Human-Computer Cloud: Application Platform and Dynamic Decision Support

A. Smirnov, N. Shilov, A. Ponomarev[a] and M. Schekotov
SPIIRAS, 14th Line, St. Petersburg, Russia

Keywords: Human-Computer Cloud, Human-in-the-Loop, Crowdsourcing, Crowd Computing, Human Factors.

Abstract: The paper describes a human-computer cloud environment supporting the deployment and functioning of human-based applications and allowing to decouple computing resource management issues (for this kind of applications) from application software. The paper focuses on two specific contributions lying in the heart of the proposed human-computer cloud environment: a) application platform, allowing to deploy human-based applications and using digital contracts to regulate the interactions between an application and its contributors, and b) the principles of ontology-based decision support service that is implemented on top of the human-computer cloud and uses task decomposition in order to deal with ad hoc tasks, algorithms for which are not described in advance.

1 INTRODUCTION

The proliferation of information and communication technologies allowing people to access global networks from almost any point on Earth pushes network-based collaborative and collective initiatives to a new level, resulting in an upsurge of crowd computing and crowdsourcing.

Common problem with the systems involving human participation is that each of these systems usually requires a large number of contributors and collecting this number of contributors may require significant effort and time. This problem is partially alleviated by crowdsourcing platforms (like Amazon Mechanical Turk, Yandex.Toloka etc.), providing tools for requesters to post tasks and an interface for workers to accomplish these tasks. However, existing platforms 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 designed for mostly simple activities (like image/audio annotation). The ongoing project is aimed on the development of a unified resource management environment, that could serve as a basis on which any human-based application could be deployed much like the way cloud computing is used nowadays to decouple computing resource management issues from application software. The proposed human-computer cloud (HCC) environment addresses all three cloud models: infrastructure, platform and software. Infrastructure layer is responsible for resource provisioning, platform layer provides a set of tools for development and deployment of human-based applications, and on top this environment there are several software services leveraging human problem-solving abilities.

This paper focuses on two specific contributions lying in the heart of the proposed human-computer cloud environment: a) application platform, allowing to deploy human-based applications (HBA) and using digital contracts to regulate the interactions between an application and its contributors, and b) the principles of ontology-based decision support service that is implemented on top of the human-computer cloud and that uses task decomposition routines in order to deal with ad hoc tasks algorithms for which are not described in advance.

The rest of the paper is structured as follows. Section 2 briefly describes other developments aimed on building hybrid human-computer cloud environments. 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. Sections 5

[a] https://orcid.org/0000-0002-9380-5064

Smirnov, A., Shilov, N., Ponomarev, A. and Schekotov, M.
Human-Computer Cloud: Application Platform and Dynamic Decision Support.
DOI: 10.5220/0007725201200131
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 120-131
ISBN: 978-989-758-365-0

and 6 describe cloud platform implementation and evaluation respectively.

2 RELATED WORK

Typical types of resources managed by cloud environments are hardware (CPU, storage) and software (cloud applications, platforms). Attempts of applying the principles of cloud computing (on-demand elastic resource provisioning) to a wider spectrum of resource types can be classified into two groups: 1) cloud sensing and actuation environments and 2) cloud-managed human resource environments.

One of the earliest examples of cloud sensing and actuation environment was presented in (Distefano et al., 2012), where sensing resource is regarded as a service that can be allocated and used in some unified way independently of the application that needs access to the resource. Based on this work Merlino et al., (2016) proposed a cloud architecture for mobile crowdsensing MCSaaS (Mobile CrowdSensing as a Service), which defines a unified interface allowing any smartphone user to become a part of a cloud and allow to use his/her smartphone sensors in some way that he/she finds acceptable in exchange for some monetary reward or even voluntary.

Another approach is proposed in ClouT (Cloud+IoT) project (Formisano et al., 2015). It is aimed on providing enhanced solutions for smart cities by using cloud computing in the IoT domain. Formisano at al. propose multi-layer cloud architecture where lower (infrastructure) layer manages both sensing and computing resources. Both ClouT and MCSaaS approaches are highly relevant to the environment presented in this paper. However, they are focused mostly on sensing and consider human resources only due to the fact, that human can provide the access to his/her smartphone and can control it to 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, or a specific kind of virtual sensor. A human, however, may be not only a supplier of information (like sensor), but a processor of it.

The second group, namely cloud-managed human resource environments, has another perspective aiming on managing member's skills and competencies in a standardized flexible way (e.g. Dustdar and Bhattacharya, 2011; Sengupta et al., 2013), regarding human as a specific resource that can be allocated from a pool for performing some tasks. For example, Dustdar and Bhattacharya (2011) consider the cloud consisting of human-based services and software-based services. 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 environment described in this paper adopts the idea of sensor virtualization and cloud implementation of IoT from (Merlino et al., 2016) and (Formisano et al., 2015), but also extends this idea by directly managing human resources by infrastructure layer (similar to (Dustdar and Bhattacharya, 2011)). Besides, the proposed approach includes 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.

3 HUMAN-COMPUTER CLOUD: PLATFORM-AS-A-SERVICE

This section introduces main design rationales and concepts of the proposed platform. It identifies 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.

There are three main categories of actors involved in the proposed cloud platform:

End users (*application/service developers*), who use the platform to create and run applications and/or services that require human effort. Of course, these applications can also use other services and hardware, like in any other PaaS.

Contributors, who are available to serve as human resources in a human-computer cloud environment.

System Administrators and Infrastructure Providers, who own and maintain the required 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 to estimate and monitor human resources available for the application. This requirement contrasts to conventional cloud applications, where overall amount of resources possessed by cloud provider is considered to be inexhaustible, and the capacity consumed by an application is in theory limited only by the available budget. However, human resources are always limited, especially when it comes to people with some specialized competencies and knowledge. Besides, the particular rewarding scheme designed by the application developer may be not appealing and not able to collect the required number of contributors. Therefore, application developer should be able to have information to know what capacity is available to the application. Based on this information he/she may change the rewarding scheme, set up his/her own SLA (for his/her consumers) etc.
- The platform must account for temporal dimension of resource availability. Like in the previous requirement, this is specific mostly for human resources. For example, some contributors are ready to participate in information processing activities controlled by the cloud platform in their spare time (non-working hours) only. It means that resource capacity during non-working hours will be larger than during working hours. However, for some applications (e.g., online tourist support) reaction time is important. Therefore, tourist decision support service developers should have temporal perspective of the resource availability.

3.1 Application Description

The aim of any cloud environment providing the PaaS 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. Instead, those resource management operations are performed automatically by PaaS environment usually according to some description (declarative configuration) provided by the developer of the service being executed. The human-computer cloud environment being developed supports similar approach, however, with inevitable extensions caused by the necessity of working with human resources. To streamline the development of applications that require human actions, the platform allows both a developer to describe what kind of human resources are required for this particular application, and a contributor to describe what kind of activities he/she can be involved in and what competencies he/she possesses. Declarative specification of service requirements is quite typical for cloud environments. They are used, for example, in cloud orchestration definition language TOSCA (Brogi et al., 2014), that allows, for example, to specify how many virtual machines and with what services should be created for a particular application running in cloud. However, these definitions turn out to be insufficient for the purpose of human-computer cloud. One of the reasons is multifarious nature 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. Besides, 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).

Application that is to be deployed in the proposed human-computer cloud beside the source code must contain a descriptor that includes following components (here we list all the components, but focus on those, 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 what human skills and competencies the application needs to function. These requirements are also (as software requirements) are resolved during the service deployment, but as (1) resolving these requirements employs ontology matching which may result in some tradeoffs, (2) human resources are much more limited than hardware/software, the status and details of the requirements resolution are available to the developer and can be browsed via the management console;
- digital contract template for each type of human resources. By the type of human resource we mean each specific profile 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. The application defines these two requirements profiles and may associate different digital contracts for them (in terms of reaction time and/or payment).

One of the distinguishing features of the proposed platform is the way to formally specify requirements addressing the different ways of describing the same human capabilities. First of all, we adopt the three-way understanding (knowledge, skill, and attitude) of human competencies, as it is common in current literature (Sampson and Fytros, 2008; Lundqvist et al., 2011; Miranda et al., 2017). Then (and the most important) we allow to use arbitrary ontology concepts as specific skills or knowledge areas. The major benefit of using ontologies is that it is possible to discover resources defined 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, or potentially even by ontology matching (Euzenat and Shvaiko, 2013).

It is important that we do not fix any particular set of ontologies, but support ontology-based resource discovery. 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 expressed in OWL 2 (OWL 2). Moreover, to guarantee computational efficiency, they have to conform to OWL 2 EL profile.

Another distinguishing feature of the approach is the concept of *digital contract*. It is an agreement between contributor and platform about terms of work, quality management principles and rewarding. This contract may be as lightweight as commonly accepted in modern microtask markets (like Amazon Mechanical Turk), specifying that a contributor may pick tasks from common service pools when he/she is comfortable and as many as he/she can. However, this contract may also be rather strict, requiring that a contributor should be available during the specified time and be able to process not less than specified number of tasks per time interval. Terms of this *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 should be informed about the human capacity available for his/her application to make an informed decision about revising *digital contracts* (making contribution to this application more appealing), or providing changes to 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

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, competencies definition, and work conditions. The competencies definition is made in terms of any ontology the contributor is aware of. For those contributors who cannot use ontologies there is another option, the definition of competencies is made iteratively via contributors' text description analysis followed by ontology-based term disambiguation. In any case, internally, each contributor is described by skills, knowledge and attitude, linked to concepts of some shared ontology.

The contributor's competency definition is multi-layered. The first layer is provided by the contributor him-/herself, while additional layers are added by applications in which a contributor takes part. For this purpose, a human resource management API available for the application code can manage application-specific skills and qualifications, which also can be described in some ontology (application-specific or not).

Therefore, despite initial description of competencies may be rather terse, during the contributor's participation in different applications running over the platform it becomes more and more rich. It alleviates further human resource discovery. Note, however, that each application can access its own contributor description layer, all the layers are visible only for deployment and resource discovery services of the platform.

Work 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 Platform Functions

Application/service developers can deploy application (which initiates advertisement process to identify human resources available to this application), edit digital contracts, monitor, and delete application (this releases all resources allocated for the application, including human resources). Editing digital contracts is necessary, for example, when application developer is trying to compete for resources with other applications by offering higher rewards. This effectively changes the descriptor of the deployed application (producing a new version of it) and leads to a new wave of advertisements. Monitor application use case generalizes various functions that are necessary for application developer and are common to many current PaaS, like reading usage statistics, logs and other. This also includes monitoring of the available human resources by each requirement type as well its prediction. Inner scenario of application advertising includes identifying compatible resources based on matching of resource definition (competence profile) and requirement specification.

Contributor can edit competence profile, providing the initial version of it or reflecting further changes, browse application advertisements routed to him/her (compatible with his/her competence profile and work conditions) with an option to accept some of them by signing digital contract and attaching to the respective application. Further, contributor can perform application-specific tasks and detach from application.

System administrator, besides monitoring the status of the platform (usage of hardware/human resources, communication channels throughput, platform services' health) can also do some activities in tuning the platform parameters, e.g., editing ontology mappings, that are used by the platform during identification of compatible resources (contributors) for advertising applications. The explicit mappings represent one of the ways for the platform to match competencies expressed in different ontologies.

3.4 Application Deployment Scenario

When a contributor registers at the human-computer platform he/she is not immediately available for requests of all human-based applications that may run on this environment. However, he/she starts to receive *advertisements* of applications, which are being deployed (or are already deployed) based on the similarity of a declared competence profile (including additional layers created by applications a contributor participates) and applications' competence requests and correspondence of digital contract templates of the application and working conditions of contributor. These advertisements include description of service (its end-user functionality), type of human-based activities that are required for this service, the proposed rewarding scheme, specific performance evaluation techniques and so on. Based on the information contained in the *advertisement,* a contributor makes a decision if he/she will receive tasks from this application in future and what is acceptable schedule and maximum load. In other words, if a registered contributor agrees to contribute to the particular service a *digital contract* is signed specifying 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.

After a contributor and the platform (on behalf of particular application) signed a *digital contract*, application's requests for human operations are made available to the contributor. 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).

As it was already noted, the process of service advertising is based on ontological representation of service requirements and human competencies and their matching.

4 ONTOLOGY-BASED DECISION SUPPORT SERVICE

The *René* decision support service is an application, running on top of the human-computer cloud infrastructure exposed as a SaaS and leveraging some features of the platform (e.g., resource management and provisioning). Expected user of *René* is a decision-maker who passes some task specification to the service to build an on-the-fly network capable of performing the task. It should be noted, that *René* exposes an API, by which ontology-based structured representation of the task specification is passed. 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.

To decompose a task specification into a smaller tasks *René* uses 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 a bit 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. In this sense, *René* is one of these human-based applications and may only distribute subtasks to those contributors who agreed to work with it. Software services are made available to *René* by their developers and maintainers by placing a special section into the deployment descriptor of the application. Practical assignment is also done via interfaces of underlying resource management layer, aware of the status and load of the resources and terms of their digital contracts.

Finally, *René* monitors the availability of the resources (via the underlying layer) during execution of the subtasks and rebuilds the assignment if some of the resources fail or become unavailable.

4.1 Task Decomposition

Task decomposition is the first step for building the resource network. Main operation that drives the decomposition is actually task composition, i.e. deriving networks of tasks connected by input/output parameters. Furthermore, task decomposition in this approach can be viewed as finding such composition of basic tasks defined in the ontology that is equivalent to the task given by the user.

For the purposes of decision support system, a structure of the task ontology is proposed. According to the proposed structure, the 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:

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

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 task decomposition ontology containing relationships between task and their subtasks in explicit form (Ko et al., 2012), in the proposed task composition ontology these relationships are implicit. Such principle of ontology structure 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. Thus, the proposed ontology opens the possibility to describe a number of different tasks in the same form and after that to construct a number of their compositions using appropriate criteria.

For the purpose of task composition ontology development the ontology language OWL 2 is used. The ontology is expressed by *ALC* description logic, which is decidable and has PSpace-complete complexity of concept satisfiability and ABox consistency (Baader et al., 2005) in the case when TBox is acyclic. In addition, SWRL-rules are specified for composition chain deriving. The main concepts of the ontology are "Task", "Parameter". The concept "Parameter" is used to describe a task semantically. 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.

The task should have at least one input and one output parameter. The parameter taxonomy in the OWL 2 ontology 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. The corresponding OWL 2 syntax expression is the Object Property. 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:

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

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.

The proposed task definition (2) is used for task composition process because in composition output parameters of one task are input for another. This relationship is used to construct task composition by the SWRL rule. The corresponding SWRL 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 essentially means that one task should be done after another. To encode this relationship in the ontology an object property has been created binding two tasks, where the domain is the predecessor task and range is the accessor one. Neither two tasks are connected by the property explicitly. The rule of task composition can be expressed as follows:

$$\text{hasInputParameter}(?ta, ?p) \wedge \text{hasOutputParameter}(?tb, ?p) \rightarrow \text{nextTask}(?tb, ?ta) \quad (3)$$

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

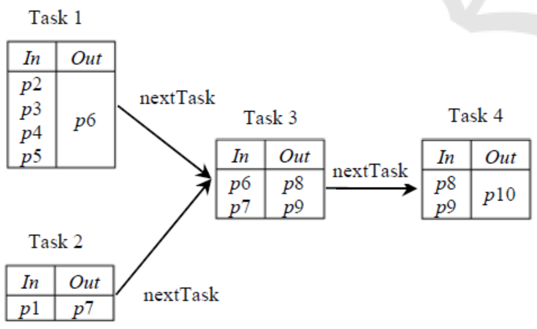

Figure 1: Task composition structure.

The proposed rule (3) allows to deriving all task connections by the object property "nextTask". The example of task composition is presented in Fig. 1. The abbreviation "ip" and "op" denote input parameter and output parameters accordingly. E.g., 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".

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

The specifics of the distribution of tasks in cloud computing systems is that usually there is a very large number of interchangeable computing resources (Ergu et al., 2013; Kong et al, 2017). This paper is focused on the solution of specialized tasks (subtasks) that require certain competencies, which on the one hand narrows the range of resources capable of solving these subtasks, and on the other hand requires taking into account these competencies. Therefore, typical algorithms used in cloud computing cannot be directly applied.

In the areas of distribution of tasks among the robots or agents that are most similar to those under consideration, the most common approach of instant distribution of tasks (instantaneous task allocation) (Sujit et al., 2008; Kim et al., 2015) focused on the dynamic uncertain environment. This approach involves tying tasks to resources that currently provide the maximum "benefit" according to the given priorities. This approach 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:

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

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

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

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

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:

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

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

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

4.2.1 Instantaneous Distribution of Tasks Algorithm

Since the problem is NP-complete, it is not possible to solve it by an exhaustive search method in a reasonable time (provided that a real-world problem is solved). As a result of the analysis of existing methods it is proposed to use the approach of instantaneous task allocation.

With regard to the problem, the algorithm based on the approach of instantaneous distribution of tasks is 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 $F(S_A) \to \min$, 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).

4.2.2 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, agents of 3 roles are singled out: a member of the coalition (an agent belonging to the coalition), a leader of the coalition (an agent negotiating on behalf of the coalition) and an applicant (an agent who can become a member of the coalition).

At the beginning of the negotiations, each agent forms a separate coalition (SC, which has the structure of the S_A solution), and becomes its leader. Suggestions of agents (tabular representation $F(s_{1,1}, s_{2,1}, \dots, s_{n,1})$ are published in all available agents repository of information on the blackboard. 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 process is terminated if at the next stage there is no changes in the composition of coalitions, after a specified time, when the permissible value of the objective function is reached.

5 IMPLEMENTATION

A research prototype of a cloud environment based on the proposed models and methods has been developed. In particular, two cloud computing mo dels were implemented: platform-as-a-service (PaaS) and software-as-a-service (SaaS). The platform pro-

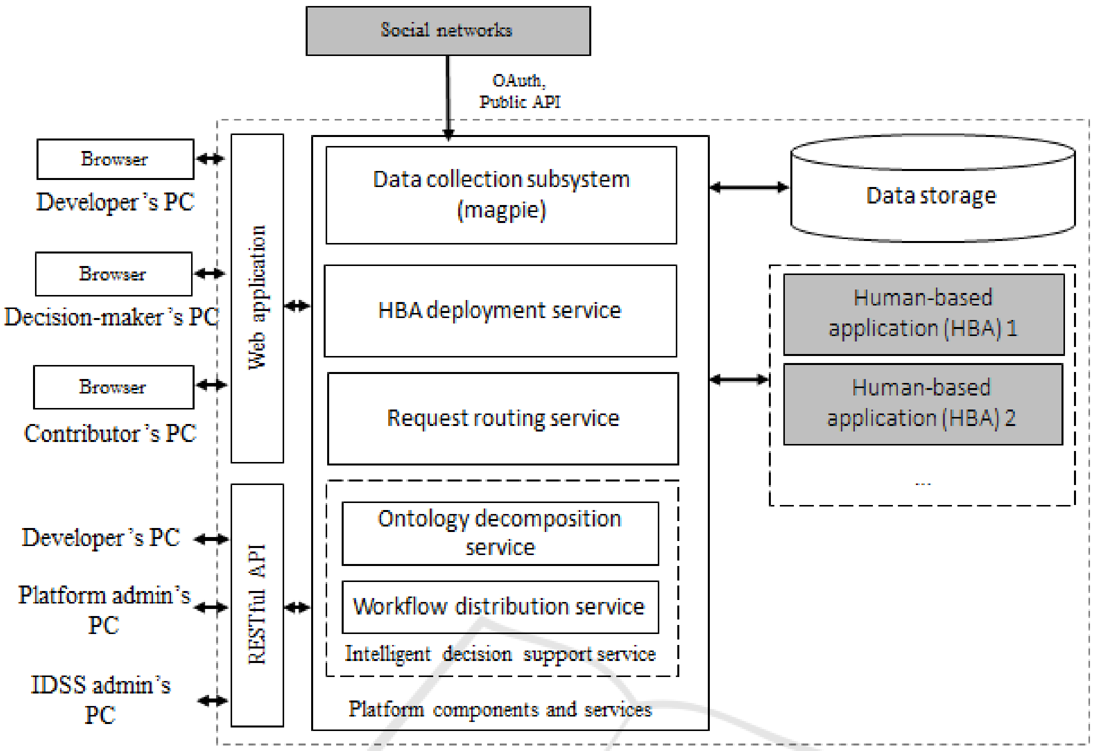

Figure 2: Platform architecture overview.

vides 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 (Fig. 2): 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.

The prototype environment interacts with social networks (ResearchGate, LinkedIn) to build and refine a participant competence profile. Also, the environment interacts with the applications deployed on it, providing them certain services (for instance, request human-participants, data warehouses, etc.).

Because at the heart of each application that uses human resources (HBA, human-based application), there is software code that provides a specific interface to the end users of this application, the open source platform Flynn[1] is chosen to support deployment and execution of this code. The capabilities of the platform are extended with 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 (e.g., Pellet, HermiT, etc.); 2) allocation of the subtasks to participants based on coalition games. The IDSS provides REST API to interact with the platform.

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

[1] http://flynn.com

- 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 ontology task decomposition service (Ontology Decomposition Service) and the workflow distribution service (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 (e.g., Pellet or HermiT). 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 level (Client Layer), REST API services are implemented for interacting with the platform.

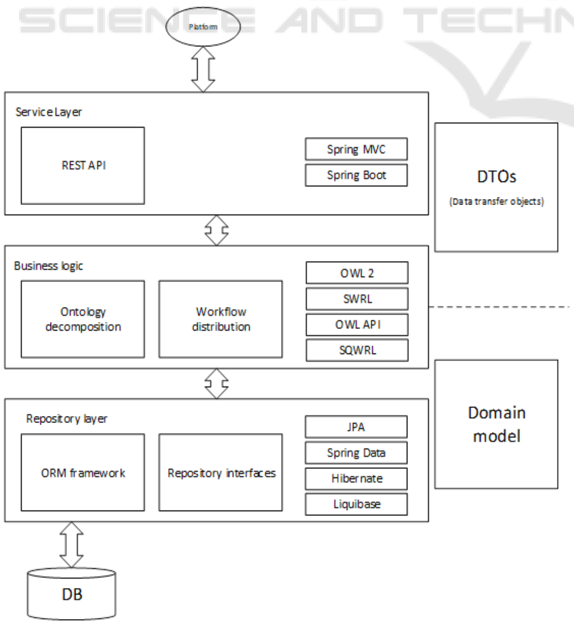

Figure 3: IDSS architecture.

6 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.

To evaluate the performance, 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 when deploying an application, in addition to creating Docker containers, compiling and launching an application (performed by Flynn), additionally a semantic search of participants is carried out using the competence ontology s and the comparison of digital contracts. However, due to this slight increase in application deployment time, the applications deployed in the implemented cloud environment gain 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.

The task ontology in 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 for testing the IDSS. In the course of the load testing, an attempt was made 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, on average, 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.

7 CONCLUSIONS

The paper describes main distinguishing features of the ongoing development of an ontology-driven human-computer cloud environment: application platform, simplifying the development and management of applications that require human information processing operations, and decision support service based on ontological task representation and processing.

The proposed resource management mechanism based on digital contracts makes the behavior of human-based applications more predictable and opens a way for building reactive human-based applications.

The proposed approach for decision support is based on two described methods:

- A method and algorithm to decompose a task into subtasks based on task ontology. As a result of applying this algorithm, a (probably complex) task set by the decision-maker can be decomposed into several simpler subtasks that can be accomplished by resources – either human or software.
- A method and algorithm to distribute the subtasks among resources based on coalition games.

The proposed methods and algorithms are used to implement decision support service *René* on top of the HCC, which allows to decompose a task received from a decision-maker and dynamically build a resource network (consisting of both software and humans) for it, capable of solving the task. *René* can be used in variety of domain areas characterized by rapid changes of the situation for building flexible automated decision support tools automatically configured by decision-maker.

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

ACKNOWLEDGEMENTS

The research was partially supported by the RSF (project # 16-11-10253), by the RFBR (# 19-07-00928, # 19-07-01120) and by Russian State Research # 0073-2019-0005.

REFERENCES

Baader, F., Milicic, M., Lutz, C., Sattler, U., Wolter, F., 2005. 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. Available at: http://lat.inf.tu-dresden.de/research/reports.html [Accessed 15 Feb. 2019]

Brogi, A., Soldani, J., Wang, P., 2014. TOSCA in a Nutshell: Promises and Perspectives. *3rd Service-Oriented and Cloud Computing (ESOCC)*, Springer, Lecture Notes in Computer Science, LNCS-8745, pp.171-186.

Distefano, S., Merlino, G., Puliafito, A., 2012. SAaaS: a framework for volunteer-based sensing clouds. *Parallel and Cloud Computing*, vol. 1, no. 2, 21-33.

Dustdar, S., Bhattacharya, K., 2011. The social compute unit. *IEEE Internet Computing*, 15(3), 2011, 64–69.

Ergu, D. et al. 2013. The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment, *The Journal of Supercomputing*, vol. 64, issue 3, pp. 835–848.

Euzenat, J., Shvaiko, P. 2013. *Ontology Matching*, 2nd edition, Springer-Verlag, Berlin Heidelberg (DE).

Formisano, C., Pavia, D., Gurgen, L., Yonezawa, T., Galache, J.A., Doguchi, K., Matranga, I., 2015. The advantages of IoT and cloud applied to smart cities. *3rd International Conference Future Internet of Things and Cloud*, pp. 325-332.

Kim, M.H., Baik, H., Lee, S., 2015. Resource welfare based task allocation for UAV team with resource constraints, *Journal of Intelligent & Robotic Systems*, vol. 77, issue 3-4, pp. 611–627.

Ko, R.K.L., Lee, E.W., Lee, S.G., 2012. BusinessOWL (BOWL) - a hierarchical task network ontology for dynamic business process decomposition and formulation, *IEEE Transactions on Service Computing*, vol. 5, issue 2, pp. 246–259.

Kong, Y., Zhang, M., Ye, D., 2017. A belief propagation-based method for task allocation in open and dynamic cloud environments, *Knowledge-Based Systems*, vol. 115, pp. 123–132.

Lundqvist, K., Baker, K., Williams, S., 2011. Ontology supported competency system. *International Journal of Knowledge and Learning*, 7 (3/4), pp. 197–219.

Merlino, G., Arkoulis, S., Distefano, S., Papagianni, C., Puliafito, A., Papavassiliou, S., 2016. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Generation Computer Systems*, vol. 56, 623-639.

Miranda, S., Orciuoli, F., Loia, V., Sampson, D., 2017. An Ontology-Based Model for Competence Management. *Data & Knowledge Engineering*, vol. 107, pp. 51–66.

OWL 2 Web Ontology Language Document Overview (Second Edition). Available at: https://www.w3.org/TR/owl2-overview/ [Accessed 15 Feb. 2019]

Sampson, D., Fytros, D., 2008. Competence Models in Technology-Enhanced Competence-Based Learning. *Handbook on Information Technologies for Education and Training*, International Handbooks on Information Systems, Springer Berlin Heidelberg, pp. 155–177.

Sengupta, B., Jain, A., Bhattacharya, K., Truong, H.-L., Dustdar, S., 2013. Collective problem solving using social compute units. *International Journal of Cooperative Information Systems*, vol. 22, no. 4.

Sujit P., George, G., Beard, R., 2008. Multiple UAV coalition formation, *in Proceedings of the American Control Conference*, pp. 2010–2015.