, # 14-07-00345 of the Russian Foundation for Basic Research, and by Government of Russian Federation, Grant 074-U01.

References

1. Ahmad S, Battle A, Malkani Z, Kamvar S (2011) The jabberwocky programming environment for structured social computing. In: Proceedings 24th annual ACM symposium user interface software and technology, UIST ’11 (ACM, New York, NY, USA, 2011), pp 53–64
2. Barowy DW, Curtsinger C, Berger ED, McGregor A (2012) AutoMan: a platform for integrating human-based and digital computation. ACM Sigplan Not 47(10):639–654
3. Bernstein A, Klein M, Malone TW (2012) Programming the global brain. Commun ACM 55 (5):41–43
4. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: 6th symposium on operation system design and implementation, San Francisco, CA, 2004
5. Faulkner M et al (2014) Community sense and response systems: your phone as quake detector. Commun ACM 57(7):66–75
6. Franklin M, Kossmann D, Kraska T, et al (2011) CrowdDB: answering queries with crowdsourcing. In: Proceedings of ACM SIGMOD conference
7. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object- oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston
8. Gomes C, Schneider D, Moraes K, de Souza J (2012) Crowdsourcing for music: survey and taxonomy. In: Proceedings of 2012 IEEE international conference on systems, man, and cybernetics, Seoul, pp 832–839, 14–17 Oct 2012
9. Haklay ME, Antoniou V, Basiouka S, Soden R, Mooney P (2014) Crowdsourced geographic information use in government. Global Facility for Disaster Reduction and Recovery (GFDRR), World Bank, London
10. Kittur A, Smus B, Khamkar S, Kraut RE (2011) Crowdforge: crowdsourcing complex work, In: Proceedings 24th annual ACM symposium user interface software and technology, UIST’11 (ACM, New York, NY, USA, 2011), pp 43–52
11. Kucherbaev P, Tranquillini S, Daniel F, Casati F, Marchese M, Brambilla M, Fraternali P (2013) Business processes for the crowd computer. In: Business process management workshops, pp 256–267
12. Kulkarni AP, Can M, Hartmann B (2011) Turkomatic: automatic recursive task and workflow design for mechanical turk. In: CHI ‘11 extended abstracts on human factors in computing systems (CHI EA ‘11). ACM, New York, NY, USA, pp 2053–2058
13. Little G, Chilton LB, Goldman M, Miller RC (2010a) TurKit: human computation algorithms on mechanical turk. In: Proceedings of UIST (2010), ACM, New York, pp 57–66
14. Little G, Chilton LB, Goldman M, Miller RC (2010b) Exploring iterative and parallel human computation processes. In: Proceedings of the ACM SIGKDD workshop on human computation (HCOMP ‘10). ACM, New York, NY, USA, pp 68–76
15. Little G, Chilton LB, Goldman M, Miller RC (2009) TurKit: tools for iterative tasks on mechanical turk. In: Proceedings ACM SIGKDD workshop on human computation, HCOMP’09 (ACM, New York, NY, USA, 2009), pp 29–30
16. Marcus A, Wu E, Karger DR, Madden SR, Miller RC (2011) Crowdsourced databases: query processing with people. CIDR ‘11
17. Mavridis N, Bourlai T and Ognibene D (2012) The human-robot cloud: situated collective intelligence on demand. In: 2012 IEEE international conference on cyber technology in automation, control, and intelligent systems (CYBER), pp 360–365
18. Meier P (2012) How crisis mapping saved lives in Haiti. http://voices.nationalgeographic.com/2012/07/02/crisis-mapping-haiti/. Accessed 28 Nov 2014
19. Minder P, Bernstein A (2012a) How to translate a book within an hour: towards general purpose programmable human computers with CrowdLang, Proceedings of the 3rd Annual ACM Web Science Conference, Evanston, Illinois, pp 209–212, 22–24 June 2012
20. Minder P, Bernstein A (2012b) CrowdLang: a programming language for the systematic exploration of human computation systems. In: Proceedings of the 4th international conference on social informatics, Lausanne, Switzerland, pp 124–137, 05–07 Dec 2012

21. Parshotam K (2013) Crowd computing: a literature review and definition. In: Machanick P, Tsietsi M (eds) Proceedings of the south african institute for computer scientists and information technologists conference (SAICSIT '13), ACM, New York, NY, USA, 2013, pp 121–130
22. Scekic O, Truong H-L, Dustdar S (2013) Incentives and rewarding in social computing. Commun ACM, 56(6):72–82
23. Sengupta B, Jain A, Bhattacharya K, Truong H-L, Dustdar S (2013) Collective problem solving using social compute units. Int J Coop Inf Syst 22(4) (World Scientific Publishing)
24. Smirnov A, Levashova T, Shilov N (2015) Patterns for context-based knowledge fusion in decision support systems. Inf Fusion 21:114–129
25. Smirnov A, Ponomarev A, Shilov N (2014) Hybrid crowd-based decision support in business processes: the approach and reference model. Procedia Technol 16:376–385
26. Ushahidi (2014) http://www.ushahidi.com/. Accessed 28 Nov 2014
27. Zhang H (2012) Computational environment design. Ph.D thesis, Harvard University, Graduate School of Arts and Sciences, Cambridge, Massachusetts
28. Zhang Y, McBroom M, Grogan J, Blackwell PR (2013) Crowdsourcing with ArcGIS online for natural resources management. http://www.faculty.sfasu.edu/zhangy2/download/crowdsourcing.pdf. Accessed 28 Nov 2014