




# Human-Computer Systems for Decision Support: From Cloud to Self-organizing Environments

Alexander Smirnov, Nikolay Shilov, and Andrew Ponomarev<sup>(✉)</sup> 

SPIIRAS, 14th Line 39, 199178 St. Petersburg, Russian Federation  
{smir,nick,ponomarev}@iias.spb.su

**Abstract.** The paper describes conceptual and technological principles of the human-computer cloud, that allows to deploy and run human-based applications. It also presents two ways to build decision support services on top of the proposed cloud environment for problems where workflows are not (or cannot be) defined in advance. The first extension is represented by a decision support service leveraging task ontology to build the missing workflow, the second utilizes the idea of human-machine collective intelligence environment, where the workflow is defined in the process of a (sometimes, guided) collaboration of the participants.

**Keywords:** Human-computer cloud · Human-in-the-Loop · Crowdsourcing · Crowd computing · Human factors

## 1 Introduction

The widespread of information and communication technologies that allow people to access global networks from almost anywhere in the world brings joint and collective initiatives to a new level, which leads to an upsurge in crowd computing and crowdsourcing.

The applicability of systems that rely on human participation (including crowd-based ones) is limited by the fact that they usually require large numbers of contributors, while collecting the required number may require significant effort and time. This problem is partially alleviated by existing crowdsourcing platforms (like Amazon Mechanical Turk, Yandex.Toloka etc.) accumulating the “online workforce” and providing tools for requesters to post tasks and an interface for workers to accomplish these tasks. Existing platforms, however, bear two main disadvantages: a) most of them implement only ‘pull’ mode in distributing tasks, therefore not providing any guarantees to the requester that his/her tasks will be accomplished, b) they are usually designed for simple activities (like image/audio annotation). This paper presents a unified resource management environment, that could serve as a basis on which any human-based application could be deployed similar to the way cloud computing is used nowadays to decouple computing resource management issues from application software.

This paper is an extended and revised version of [1]. In [1] a human-computer cloud (HCC) architecture was described. The proposed HCC includes: 1) an application platform, allowing to deploy and run in the cloud human-based applications (HBA), and 2) an ontology-based decision support service providing task decomposition mechanism in order to automatically build an execution plan for tasks in an *ad hoc* way (for decision support tasks, algorithms for which are not described in advance). However, historically attempts to build human-computer (crowd) information processing systems for complex tasks came to conclusion that no workflow (or, predefined coordination program) is able to account for all potential complications that might arise when dealing with complex tasks [2]. Therefore, any collaborative environment for dealing with complex tasks must support on the fly adaptation of the workflow and (to some extent) leverage self-organization potential of human collectives. The solution proposed earlier in [1] (the ontology-based DSS) only partly satisfies this requirement, as it relies on a task ontology where all possible tasks have to be described (but not algorithms for them). In this paper, we extend the earlier proposed approach by describing a concept of collective intelligence environment built on top of the HCC and supporting self-organization.

Together, the three proposed solutions allow to build human-computer decision support systems for all the spectrum of problems (on a complexity scale). For the most simple ones, where mere programming is enough, PaaS with HBA support should be used, for more complex ones, but studied and structured enough to build a task ontology, the ontology-based decision support service (*René*) is suitable, and finally, the most complex and underdeveloped ones, should be dealt with the help of the self-organization environment build on top of the cloud.

The rest of the paper is structured as follows. Section 2 briefly describes other developments aimed on building human-computer cloud environments and research on crowd support for complex tasks. Section 3 describes the organization of the platform layer of the proposed HCC with a focus on the digital contract concept. Section 4 describes ontology-based decision support service. Section 5 describes the concept of self-organization environment for human-machine collective intelligence. Sections 6 and 7 describe cloud platform implementation and evaluation respectively.

## 2 Related Work

### 2.1 Cloud with Humans

Generally, resources managed by cloud environments are hardware (CPU, storage) and software (cloud applications, platforms). There are, however, a number of attempts to extend the principles of cloud computing (first of all, on-demand elastic resource provisioning) to a wider spectrum of resource types. These attempts can be classified into two groups: 1) cloud sensing and actuation environments, and 2) cloud-managed human resource environments.

A very representative (and one of the earliest) examples of human-based cloud sensing and actuation environment was presented in papers [3] and [4]. Sensing resource is regarded there as a service that can be allocated and used a unified way

independently of the application that needs access to the resource. The cloud system (called MCSaaS – Mobile CrowdSensing as a Service) provides an interface allowing any smartphone user to become a part of a cloud and allow to use his/her smartphone sensors for the benefit of cloud consumers.

The approach proposed in ClouT (Cloud+IoT) project [5] is aimed on providing enhanced solutions for smart cities by using cloud computing in the IoT domain. Both ClouT and MCSaaS approaches are highly relevant to the cloud environment presented in this paper. However, these solutions are focused mostly on sensing and the role played by human in these systems is very limited: human can provide an access to his/her smartphone and make some operations (i.e. point camera lens to some object and make a picture) requested by the application working on top of the infrastructure layer.

Solutions of the second group, earlier referred to as cloud-managed human resource environments, implement another perspective on managing member's skills and competencies in a standardized flexible way (e.g. [6, 7]). In these environments and systems human is regarded as a specific resource that can be allocated from a pool for performing some tasks (not necessary sensing). For example, in [6] the cloud consisting of human-based services and software-based services is considered. On the infrastructure layer, they define a human-computing unit, which is a resource capable of providing human-based services. Like hardware infrastructure is described in terms of some characteristics (CPU, memory, network bandwidth), human-computing unit in this model is described by the set of skills. The authors do not list the exact skills, leaving it to the application domain.

The human-computer cloud environment described in this paper stands more closely to the cloud-managed human resource environments (like [6]). It extends the existing works by an application platform based on a machine-processable specification of obligations in a form of a digital contract and a decision support service that is deployed on top of all resource-management services and can be used to solve *ad hoc* problems in some domain.

## 2.2 Human-Computer Programs for Complex Tasks

The overwhelming part of research in the field of human-machine computing (crowdsourcing, crowd computing) systems understands human participant as a special type of “computing device” that can process requests of a certain type. During the design time, the whole information processing workflow is build and operations requiring human processing are identified. In the run time the function of a human participant, is reduced to performing a specific task, proposed to him/her by the system, interacting with it in a strictly limited manner [8–11].

Although such a rigid division of roles (designer vs. participant of the system) and strict limitation of the participant's capabilities pay back in a wide range of tasks (mostly, simple ones like annotation, markup, etc.), the creative and organizational abilities of a person in such systems are discarded. It is also shown, that fixed pre-defined workflow is too limiting for complex tasks [2]. First attempts to build crowd systems, where participants could refine the workflow appeared in 2012 [12], but the problem is starting to receive the closest attention of the research community only

nowadays. In particular, in recent years several studies have appeared on the limitations of systems based on the fixed flow of work [2] and proposing the formation of dynamic organizations from members of the crowd community (the so-called flash organizations [13]). We adopt main conclusions of these studies and integrate them with the earlier proposed HCC.

### 3 Human-Computer Cloud: Platform-as-a-Service

This section introduces platform-as-a-service (PaaS) functionality of the HCC, designed to enable development and deployment of human-based applications. The section enumerates main actors, interacting with the platform, major requirements that drive the design, information associated with applications and contributors, allowing to fulfill the requirements. The section finishes with main use cases presenting a general view of the platform.

All the actors of the proposed cloud platform can be divided into three main categories:

*End users (application/service developers)*, who leverage the features of the platform to create and deploy applications requiring human effort.

*Contributors*, who may process requests issued by human-based applications running in the human-computer cloud environment.

*System Administrators and Infrastructure Providers*, who own and maintain the required computing infrastructure.

Primary requirements from the side of *End users* considered in the platform design are following:

- The platform must provide tools to deploy, run, and monitor applications that require human information processing.
- The platform must allow to specify what kind of human information processing is required for an application (as some human-based services, like, e.g., image tagging, require very common skills, while others, like tourism decision support, require at least local expertise in certain location).
- The platform must allow estimating human resources available for the application. This requirement, in particular, is different from conventional cloud infrastructures where resources are considered inexhaustible. Human resources are always limited, especially when it comes to people with some uncommon competencies and knowledge. Besides, the rewarding scheme of the application may be not able to collect the sufficient number of contributors. Therefore, having the information about the resource availability, an application developer is know what capacity is actually available to the application, and based on this information he/she may change the rewarding scheme, set up his/her own SLA (for his/her consumers) etc.

#### 3.1 Application Description

The ultimate goal of the PaaS cloud service model is to streamline the development and deployment of the applications by providing specialized software libraries and tools

that help developers to write code abstracting from many details of resource management. All the resource (and dependency) management operations are performed automatically by PaaS environment according to some description (declarative configuration) provided by the developer of the application being executed. The proposed human-computer cloud environment supports similar approach, however, with inevitable modifications caused by the necessity of working with human resources. To streamline the development of applications that require human actions (human-based applications, HBA), the platform allows both a developer to describe what kind of human resources are required for a particular application, and a contributor to describe what kind of activities he/she can be involved in and what competencies he/she possesses. Existing declarative specification means used in cloud systems allow to specify computing resource requirements and software dependencies of an application. However, they are insufficient for the purpose of human-computer cloud, first of all, because of variety of possible human skills and competencies. While virtual machine can be described with a very limited number of features (e.g. CPU, RAM, I/O capacity), human contributor's skills and abilities are highly multidimensional, they can be described in different levels of detail and be connected to a wide range of particular application areas. Moreover, the same skills can be described in different ways, and, finally, most of the skill descriptions in real world are incomplete (however, there might be a possibility to infer some skills that a human might possess from those that he/she explicitly declared).

Therefore, application deployed in the human-computer cloud environment must contain a descriptor that includes following components (we list all the components, but focus on those, that are relevant to human part):

- configuration parameters (e.g. environment variables controlling the behavior of the compiled code);
- software dependencies of the application (what platform services and/or other applications it relies on, e.g., database service, messaging service, etc.);
- human resource requirements, specifying contributors with what skills and competencies are needed for the application. Along with the software requirements, these requirements are resolved during the service deployment. In contrast to the software requirements, which are usually satisfied, these requirements may be harder to satisfy (resolving these requirements employs ontology matching which may result in some tradeoffs, besides, human resources are usually limited), therefore, the status and details of the requirements resolution are available to the developer and can be browsed via the management console. Human resource requirements specification may describe several types of human resources with different profiles of requirements. For example, an itinerary planning application may require people with significant local expertise as well as people with shallow local expertise but good language skills. These two categories of contributors may be declared as two types of resources;
- digital contract template for each type of human resources. Digital contract defines measurable and machine understandable/checkable specification of contributor's involvement, his/her obligations (e.g., number of requests, reaction time) and rewarding.

For a formal specification of requirements the proposed environment leverages the apparatus of formal ontologies. Specifically, arbitrary ontology concepts may be used to describe skills or knowledge areas. Main benefit of using ontologies is that they allow to discover resources described with related, similar but not exact terms. This is done either by using existing public mappings between ontologies (stating equivalence between concepts of different ontologies), or by ontology inference. We do not fix any particular set of ontologies to describe competencies. It allows tourist applications deployed on the platform to use public cultural, historical, and geographical ontologies, whereas, e.g., applications, that employ human-based information processing in the area of medicine or biology use the ontologies of the respective domain. The only restriction is that these ontologies have to be encoded in OWL 2.

Another important feature of the approach is the concept of digital contract, representing an agreement between contributor and platform about terms of work, quality management principles and rewarding. Terms of the digital contract are essential for estimating the amount of resources available for a service and its capacity (including time perspective of the capacity). The necessity of this digital contract is caused by the fact that human resources are limited. In case of ordinary hardware, the cloud infrastructure provider can buy as many computers as needed, human participation is less controllable due to free will, therefore, attracting and retaining contributors can be a complex task. As a result, the abstraction of inexhaustible resource pool that is exploited in the provider-consumer relationship of current cloud environments turns out to be inadequate for human-computer cloud. A consumer (*end user*) should be informed about the human capacity available for his/her application to make an informed decision about revising digital contracts (for example, making contribution to this application more appealing), or updating their own service level agreements. This creates a competition between consumers for the available resources and finally will create a kind of job market where different digital contract statements will have its own price.

### 3.2 Contributor Description

To support semantic human resource discovery and task allocation, human resources have to be described with ontology terms. Besides, they have to specify their preferences about task types, availability time etc. Therefore, when a contributor joins the cloud platform he/she provides two main types of information, that are very similar to the respective pieces of application descriptor. Namely, the description of competencies and working conditions. The competencies are described in terms of any ontology familiar to the contributor. For the contributors who cannot use ontologies the description is built iteratively via the analysis of contributors' text description followed by ontology-based term disambiguation. In any case, internally, each contributor is described by skills, knowledge and attitude, associated with the concepts of some shared ontology.

Moreover, the contributor's competency description is multi-layered. The first layer is provided by a contributor him-/herself, further layers are added by applications in which the contributor takes part. For this purpose, a human resource management API available for the application code allows to manage application-specific skills and

qualifications, which can also be described in some ontology (application-specific or not). Therefore, despite initial description of competencies may be rather short, during the contributor's participation in various applications running on the platform it becomes richer. This facilitates further human resource discovery.

Working conditions include preferred skills, as well as payment threshold, reactivity and availability limitations. These parameters are also similar to those included in digital contract template in the application descriptor. During application enquiry and application deployment its contract template is matched against contributors' work conditions. Moreover, this matching touches not only contributor's declared work conditions and one application contract (of the deployed application) but also other applications which contracts this contributor has already accepted, controlling overall responsibilities that are taken by the contributor and resolving possible conflicts.

### **3.3 Main Functions of the Platform**

Each category of users (identified earlier as actors) has unique purpose of using the environment. Main functions of the environment are the ones that allow users of each category to accomplish their primary goal.

Main functions exposed to application/service developers are application deployment (which initiates advertisement process to identify human resources available to this application), editing digital contracts, monitoring, and deleting applications. Editing digital contracts is important, for example, when the developer of the application competes for human resources with other application developers by offering higher rewards. Changing the contract produces a new version of the deployed application descriptor and leads to a new wave of advertisements. Application monitoring is a general name for a number of functions like reading usage statistics, logs, that are necessary for application developer and are common to many current PaaS. This also includes monitoring of the available human resources by each requirement type as well its prediction.

Contributors can edit their competence profiles (providing an initial version and updating it), browse application advertisements addressed to them (compatible with his/her competence profile and work conditions) optionally accepting some of them by signing digital contract and attaching to the respective application. Contributors can also execute application-specific tasks and detach from application (possibility of detachment in a particular moment and the effect of it might be affected by the digital contract).

System administrators can monitor the status of the platform (usage of hardware/human resources, communication channels throughput, platform services' health) and tune the platform parameters (e.g., edit ontology mappings used by the platform during identification of compatible resources (contributors) for advertising applications).

### **3.4 Application Deployment**

A newly registered contributor is not automatically available for requests of all human-based applications running on the environment. However, he/she starts to receive the

so-called *advertisements* from applications based on the similarity of a declared competence profile (including additional layers created by applications a contributor participates) and applications' competence requests as well as the correspondence of the declared working conditions and applications' digital contract templates. These *advertisements* describe the intent of the application, required kind of human involvement, rewarding scheme etc. Based on the information contained in the *advertisement*, a contributor can decide to attach to the application, which means that he/she will then receive tasks from this application. In other words, if a registered contributor agrees to contribute to the particular application a digital contract is signed specifying the intensity of task flow, the rewarding and penalty details and quality measurement strategy. In general, there is many-to-many relation between applications and platform contributors, i.e., one contributor may sign digital contracts with several applications.

Upon signing of a digital contract with an application (via platform mechanisms), the contributor starts to receive requests (tasks) from this application. The application issues mostly non-personalized requests, and it is the platform that routes requests to the contributors, ensuring that load of each contributor conforms the terms of the contract. A contributor also can detach from an application (however, the mechanism and terms of this detaching can also be a part of digital contract to ensure the application provider can react to it accordingly).

## 4 Ontology-Based Decision Support Service

The decision support service (named *René*) is an application, running on top of the human-computer cloud infrastructure, leveraging some features of the platform (e.g., resource management and provisioning), and provided to end users according to SaaS model. Users of *René* are decision-makers who pass task specifications to the application. The API of *René* accepts ontology-based structured representation of the task specification. The problem of creating such specification (for example, as a result of text analysis) is out of the scope both of this paper and of *René* functions. The core principle behind *René* is that it builds an on-the-fly network of resources (human and software) capable of performing the specified task.

In order to build the network, *René* decomposes the task into smaller tasks (sub-tasks) using a problem-specific task ontology, where domain tasks and their input and output parameters are described. After performing the decomposition *René* tries to distribute the elementary subtasks among the available resources. The list of available resources is retrieved via an API from underlying layers of the environment, which monitor all the contributor's connections and disconnections and software resource registrations. The resource management service under the hood treats human and software resources differently. Human resources (contributors) describe their competencies with a help of ontology and receive advertisements to join human-based applications if their requirements are compatible to the declared competencies of the user. *René* is basically one of human-based applications and may only distribute subtasks among contributors who agreed to work with it (attached to it by signing a digital contract). Software services are made available to *René* by their developers and



maintainers by placing a special section into the application deployment descriptor. Task assignment is also done with a help of interfaces of resource management layer, aware of the status and load of the resources and terms of their digital contracts.

As contributors may go offline, *René* monitors the availability of the resources (via the underlying resource management) during execution of the subtasks and rebuilds the assignment if some of the resources fail or become unavailable to keep the bigger task (received from the end user) accomplishable.

#### 4.1 Task Decomposition

Task decomposition is the first step to building the resource network. The goal of this process is to build a network of smaller (and simpler) tasks connected by input/output parameters, such that this network is equivalent to the original task given by the user. Therefore, in some sense, the process of task decomposition is actually driven by task composition (particularly, to searching a composition, equivalent to the original task). The proposed approach is based on the fact that there is an ontology of tasks. This task ontology should consist of a set of tasks and subtasks, sets of input and output parameters of task, set of valid values of parameters, as well as the set of restrictions describing the relations between tasks/subtasks and parameters and between parameters and their valid values:

$$O = (T, IP, OP, I, E) \quad (1)$$

where  $T$  is set of tasks and subtasks,  $IP$  – set of input task parameters,  $OP$  – set of output task parameters,  $I$  – set of valid parameter values,  $E$  – restrictions on the parameters of the task and parameter domain.

Unlike existing task ontologies (e.g., [14]) that usually contain relationships between task and their subtasks in explicit form, in the proposed task composition ontology these relationships are implicit. This allows, on the one hand, to specify tasks and subtasks in the same axiomatic form and, on the other hand, to derive task composition structure by reasoning tools. Therefore, the proposed ontology opens the possibility to describe a number of different tasks in the same form and dynamically construct their possible compositions using appropriate criteria.

The task composition ontology is developed in OWL 2. The ontology is expressed by ALC description logic, which is decidable and has PSpace-complete complexity of concept satisfiability and ABox consistency [15] in the case when TBox is acyclic. In addition, SWRL-rules are defined for deriving composition chains. The main concepts of the ontology are “Task” and “Parameter”. The concept “Parameter” is used to describe semantics of a task via its inputs and outputs. The main requirement for TBox definition is that it shouldn’t contain cyclic and multiple definitions, and must contain only concept definitions specified by class equivalence.

Each task should have at least one input and at least one output parameter. The taxonomy of parameters is presented by a number of subclasses of the class “Parameter”. The type of parameters related to their input or output role are defined by appropriate role construct. In the ontology, the appropriate object properties are “hasInputParameter” and “hasOutputParameter”. The domain of the properties is

“Task” and the range – “Parameter”. Thereby the parameter could be input parameter of one task and output parameter of another. The task definition is expressed formally as follows:

$$T \equiv (\exists R.IP_1 \sqcap \exists R.IP_2 \dots \sqcap \exists R.IP_N) \sqcap \sqcap (\exists R.OP_1 \sqcap \exists R.OP_2 \dots \sqcap \exists R.OP_N) \quad (2)$$

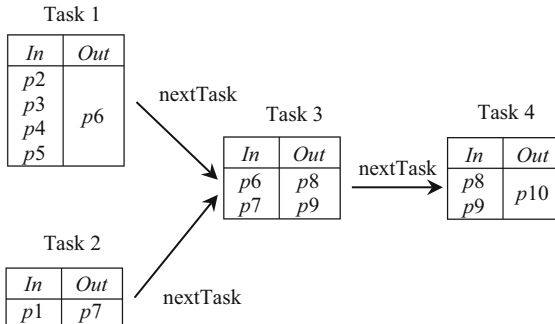
where  $T$  is the task,  $IP_i$  – the input parameter subclass,  $OP_i$  – the input parameter subclass,  $R$  – the appropriate role. In other words, the task is defined solely by its input and output parameters.

Task composition process is based on the fact that in a composition output parameters of one task are input for another. This relationship is utilized (and formalized) by an SWRL rule that infers the order of tasks. The rule specifies input and output parameter match condition in the antecedent and if the antecedent condition is satisfied, infers the relationship “nextTask” between the respective tasks. This relationship means that one task can be done only after another. This relationship is encoded as an object property binding two tasks (both domain and range of the property are “Task” instances). The rule of task composition can be expressed as follows:

$$hasInputParameter(?ta, ?p)hasOutputParameter(?tb, ?p) \rightarrow nextTask(?tb, ?ta) \quad (3)$$

where  $hasInputParameter$ ,  $hasOutputParameter$ ,  $nextTask$  are the mentioned object properties,  $ta$  – the next task,  $tb$  – the previous task,  $p$  – the parameter.

The proposed rule (3) allows to derive all task connections by the object property “nextTask”. The example of task composition is presented in Fig. 1. For example, relationship “nextTask” is inferred between tasks “Task 1” and “Task 3” because parameter  $p_6$  is input for “Task 3” and output for “Task 1”, meaning that “Task 3” can only be executed after “Task 1”.



**Fig. 1.** Task composition structure (adopted from [1]).

The advantages of the described approach is that it allows to simplify task description (in comparison to the approaches where task/subtask relations are explicit) and to derive task compositions dynamically. The shortcomings are the possible deriving complexity and the lack of the support of alternative task compositions.

## 4.2 Subtask Distribution

In cloud computing systems, the number of interchangeable computing resources is usually very high [16, 17]. The decision support service distributes the specialized tasks that require certain competencies, therefore, a) the distribution algorithm has to take into account the competencies, b) not all resources are interchangeable for the assignment and the number of appropriate resources may be lower. Therefore, typical algorithms used in cloud computing cannot be directly applied.

In the areas with similar task characteristics (e.g., distribution of tasks among the robots or agents) the most common approach is instant distribution of tasks (instantaneous task allocation) [18, 19]. This approach involves assigning tasks to resources that currently provide the maximum “benefit” according to the given priorities. It does not take into account that at some point all resources with the required competencies may be occupied. Thus, it is usually supplemented by some heuristics specific to a particular application area.

Let,  $A$  – is a task, which contains several subtasks  $a_i$ :

$$A = \{a_i\}, i \in \{1, \dots, n\} \quad (4)$$

Let,  $O$  – is the vocabulary of competencies:

$$O = \{o_1, o_2, \dots, o_m\} \quad (5)$$

Thus, the matrix of competencies required to accomplish subtasks can be defined as:

$$(ao_{i,j} \in \{0, 1, \dots, 100\}), i \in \{1, \dots, n\}, j \in \{1, \dots, m\} \quad (6)$$

The set of human-computer cloud resources  $R$  is defined as:

$$R = \{r_1, r_2, \dots, r_k\} \quad (7)$$

The set of resource characteristics (speed, cost, etc.)  $C$  is defined as:

$$C = \{c_1, c_2, \dots, c_l\} \quad (8)$$

Thus, each resource  $r_i$  is described by the following pair of competencies and characteristics vectors:

$$r_i = ((ro_{i,1}, \dots, ro_{i,m}), (rc_{i,1}, \dots, rc_{i,l})) \quad (9)$$

where  $i \in \{1, \dots, n\}$ ,  $ro_{i,j} \in \{0, \dots, 100\}$  – is the value of competency  $j$  of the resource  $i$ , and  $rc_{i,j}$  is the value of the characteristic  $j$  of the resource  $i$ .

The solution of the task  $A$  describes the distribution of work among system resources and is defined as:

$$S_A = (s_{ij}), i \in \{1, \dots, n\}, j \in \{1, \dots, k\} \quad (10)$$

where  $s_{ij} = 1$ , if the resource  $j$  is used for solving subtask  $i$ , and  $s_{ij} = 0$  otherwise.

The objective function, which also performs normalization of various characteristics, is defined as follows:

$$\begin{aligned} F(S_A) = f(F_1(s_{1,1}, s_{2,1}, \dots, s_{n,1}), \\ F_2(s_{1,2}, s_{2,2}, \dots, s_{n,2}), \dots, \\ F_k(s_{1,k}, s_{2,k}, \dots, s_{n,k})) \rightarrow \min \end{aligned} \quad (11)$$

Specific formulas for calculating partial assignment efficiency ( $F_i$ ) can use values of resource characteristics (e.g., speed or cost)  $rc_{i,j}$ , as well as competence values of both resources ( $ro_{i,j}$ ) and subtasks ( $ao_{i,j}$ ).

The minimization must be performed with respect to the following constraints. First, each subtask must be assigned to some resource:

$$\forall i = \sum_{j=1}^k s_{ij} \geq 1 \quad (12)$$

Second, assignment can only be done if the competency values of the resource are not less than the required competency values of the subtask:

$$\forall i, j, q : ((s_{ij} = 1) \rightarrow (ro_{j,q} \geq ao_{i,q})) \quad (13)$$

### Instantaneous Distribution of Tasks Algorithm

In general, the specified problem is NP-complete, therefore, it is not possible to solve it by an exhaustive search method in a reasonable time (provided that a real-world problem is solved). Based on the analysis of existing methods it is proposed to use the approach of instantaneous task allocation. The algorithm based on the approach of instantaneous distribution of tasks is defined as follows:

1. Take the first subtask from the existing  $ai$ , and exclude it from the set of subtasks  $A$ ;
2. Select such resource  $j$  from the available resources to satisfy all conditions and  $F(S_A) \rightarrow \min$ , where  $S_A = (s_{1,1} = 0, \dots, s_{1,j} = 1, \dots, s_{1,k} = 0)$ ;

3. If a suitable resource is not found, assume that the problem is unsolvable (the system does not have a resource that meets the required competencies);
4. Repeat steps starting from step 4 until set A is empty (i.e. all tasks are assigned to resources).

### Multi-agent Distribution of Tasks

There are two types of agents that are used to perform multi-agent modeling: the customer agent that is responsible for generating jobs and making the final decision, and the execution agents that represent the resources of the cloud environment and perform on-premises optimization for each resource. In the optimization process, agents form coalitions that change from step to step to improve the values of the objective function.

In the process of negotiations, an agent can play one of the following roles: a coalition member (an agent belonging to the coalition), a coalition leader (an agent negotiating on behalf of the coalition) and an applicant (an agent who can become a member of the coalition).

First, each agent forms a separate coalition (SC, which has the structure of the SA solution), and becomes its leader. Suggestions of agents (tabular representation  $F(s_{1,1}, s_{2,1}, \dots, s_{n,1})$ ) are published on the blackboard (information exchange entity available to all agents). At each stage of the negotiations, the agents analyze the proposals of other agents, and choose those whose proposals can improve the coalition: to solve a larger number of subtasks or the same number of subtasks but with a better value of the objective function ( $F(SC) > F(SC')$ ), where SC is the current coalition, SC' – possible coalition). Coalition leaders make appropriate proposals to agents, and the latter decide whether to stay in the current coalition or move to the proposed one. The transition to the proposed coalition is considered if one of the above conditions is met: the proposed coalition can solve more subtasks than the current one, or the same number of subtasks, but with a better value of the objective function.

The negotiations process is terminated if one of the following conditions is met: a) there are no changes in the composition of coalitions at some stage, b) timeout, and c) the permissible value of the objective function is reached.

## 5 Self-organizing Environment

This section describes the concept of self-organization environment for human-machine collective intelligence, which is built on top of the HCC (e.g., leveraging the resource discovery and communication facilities of the cloud). In this way, the self-organizing environment is another specific application that may be deployed on the HCC.

The proposed environment aims at supporting the process of making complex decisions and/or making decisions in complex problem domains. The complexity of making such decisions generally stems from problem uncertainty in many levels and the lack of relevant data at decision maker's disposal. Hence, while in the upper level the methodology of decision-making stays quite definite (identification of the alternatives, identification of the criteria, evaluation of the alternatives etc.), the exact steps

required to collect all the needed data, analyze it and present to the decision maker may be unclear. That is why decision support requires *ad hoc* planning of the low-level activities and should leverage self-organizing capabilities of the participants of the decision support process. Besides, currently most of the complex decisions are based not only on human intuition or expertise, but also on the problem-relevant data of various types and sources (starting from IoT-generated, to high-level Linked Data), processed in different ways. In other words, decision support is in fact human-machine activity, and the environment just offers a set mechanisms and tools to mitigate this activity.

There are several typical roles in the decision support process. *Decision-makers* are responsible for the analysis of a situation and making a decision. In some cases, where the uncertainty associated with the situation is too high, the decision-maker requires some additional expertise that may be provided by participants of a human-machine collective intelligence environment. Bearing in mind, that using collective expertise is usually rather expensive and can be justified only for important problems, the decision-maker is usually a middle-to-top level manager in terms of typical business hierarchy. After the decision-maker posts the problem to the collective intelligence, he/she may oversee the process of solution and guide it in some way.

*Experts* possess problem-specific knowledge and may contribute into decision support process in several ways. First, they can propose procedures of obtaining relevant judgments, constructing in an *ad hoc* way elements of the whole workflow. This can be done not only in a direct manner, but also indirectly, by posting various incentives for other participants. Second, they can use their expertise by providing data as well as processing it to come to some problem-related conclusions. In general, an expert can be anyone – within or without the organization boundary, the difference is mostly in the incentives important for the particular expert.

*Service providers* design and maintain various software tools, services and datasets that can be used for decision support. Their goal is to receive remuneration for the use of these tools, that is why they are interested in making these services available for other participants of the environment.

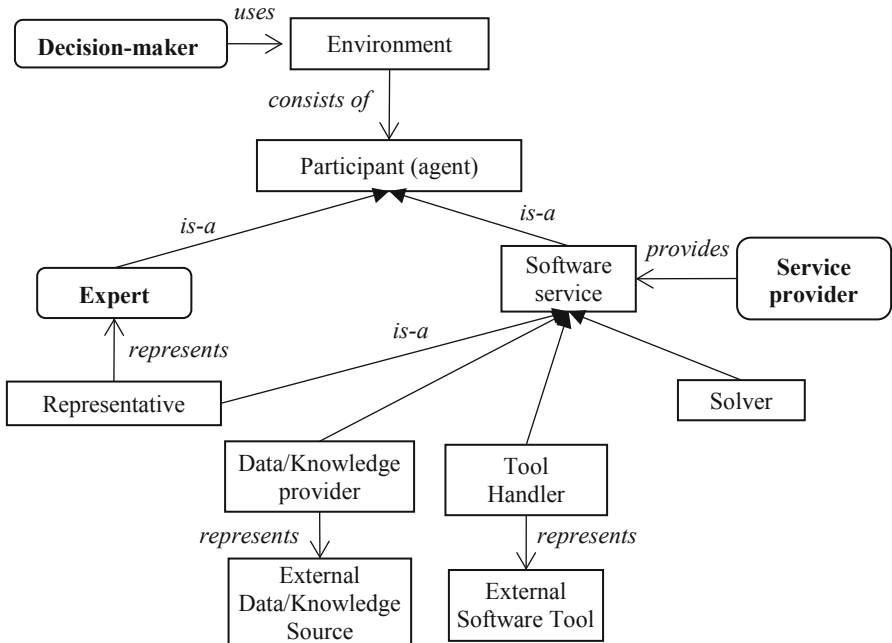
The environment should provide means and mechanisms using which participants of different nature (human and machine) could be able to communicate and decide on the particular steps of decision support process, perform these steps and exchange results, motivated by some external or internal mechanisms, making the whole environment profitable for all parties.

The rest of the section introduces foundational technologies and enablers for the proposed environment.

**Meeting Collective Intelligence and Artificial Intelligence.** Methods of collective intelligence (construed as methods for making people to work together to solve problems) and methods of artificial intelligence are two complementary (in some industries even competing) methods of decision support. Mostly, these approaches are considered as alternative (some tasks due to their nature turn out to be more “convenient” for artificial intelligence methods, and others – for collective), however, the scientists are currently tending to speak about possibility of their joint usage and the potential that human-machine technologies have [20–22].

In the proposed environment artificial and collective intelligence are meeting in the following way. The environment itself provides possibility of communication and coordination of agents while working on solving the problem (collective part). Software services have to “understand” common goal and build their strategy (AI part). Besides, some agents can provide application level AI methods.

There are four types of intelligent software services that take part in the functioning of the environment (Fig. 2):



**Fig. 2.** Main entities of the environment and their relationships.

- Solver. A software code that can transform a task description in some way, enriching it with some derived knowledge.
- Data/knowledge provider. Interface-wise similar to the previous type, however, only provides some problem-specific information.
- Tool handler. A utility agent that manages human access to some software tools (with GUI). In many cases, certain data processing routines required for decision-making can be implemented with some software (or, SaaS). It is not practical to re-implement it in a new way, however, granting an access to such tools might be useful for all the involved parties.
- Representative. Allowing expert to communicate with other services.

**Self-organization Protocols Taking into Account both Human and Machine Agents.** One of the distinguishing features of the proposed approach is to overcome the preprogrammed workflows that rigidly govern interaction of participants during decision support and to allow the participants (human and machine agents) to dynamically decide on the details of the workflow unleashing creative potential of humans. Therefore, agents should be able to coordinate and decide on task distribution, roles etc., in other words a group of agents should be able to self-organize.

The protocols of self-organization in such environment have to respect both machine and human requirements. The latter means that widely used models of bio-inspired self-organization turn out to have less potential to be applied, as they are taken mostly from the analysis of primitive behaviors (e.g., of insects). On the other hand, market (or, economics) based models best of all match the assumed business model (on demand service provisioning). Another possible source are socio-inspired mechanisms and protocols, which are totally natural for people, and there are already some attempts to adapt them for artificial systems [23].

**Interoperability of Agents.** To sustain various coordination processes, as well as information flow during decision-making multilevel interoperability has to be provided inside the collaborative environment. This is especially acute in the case of mixed collectives, consisting of human and machine agents.

To implement any self-organization protocols, the participants of the system have to exchange several types of knowledge:

- Domain knowledge. What object and what relationships between objects are in the problem area.
- Task knowledge. Both goal description, and possible conceptualization of the active decision support task, e.g., mapping some concepts to alternatives, functions to criteria.
- Protocol knowledge. Terms of interaction, incentives, roles etc.

It is proposed to use ontologies as the main means ensuring the interoperability. The key role of the ontology model is in its ability to support semantic interoperability as the information represented by ontology can be interpreted both by humans and machines. Potentially, ontology-based information representation can provide the interoperability for all kinds of possible interactions (human-human, human-machine, machine-human). Taking into account the heterogeneity of the participants of the human-machine collective intelligence systems and the multidimensionality of the decision support activities, it is proposed to use multi-aspect ontologies. The multi-aspect ontologies will avoid the need for standardization of all services of environment through providing one aspect (some viewpoint on the domain) to services of one collective (services of one producer, services that jointly solve a certain task, etc.) for the service collaboration.

**Soft Guidance in Collective Action.** Though the execution process in the proposed environment is self-orchestrated and driven by negotiation protocols, human participants, however, will need intelligent assistance when communicating with other agents in the environment. The role of this assistance is to offer viable organization structures and incentive mechanisms based on current goals. An important aspect during the soft



guidance is mapping actions defined by decision-making methodologies to human-computer collaboration scenarios. It means that the environment (or representative service) uses the existing knowledge on decision making process to offer agents viable collaboration structures.

## 6 Implementation

The implemented research prototype of the cloud environment contains two parts: platform-as-a-service (PaaS) and software-as-a-service (SaaS). The platform provides the developers of applications that require human knowledge and skills with a set of tools for designing, deploying, executing and monitoring such applications. The SaaS model is represented by an intelligent decision support service (referred to as *René*) that organizes (human-computer) resource networks for on-the-fly tasks through ontology-based task decomposition and subtasks distribution among the resources (human participants and software services).

The prototype environment comprises several components: 1) a server-side code that performs all the resource management activities and provides a set of application program interfaces (APIs), 2) a set of command line utilities that run on the computers of appropriate categories of users (platform administrator, developer, administrator of IDSS) and by accessing the API, enable to implement the main scenarios necessary for these users, 3) Web-applications for participants and decision makers. Also, it is possible to implement the interface of the participant for an Android-based mobile device.

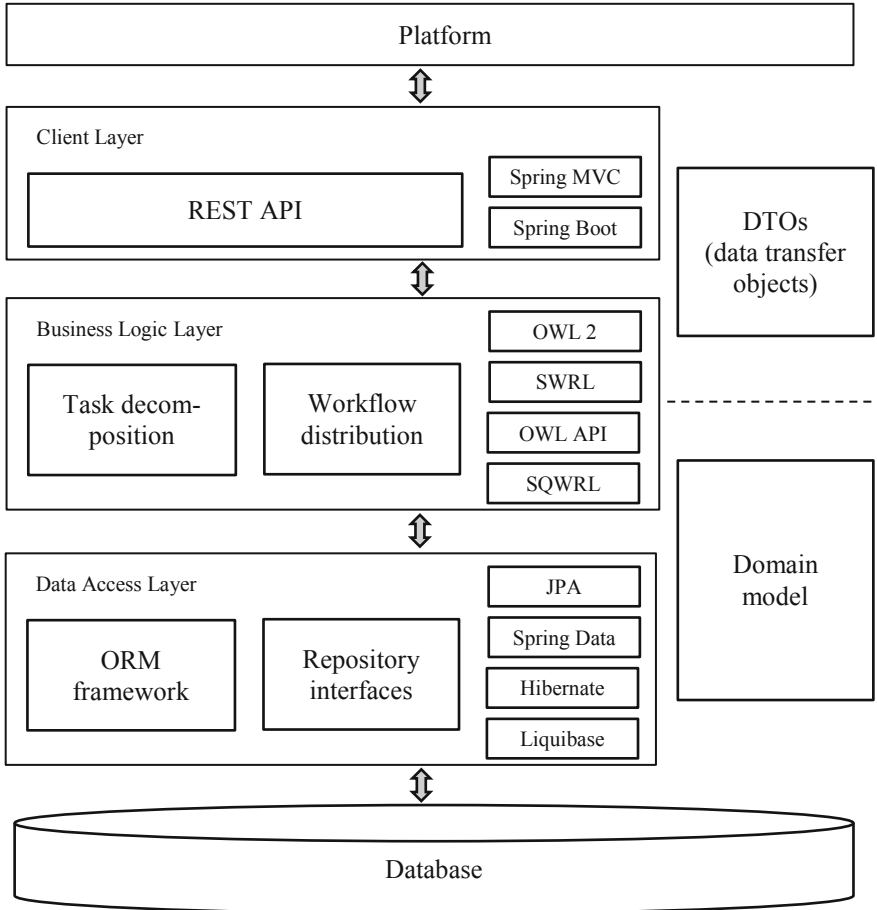
To build and refine a participant competence profile the prototype environment interacts with social networks (ResearchGate, LinkedIn). It also exposes several APIs to the applications deployed on it, providing them basic services (for instance, request human-participants, data warehouses, etc.).

To support the processes of scalable software deployment (which is necessary for the PaaS, but peripheral to the main contributions of our work) the open source platform Flynn<sup>1</sup> is used. The capabilities of the platform has been extended by special functions (registration of participants, an ontological-oriented search for participants, and a mechanism for supporting digital contracts).

The intelligent decision support service (IDSS) is an application deployed in the human-computer cloud environment and using the functions provided by the environment (for instance, to organize interactions with participants). Main functions of the IDSS are: 1) decomposition of the task that the decision maker deals with into subtasks using the task ontology and inference engine that supports OWL ontology language and SWRL-rules (for instance, Pellet, HermiT, etc.); 2) allocation of the subtasks to participants based on coalition games. The IDSS provides REST API to interact with the platform.

---

<sup>1</sup> <http://flynn.com>.



**Fig. 3.** IDSS implementation.

The architecture of IDSS (Fig. 3), in its turn, can be divided into several logical layers:

- The Data Access Layer is a series of DAO abstractions that use the JPA standard for object-relational mapping of data model classes (Domain model) that perform the simplest CRUD operations using ORM Hibernate and implemented using Spring Data.
- The Business Logic Layer of the application is represented by two main services: the task decomposition service and the workflow distribution service. The task decomposition service operates with an ontology, described using the ontology description language OWL 2, which includes rules in SWRL and SQWRL. Knowledge output (task decomposition) is carried out using inference engines (Pellet, HermiT, and others). To extract data from ontology, Jena APIs are used for ontologies recorded using OWL/RDF syntax using the SPARQL query language

and OWL API for other ontology scenarios (changing ontology structure, managing individuals, logical inference). The workflow building service provides support for the coalition game of agents of the human-machine computing platform agents.

- At the Client Layer, REST API services are implemented for interacting with the platform, providing an interface for interacting with the platform and indirect user interaction.

## 7 Evaluation

An experimental evaluation of the research prototype has been carried out. As the functionality of the application of the problem-oriented IDSS built according to the proposed approach is determined by the task ontology (namely, the basic tasks represented in this ontology and their input and output parameters), a task ontology for the e-tourism domain has been developed (specifically, for building tourist itineraries). In the experiments, dynamic task networks (for building tourist itineraries) did actually organize, and their execution resulted in valid lists of itineraries.

The developed software was deployed at the computing nodes of the local network of the research laboratory (an access to the server components from the Internet was also provided). 34 people were registered as participants available for task assignment. An ontology to describe the competences of resources (and the requirements for competences of deployable applications requiring human participation) representing 46 basic competences was used. With the experimental parameters (performance of hardware resources, number of participants, size of the competence ontology), the application deployment time differed from the application deployment time in the Flynn core cloud environment slightly (by 3–7%). The increase in time is inevitable, because during the application deployment process, in addition to creating Docker containers, compiling and launching an application (performed by Flynn), the semantic search of participants and the comparison of digital contracts are carried out. However, in exchange for this slight increase in application deployment time, the applications in the implemented cloud environment receive an opportunity to access human resources. In the future, most of the operations related to the resolution of application dependencies on human resources can be performed in the background, which will save the deployment time at the level of the cloud environment.

For testing the IDSS a task ontology of the electronic tourism domain, represented in the OWL 2 language corresponding to the description logic of ALCR (D) and containing 293 axioms and 40 classes, was used. The scenario for the load testing was to build a network of resources for the task of building a tourist route (the network assumes the fulfillment of 6 subtasks). The time of task decomposition and network construction obtained as a result of averaging over 25 tests is, 1157 ms (994 ms takes the task decomposition, 163 ms takes the allocation of the subtasks to resources). It should be noted that this time only takes into account the task decomposition and the resource network organization, and does not take into account the time spent by the software services and participants on solving the subtasks assigned to them.

## 8 Conclusions

The paper addresses the problem of building decision support systems, that leverage not only computing power of modern hardware and software, but also rely on the expertise of human participants (allocated from a big pool).

The paper describes a spectrum of solutions for this problem, tailored for slightly different situations.

The first proposed solution is an application platform (or, Platform-as-a-Service) for the development of human-based application. The platform is intended for the use cases when a user can specify the exact information processing workflow and this workflow includes operations that has to be performed by human experts. The platform provides tools for deploying and running such application and manages human resources based on semantic descriptions and digital contracts.

The second is a decision support service based on ontological task representation and processing. This service is intended for the use cases where exact information workflow cannot be specified in advance, but there are a number of information processing tasks in the problem domain that can be used to automatically construct the workflow required by the end user (decision-maker) in an ad hoc way. The service decomposes the task into subtasks based on the task ontology and then distribute the subtasks among resources (human and software).

Finally, the paper presents an extension of the human-computer cloud, allowing to address complex problems for which it is hard to design a workflow in advance, and/or there is no detailed task ontology. This extension is represented by the concept of human-machine collective intelligence environment, created on top of the cloud resource management facilities. The distinctive features of the proposed environment are: a) support for human and software participants who can build coalitions in order to solve problems and collectively decide on the required workflow, b) support for natural self-organization processes in the community of participants.

Experiments with a research prototype have shown the viability of the proposed models and methods.

Overall, the proposed set of tools allow to build human-machine decision support systems for problems of varying complexity in variety of domains (e.g., smart city, business management, e-tourism, etc.).

**Acknowledgements.** The research was funded by the Russian Science Foundation. The HCC architecture, PaaS and ontology-based decision support service based on task decomposition were developed as a part of project # 16-11-10253, the self-organizing environment for collective human-machine intelligence is being developed as a part of project # 19-11-00126.

## References

1. Smirnov, A., Shilov, N., Ponomarev, A., Schekotov, M.: Human-computer cloud: application platform and dynamic decision support. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER, pp. 120–131 (2019)
2. Retelny, D., Bernstein, M.S., Valentine, M.A.: No workflow can ever be enough: how crowdsourcing workflows constrain complex work. *Proc. ACM Hum.-Comput. Interact.* **1**, Article 89 (2017)
3. Distefano, S., Merlino, G., Puliafito, A.: SAaaS: a framework for volunteer-based sensing clouds. *Parallel Cloud Comput.* **1**(2), 21–33 (2012)
4. Merlino, G., Arkoulis, S., Distefano, S., Papagianni, C., Puliafito, A., Papavassiliou, S.: Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Gen. Comput. Syst.* **56**, 623–639 (2016)
5. Formisano, C., Pavia, D., Gurgun, L., et al.: The advantages of IoT and cloud applied to smart cities. In: 3rd International Conference Future Internet of Things and Cloud, Rome, pp. 325–332 (2015)
6. Dustdar, S., Bhattacharya, K.: The social compute unit. *IEEE Internet Comput.* **15**(3), 64–69 (2011)
7. Sengupta, B., Jain, A., Bhattacharya, K., Truong, H.-L., Dustdar, S.: Collective problem solving using social compute units. *Int. J. Coop. Inf. Syst.* **22**(4), 1341002 (2013)
8. Kulkarni, A.P., Can, M., Hartmann, B.: Turkomatic: automatic recursive task and workflow design for mechanical turk. In: CHI 2011 Extended Abstracts on Human Factors in Computing Systems. CHI EA 2011, pp. 2053–2058. ACM (2011)
9. Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST 2011), pp. 53–64. ACM (2011)
10. Minder, P., Bernstein, A.: *CrowdLang*: a programming language for the systematic exploration of human computation systems. In: Aberer, K., Flache, A., Jager, W., Liu, L., Tang, J., Guéret, C. (eds.) SocInfo 2012. LNCS, vol. 7710, pp. 124–137. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-35386-4\\_10](https://doi.org/10.1007/978-3-642-35386-4_10)
11. Tranquillini, S., Daniel, F., Kucherbaev, P., Casati, F.: Modeling, enacting, and integrating custom crowdsourcing processes. *ACM Trans. Web* **9**(2), 7:1–7:43 (2015)
12. Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. Seattle, Washington, USA (2012)
13. Valentine, M.A., et al.: Flash organizations. In: 2017 CHI Conference on Human Factors in Computing Systems – CHI 2017, pp. 3523–3537. ACM Press, New York (2017)
14. Ko, R.K.L., Lee, E.W., Lee, S.G.: BusinessOWL (BOWL) - a hierarchical task network ontology for dynamic business process decomposition and formulation. *IEEE Trans. Serv. Comput.* **5**(2), 246–259 (2012)
15. Baader, F., Milicic, M., Lutz, C., Sattler, U., Wolter, F.: Integrating description logics and action formalisms for reasoning about web services, LTCS-Report 05-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany (2005). <http://lat.inf.tu-dresden.de/research/reports.html>
16. Ergu, D., et al.: The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. *J. Supercomputing* **64**(3), 835–848 (2013)

17. Kong, Y., Zhang, M., Ye, D.: A belief propagation-based method for task allocation in open and dynamic cloud environments. *Knowl.-Based Syst.* **115**, 123–132 (2017)
18. Sujit, P., George, G., Beard, R.: Multiple UAV coalition formation. In: *Proceedings of the American Control Conference*, pp. 2010–2015 (2008)
19. Kim, M.H., Baik, H., Lee, S.: Resource welfare based task allocation for UAV team with resource constraints. *J. Intell. Robot. Syst.* **77**(3-4), 611–627 (2015)
20. Kamar, E.: Directions in hybrid intelligence: complementing AI systems with human intelligence. *IJCAI Invited Talk: Early Career Spotlight Track.* (2016)
21. Nushi, B., Kamar, E., Horvitz, E., Kossmann, D.: On human intellect and machine failures: troubleshooting integrative machine learning systems. In: *31st AAAI Conference on Artificial Intelligence*, pp. 1017–1025 (2017)
22. Verhulst, S.G.: *AI Soc.* **33**(2), 293–297 (2018)
23. Smirnov, A., Shilov, N.: Service-based socio-cyberphysical network modeling for guided self-organization. *Procedia Comput. Sci.* **64**, 290–297 (2015)