Computing and Informatics, Vol. 35, 2016, 1005–1026

LEARNING AGENT FOR A SERVICE-ORIENTED CONTEXT-AWARE RECOMMENDER SYSTEM IN HETEROGENEOUS ENVIRONMENT

Piotr Nawrocki, Bartłomiej Śnieżyński, Jakub Czyżewski

AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30
30-059 Krakow, Poland
e-mail: piotr.nawrocki@agh.edu.pl

Abstract. Traditional recommender systems provide users with customized recommendations of products or services. They employ various technologies and algorithms in order to search and select the best options available while taking into account the user's context. Increasingly often, such systems run on devices in heterogeneous environments (including mobile devices) making use of their functionalities: various sensors (e.g. movement, light), wireless data transmission technologies and positioning systems (e.g. GPS) among others. In this paper, we propose an innovative recommender system that determines the best service (including photo and movie conversion) and simultaneously accommodates the context of the device in a heterogeneous environment. The system allows the choice between various service providers that make their resources available using cloud computing as well as having the services performed locally. In order to determine the best possible recommendation for users, we employ the concept of learning agents, which has not been thoroughly researched in connection with recommender systems so far.

Keywords: Service-oriented recommender system, context-aware, heterogeneous environment, learning agent, supervised learning, cloud computing

1 INTRODUCTION AND MOTIVATION

The concept of recommender systems was born in the 1990s [19] when client-server architecture was broadly applied and heterogeneity was not crucial. This is why such systems, which supported the users' decision-making, did not accommodate the peculiar features of heterogeneous environments at the beginning. Nowadays, there are many types of devices on which such systems may be executed. The use of devices in heterogeneous environments (characterized by mobility, different processing power and the ability to use different data transmission technologies) has enabled the extension of their functionality and the creation of systems that are aware of their position and time, among other things. Therefore, recommender systems should be able to use the functionalities of such devices, including wireless connections, various sensors (accelerometer, light, etc.), and positioning systems.

Currently, research on such systems focuses mainly on supporting decision-making in terms of selecting movies, music, TV channels, books, applications, etc. [20]. However, recommender systems as tools for selecting the best service and the place where it is to be performed have not been thoroughly researched yet.

In our research, we focused on the possibility of building a novel recommender solution for heterogeneous environments that would deal with the choice of services (and the locations to perform them) and accommodate the context. In order to obtain service recommendations, we used the learning agents concept, which has not been thoroughly researched in this field so far. Only a few authors [7] attempted to make use of the learning agents concept in connection with recommender systems.

Classic recommender systems search a significant number of information sources in order to find the best product. As far as our research is concerned, we decided to focus on the system's recommendations concerning the location to perform the service rather than on the service search itself. We assume that the user knows what kind of service he or she is looking for (e.g. movie conversion, text recognition, etc.), but does not know where to have it performed most efficiently. We considered the mobile device itself or various Mobile Cloud Computing (MCC) platforms (such as Google Cloud Computing or Amazon EC2) as potential locations to perform the service. Numerous providers may make a certain service available using mobile cloud computing and the recommender system should be able to determine the place where the service will be performed in the optimal way. While conducting our research, we took into account the context of service performance on a mobile device, including the positioning of the device, the condition of its battery, the possibility of data transmission using wireless data transmission technologies (Wi-Fi, 3G, 4G) [17] and the time of service performance as well as its cost. In order to suggest the best (least costly and optimal from the point of view of the mobile device) location for service performance, we made use of the learning agents mechanism.

The paper is structured as follows: Section 2 includes the review of related work, Section 3 focuses on using learning agents in optimizing service performance, Section 4 describes a context-aware recommender system for heterogeneous environments that recommends video conversion services, Section 5 presents the results of the tests conducted, and Section 6 contains the conclusion.

2 RELATED WORK

In recent years, the significance of recommender systems that operate in heterogeneous environments and of context information has greatly increased. Therefore, numerous papers describing various concepts related to such systems have been written.

In [8] the authors present a context-aware recommender system named South Tyrol Suggests (STS) that recommends places of interest (POIs) using mobile devices. The system learns the users' preferences on the basis of their previous reviews and personal choices.

In another paper [29] the authors propose a recommender system based on social networks. The inclusion of such information may significantly enhance the outcome and provide more accurate recommendations. Empirical analysis of real data has shown that the solutions proposed achieve better results than earlier systems of the same kind.

The authors of yet another paper [12] offer an example of a classic recommender system called RecomMetz, which focuses on movies, cinemas and movie screenings. The system is based on Semantic Web technologies and accommodates various parameters such as positioning, numbers of viewers and time. The system was tested in a real-life environment on various Android mobile devices.

There are also recommender systems using hardware solutions. In [30] the authors present a Collaborative Filtering (CF) recommender system using a Single Chip Cloud computer (SCC) and a Field-Programmable Gate Array (FPGA) [9, 11].

Recommender systems based on services are also a valid area of research. The Context-Aware Recommender System (CARS) described in [1], is one such system. Based on the Personalized Access Model (PAM) framework, it provides recommendations based on services that make context available. The use of services [10] (or hardware-based services [5]) in recommender systems is related to the problem of service search [16] and the SLA (Service Level Agreement) concept [15] that defines the terms of using certain services.

There are several other studies of multi-agent system applications in heterogeneous environments [2, 21]. A problem similar to the one considered here is presented in [4], where an agent-based system for MCC optimization is investigated. The main component of the system is the Execution Manager, which is a service on a mobile device that is responsible for deciding where to execute application components. In order to make this decision, a cost model is used, in which execution times are collected offline by the application profiler (in our solution we leverage online learning). Learning on a mobile device is used in [22], where an agent performing behavioral detection is discussed that samples selected system metrics (e.g. CPU usage, network communication, active processes, battery level) and detects anomalies using classification methods.

Autonomous reasoning about resources and tasks by agents is discussed in [32]. Domain knowledge is represented using ontology but no learning is used. The application of machine learning algorithms in agent-based systems has been broadly discussed in literature. Valuable studies in this area include [18, 31]. In most cases, reinforcement learning or evolutionary computations are applied for the purpose of agent adaptation. However, there are also works where supervised learning was used [24, 13], like in this paper. Such learning was applied for the purposes of adapting service-oriented applications as described in [28]. It is an approach similar to the one presented in this paper, but the choice of the service is based on one criterion only (calculation time). It is worth noting that there are advantages to supervised learning since it accelerates the learning process compared to reinforcement learning, especially if the state space is large [25, 27]. In contrast to evolutionary computation (where a population of agents is necessary), a single agent can learn a strategy autonomously [26].

Several studies of using genetic algorithms and machine-learning methods on mobile devices, e.g. for computation offloading, have been published. The MALMOS environment [14] is an adaptable offloading scheduler. It works in two modes: the scheduling phase (in which tasks are executed locally or remotely) and the online training phase (in which two separate threads for offloading and local processing are created to compare execution times). In contrast, our method does not need a separate training phase. Three supervised learning algorithms are applied: instance-based learning, single layer perceptron, and Naïve Bayes. Potentially, various attributes may be used to describe the task and the context, but in experiments described in the article, only the amount of data to be sent and network bandwidth were taken into account. During these experiments, a Traffic Control (TC) tool was applied to emulate various types of networks between mobile devices and laptops/PCs; in our experiments, the mobile device connects to the Google cloud using real LTE and EDGE networks.

In [6] the authors present the partitioning of monolithic applications and the appropriate allocation of the resulting parts in order to optimize energy consumption of a mobile device. In order to properly deploy the tasks on the resources, they used two genetic algorithms. The simulations performed showed that the proposed method of task deployment made it possible to save 35 % energy of the mobile device. In contrast to our solution, the authors of the article did not use machine-learning algorithms but genetic algorithms to optimize the allocation of resources. The optimization only takes into account energy consumption and the task execution time is not included. The tests were carried out only by simulation without using real devices and therefore the authors did not take into account the characteristics of wireless transmission using technologies such as EDGE or LTE.

In [3] machine learning algorithms are applied in the context of mobile devices. However, rather than touching upon the problem of task allocation that we are working on, mobile device location is predicted using supervised learning with a sliding window. Three algorithms included in the Weka [33] library are applied: 1NN, C4.5, and voting (based on 1NN and C4.5). Experiments are also performed offline on previously generated data, while in our experiments learning is performed online.

In [23] task execution time (on the mobile device and in the cloud) is predicted using machine learning. The prediction system consists of two components: an offline one (developing a task performance model for each computational device before deploying the app) and an online one (applying the model learned to the current case). Additionally, Internet connection quality is predicted based on historical information about the user and signal strength. Risk-controlled decisions are made where to execute the task. Our system includes exclusively an online part, which learns local, device-specific models and therefore no task performance model needs to be developed offline for every computational device.

3 SERVICE RECOMMENDATIONS BY LEARNING AGENTS

In the application considered, recommendations are provided by an agent, which operates on a device in a heterogeneous environment and monitors the environment and may collect information on:

- the task to be executed in the service in question, including its type, key arguments, estimated data input/output size, estimated execution time, the cost of performing the computation and the time when the result is required;
- the cost of performing the service in the cloud, which affects the assessment of service cost-effectiveness;
- other parameters of the cloud service (such as quality, reliability, speed). Cloud-based service parameters may also be described according to the SMI (Service Measurement Index) which allows the user to rate various types of services in terms of six key metrics (quality, agility, risk, cost, capability, and security);
- in case of the mobile device, the location of the device (domestic/roaming), indicating whether the device uses data transmission services from local carriers (lower costs) or must use roaming services abroad (higher transmission costs);
- possible device connection modes (wired/wireless), technologies (Ethernet, Wi-Fi, 2G/3G/4G) and connection quality affecting network throughput between the device and the cloud. The type of connection may also affect the power consumption of the device;
- battery status specifies for how long the device can operate (without being plugged in) and for how long the service can be performed on this device;
- the current time and date (including day of week, holidays, etc.) affect the ability to take advantage of better rates related to data transmission or to transmit/receive data during periods when the carrier's infrastructure is less busy;
- readings of sensors such as the accelerometer, light sensor, etc. for determining the status of the device (device movement, ambient lighting, etc.).

On the basis of the information collected, the agent makes decisions when and where to perform the service (locally or in the cloud). After completing the task, the agent assesses its decision, considering one or more criteria such as:

- device power consumption;
- the time spent waiting for the result;
- the user's satisfaction (the user could override the agent's decision, which means that he or she does not agree with it);
- costs (e.g. charges related to data transfer or using cloud resources).

The agent gathers experience and updates its strategy, applying some learning algorithm. Simultaneously, the agent may generate:

- models of the user's behavior that enable it to assess the impact of factors such as his or her location/connection accessibility/ability to charge the device;
- models of estimated outcomes of performing a service locally/in the cloud (energy consumption, time).

These models may be used to improve the estimates of decision consequences.

Energy consumption and time during task execution depend on the task type and parameters, context and device characteristics (CPU, GPU, RAM etc.). The first two will be observed by the agent. The latter will be represented indirectly by observing execution time, because learning is local (performed on a device) and therefore the knowledge learned is device-specific. If the knowledge were general, covering various devices, such characteristic should be also taken into account.

Let us define the Task Allocation Learning Agent (Ag) as a tuple:

$$Ag = (T, S, A, GK, TD, Act), \tag{1}$$

where T is a set of computational tasks, S is a set of states of the heterogeneous environment (connection, date, processing power, cloud service description, available energy source, etc.), A is a set of attributes, which are used to describe percepts, GK is *generated knowledge*, and TD is *training data*. The goal of the agent is to select an action $a \in Act = \{l, c\}$, which corresponds to engaging local or cloud resources for a given task in a given state $(t, s) \in T \times S$, using GK.

The agent observes $x = (t, s) \in T \times S$, and its processing module describes it with attributes $O = \{o_1, o_2, \ldots o_n\}$ which yields

$$x^O = (o_1(x), o_2(x), \ldots o_n(x)), \tag{2}$$

i.e. a description of the *Problem*. Next, using the knowledge stored in GK it solves the *Problem* by selecting $a \in Act$, which has the minimum predicted cost. If GK is empty, a is randomized.

The agent's action a is then executed and the task is run locally or in the cloud. The agent observes execution results, which are described by attributes

$$R = \{r_1, r_2, \dots r_m\} \tag{3}$$

(e.g. whether execution was successful $es(x, a)$, battery consumption $b(x, a)$, calculation time $ct(x, a)$ and user's dissatisfaction $d(x, a)$, which may be measured by observing if the user overrode the agent's decision). Therefore, the set of all attributes used to describe percepts is a sum of O and R

$$A = O \cup R. \tag{4}$$

The agent stores these results together with x^O and action a in TD. Therefore the complete example description stored in TD has the form

$$x^{A \cup Act} = (a_1(x), a_2(x), \dots a_n(x), r_1(x, a), r_2(x, a), \dots r_m(x, a), a). \tag{5}$$

The models to predict R values based on x and a are constructed using machine learning algorithms and stored in GK. These models influence the choice of the action to be taken. They are independent and may be learned separately, using a different algorithm for every model. Any supervised learning algorithm may be used. However, one should remember that learning is executed on a device in a heterogeneous environment (using battery power and having limited resources), and therefore algorithms with low computational complexity are preferred. In our experiments, we applied the Naïve Bayes and random forest algorithms (see Section 5 for details).

Using value predictions $r_i(x, d)$, the agent may rate its decisions $d \in Act$ by calculating predicted expenses $e(x, d)$:

$$e(x, d) = \sum_{i=1}^{m} w_i * r_i(x, d), \tag{6}$$

where w_i are weights of the result r_i.

The agent should select the action for which execution is predicted to be successful and the expense is predicted to be the lowest.

The internal structure of the agent should reflect its learning abilities. The architecture of the agent is presented in Figure 1; the agent consists of four main modules:

- *Processing Module*, which is responsible for basic agent activities such as processing the percepts, storing training data, executing the learning process and leveraging the knowledge learned;
- *Learning Module*, which is responsible for executing the learning algorithms and providing answers to problems using the knowledge learned;

- *Training Data*, (TD) which provides storage for the examples (experience) used in learning;
- *Generated Knowledge*, (GK) which provides storage for the knowledge learned (models).

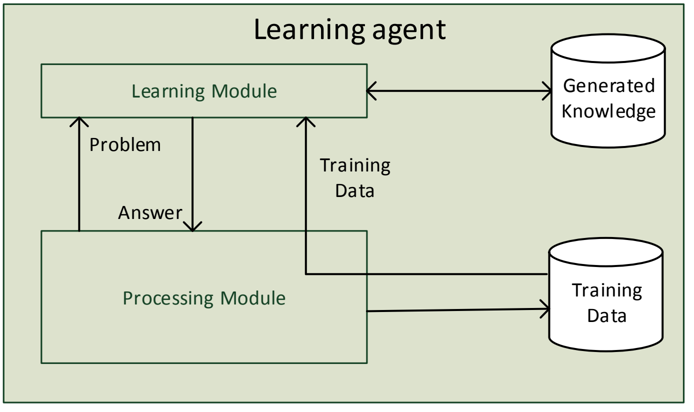

Figure 1. Agent architecture reflecting its learning abilities

These components interact in the following way: the *Processing Module* receives *Percepts* from the environment (the parameters listed in the previous subsection), may process them and execute *Actions*. If the knowledge learned is required during processing, it formulates a *Problem* by describing observations with available attributes and sends it to the *Learning Module*, which generates an *Answer* for the *Problem* using GK. The *Processing Module* also decides what data should be placed in TD storage. When required (e.g. periodically or when TD contains many new examples), it calls the *Learning Module* to execute the learning algorithm that generates new knowledge from TD. The knowledge learned is stored in the GK base.

The agent's algorithm, which is executed in the *Processing Module*, is presented in Figure 2. At the beginning, GK and TD are empty. Next, if there is no learned knowledge, the action is randomized. If there is some knowledge, *Problem* is a description of the observations (x^O). Next, *Learning Module* is used to select the best action for the *Problem* according to the current knowledge. Results are observed and example ($x^{A \cup Act}$) is stored in the TD. After processing a given number of tasks, *Learning module* is called to generate a new knowledge from TD. This knowledge is stored in GK.

The form of the knowledge stored in GK depends on the learning algorithm utilized. It may have an explicit form, e.g. rules, a decision tree or a Bayesian model in the case of supervised learning. It may also be stored in a lower-level form such as parameters representing a linear regression model, an action-value function

```
begin
    Generated Knowledge:= ∅;
    Training Data:= ∅;
    while agent is alive do
        if Generated Knowledge = ∅ then
            a:= random action
        end
        else
            Problem := description of the current
              (observed) state;
            a:= action determined for Problem by
              model(s) stored in Generated Knowledge
        end
        execute a;
        observe execution results;
        store example in the Training Data;
        if it is learning time (e.g. every 100 steps)
          then
            learn from Training Data;
            store knowledge in Generated Knowledge;
        end
    end
end
```

Figure 2. Learning agent algorithm that makes it possible to generate the agent's strategy using online supervised learning based on the agent's experience

or a neural network approximator of such a function if reinforcement learning is applied. It is also possible to store multiple models in GK.

4 SERVICE-ORIENTED CONTEXT-AWARE RECOMMENDER SYSTEM USING LEARNING AGENTS

Peculiar features of systems that use mobile cloud computing have necessitated a different look at recommender systems. The mobile cloud computing environment consists of devices that use wireless data transmission technologies, among other things. The use of such devices may result in a significant variability in terms of available network communication bandwidth and – since they rely on battery power – limit the possible time during which the service can be used. Therefore the classic approach, in which the best services (available through cloud computing) are recommended mainly on the basis of their ratings, must be modified. Recommender systems should accommodate cases where using a service locally on a mobile device is better than leveraging cloud computing.

On the basis of prior analysis of existing solutions related to recommender systems and machine learning methods, in this paper we propose an innovative service recommendation system that operates in a heterogeneous environment and takes into account the context and employs learning agents.

The general concept of the system's operation is presented in Figure 3. The decision-making process recommending the location where the service can be performed (through cloud computing or on a mobile device) incorporates two stages. During the first stage, a recommendation request is processed by a classic recommender system that chooses the cloud computing system available in which the service can be performed best. The process that determines which cloud computing system should be selected takes into account parameters such as, *inter alia*, the cost of the service and previous ratings by users. In the first stage of implementation (the scope of classic recommender systems), various solutions may be applied. The choice of this solution is beyond the scope of this paper. During the second stage, on the basis of the data collected and the device context (location, time, battery level, movement of the device), the learning agent chooses where the service should be performed – using the cloud computing system selected in the first stage or on the mobile device. The agent should be aware of the fact that there are several clouds with possibly different characteristics and therefore it should observe the cloud selected in the first stage, allowing it to account for cloud characteristics in the learned knowledge. The detailed structure of the recommender system developed is presented in Figure 4.

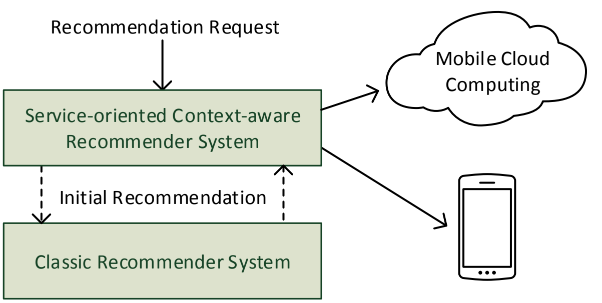

Figure 3. Processing the recommendation request is divided into two phases: classic recommendation, which is based on user opinions processed in a central system, and context-based stage involving local learning on a mobile device using the service characteristics observed

A user who wants to receive a recommendation concerning a certain service (e.g. a video file processing service) sends a request to the Recommendation Engine (RE), which is the main element of the novel solution developed by us. During the first stage, the request is sent by the Processing Module (PM) to the classic recommender system that selects the service performed by the best cloud computing system in its opinion. In order to determine which cloud computing system is the best, various parameters are taken into account, including the cost of performing the service and the ratings given by previous users to the service (and the cloud computing system). As a result of the operation of the classic recommender system, a single cloud computing system and the video file processing service offered by it are indicated. In the second stage, the PM uses information from various sensors present in the mobile device (accelerometer, GPS module, light sensor, clock) to determine the context in which the service is to be performed. The context, information about the cloud computing system and the possibility to perform the service directly on the mobile device constitute the basis on which the recommendation process using the Learning Module operates. Based on the data collected and training data obtained from previous recommendations, information about which service should be selected, i.e. where the video file processing service should be performed, is sent by the PM to the user. The user, in turn, can express satisfaction or dissatisfaction after the service has been rendered. This information may be taken into account by the PM and the learning agent during the learning process and by the classic recommender system in the rating of the service offered by the cloud computing system. The ratings (user satisfaction or dissatisfaction) of service provision quality include a context in which a given service was provided, i.e. among others: the communication technologies used (e.g. usage of WiFi, 3G or EDGE), the parameters of a given mobile device and other information on where the service was provided: in a cloud or on a mobile device. Adding the context to the service rating, allows a classic recommender system to create a database of user ratings including information relating to the environment in which the service was activated. Such a solution improves the precision of the process for recommending services in a classic recommender system. Examples of video file processing service ratings are presented in Figure 4 in the form of stars.

During learning, two models are built. The first one is GK_{succ} – a classifier, which allows to predict the *successful* category from $O \cup Act$ attributes. The second one is GK_{ct} – a random forest model which is used to predict *calctime* from $O \cup Act$. The *calctime* is discretized into 5 equal ranges. As a result, $GK = (GK_{succ}, GK_{ct})$.

The learning agent only takes $ct(a)$ into account in cost calculations ($w_b = w_d = 0$, $w_{ct} = 1$). Hence, the action is selected in the following way: first, GK_{succ} is used to determine if for the task observed t and state s calculations will be successful locally: $success_l = GK_{succ}(t, s, l)$ and using the cloud: $success_c = GK_{succ}(t, s, c)$. Next, computation time is predicted for both resources: $ct_l = GK_{ct}(t, s, l)$, $ct_c = GK_{ct}(t, s, c)$. If $success_l$ is true and $success_c$ is false, local execution $a = l$ is selected. Conversely, if $success_c$ is true and $success_l$ is false, cloud execution $a = c$ is selected. If both predictions are false, the action is randomized with uniform probability distribution. If both predictions are true, the action is selected based on predicted calculation time: if $ct_l > ct_c$ the cloud is selected as the execution place, otherwise local execution is chosen. If $ct_l = ct_c$, the action is selected randomly. The action selected is returned by the PM as a recommendation in the second stage.

Recommendation divided into two stages – initial and subsequently using context and learning mechanisms – makes it possible to account for the ratings given by users

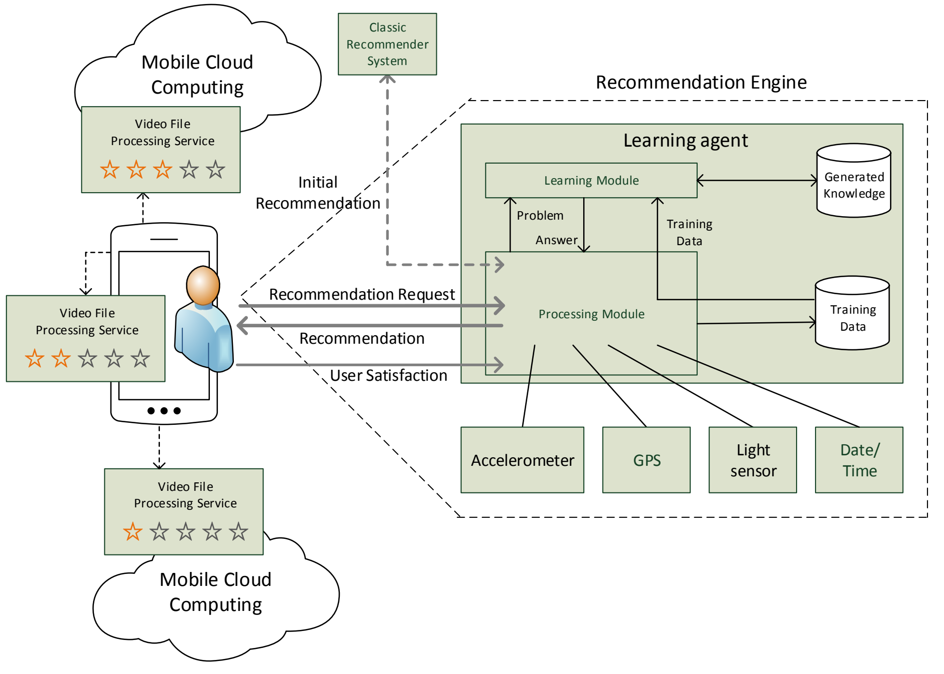

Figure 4. Service-oriented context-aware recommender system architecture

in connection with the performance of the service by a certain cloud computing system. As a result, peculiar features of mobile devices and mobile cloud computing systems are accommodated.

5 EXPERIMENTAL RESULTS

In this section, we describe experiments where the purpose was to check how our novel solution performs in a real-life heterogeneous environment. We assume that the first stage, i.e. the classic recommender system, is well tested and therefore we have concentrated on testing the second stage, where we focused on estimating the calculation time required for the task to be performed in a given location. Experiments were performed using an application written especially for this purpose[1].

Every experiment consists of a series of task packages being executed. Every task package is a combination of selected task parameters and system state values. Each test package involves measuring conversion time for the multimedia file in question on the mobile device and using MCC for various file sizes (with the maximum file size being 650 kB), five different conversion types and three network connection

[1] Software may be downloaded from the following address: `http://home.agh.edu.pl/piter/resources/RecommenderSystem/software-v0.1.zip`.

statuses (LTE/HSPA/EDGE) representing the context (adding other parameters to the context is planned as a future work). Each package includes tasks that may potentially run faster in the cloud (e.g. conversion while connected via LTE) as well as tasks where local execution may be more beneficial in terms of computation time (e.g. conversion while connected via EDGE). During the series of measurements conducted, information is collected on conversion time and its result (task execution success or failure). Examples ($x^{O \cup R \cup Act}$) are stored in TD. After every task package executed, the decision module initiates the learning process and builds the GK_{succ} classifier based on the *Naïve Bayes* and GK_{ct} classifier based on *random forest model*, which are used to process the task package in the next round. Learning algorithms were selected during preliminary experiments and the best performance criterion was applied. Numerical values of the time observed are grouped into seven bins of equal frequency using unsupervised discretization. We have used Weka [33] implementations of the classifiers with default values of all parameters. During round n, the decision module uses the examples collected in rounds $1 \dots n-1$ for learning purposes. When the series is completed, TD and GK are cleared and the next series is executed to collect statistical data. For every round, task package execution time is measured. If task execution results in failure, task execution time is set to the maximum successful execution time observed.

The experiment consisted of executing tests in series of ten rounds, which were repeated eight times. The results of that experiment are shown in Table 1.

Round execution time (s)	1	2	3	4	5	6	7	8	9	10
Minimum	2 126	2 592	2 222	2 123	2 544	2 619	2 090	2 205	2 404	2 362
Maximum	7 847	7 753	8 593	6 963	4 289	5 417	5 257	6 947	5 188	5 720
Average	4 214	4 765	4 723	3 803	3 347	3 601	3 442	3 512	3 341	3 535

Table 1. Experiment results – task package execution time for ten subsequent rounds

As Table 1 demonstrates, the application of machine learning in making the decision as to the location where the task should be performed (mobile device/MCC) significantly increases execution speed for the tasks requested. The mean task execution time in subsequent rounds of tests using the knowledge gained as a result of machine learning shows a downward trend. Figure 5 shows how task package calculation time changes in ten subsequent rounds. One may notice that after the initial random round, time increases in round two. At this point, after a random round, the system does not have sufficient knowledge about the environment and how tasks are performed for each combination of system state values. As the system gains more information, times drop notably until round 5 where average execution time reaches its minimum. After that, the value does not change significantly, which means that sufficient knowledge has been gathered by the system to perform optimally. Further learning may cause overfitting and thus increase the execution time.

Figure 5 also presents the results of executing the entire task package in a single location (in the cloud and locally). Experimental results show that the execution time of multimedia file conversion services was significantly decreased and after three

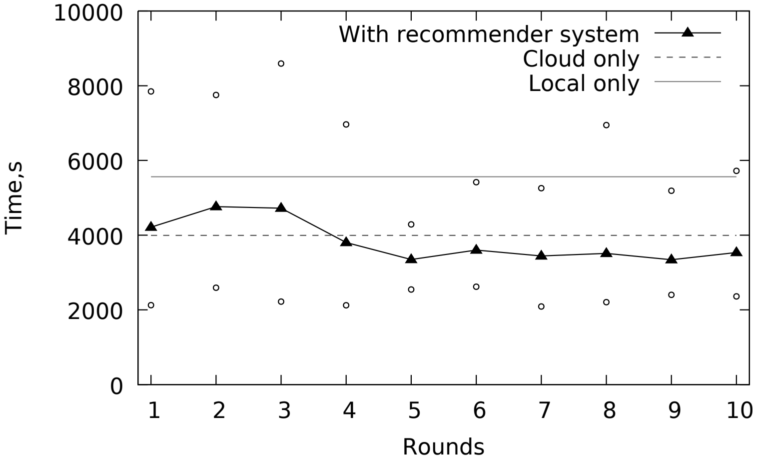

Figure 5. Average task package calculation time (triangles) and minimum/maximum values (circles) for ten subsequent rounds

model updates (rounds) it is lower than both single-location execution times. This suggests that the autonomous context-based service optimization method, which we have proposed, may be applied to improve Quality of Experience in real scenarios as part of a Mobile Recommender System.

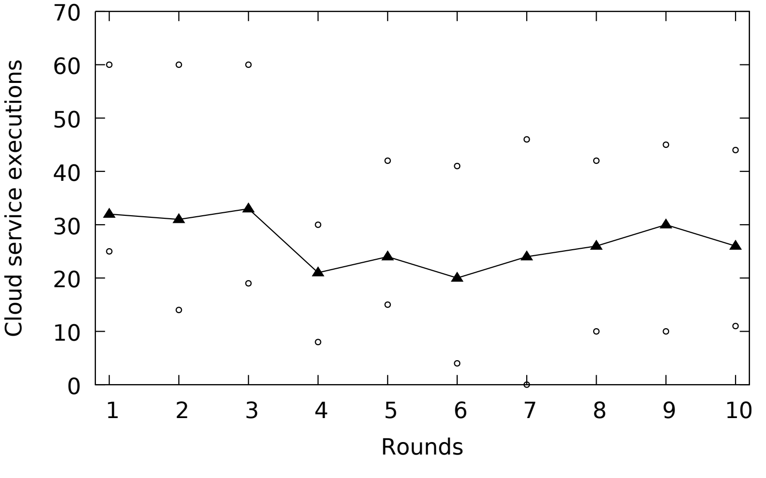

Figure 6. Average number of service executions in the cloud (triangles) and minimum/maximum values (circles)

We have also checked how the location of executing the service (in the cloud or locally) changes. Figure 6 presents the average numbers of times the multimedia file conversion service was started within the cloud in subsequent rounds. The number of all service calls in every round was 60. It can be noticed that, based on Figure 5 and Figure 6, shorter service execution times occur when most calls are executed on the device. However, it should be remembered that executing the service locally (on a mobile device) in a heterogeneous environment may result in a higher load on the device and the resulting faster battery drain leading to a shorter device operation time. Moreover, executing all services locally in all conditions results in a higher total execution time (see Figure 5). This means that the agent has to learn the execution strategy, which is not trivial.

It should be noted that the experiments were performed in a real-world MCC environment (using Google App Engine) and using a real mobile network. As a result, we cannot control transmission delays or cloud load. Therefore it may happen that for some tasks executed in the cloud the execution time is much higher than the average.

To visualize the knowledge learned, we have applied the C4.5 algorithm to generate GK_{ct} from the training data collected by the agent in one of the series of previous experiments. To make the knowledge more compact, the time observed was discretized into three bins of equal frequency (with values FAST, MEDIUM, SLOW). The accuracy of this model calculated using 10-fold cross validation is 81.6. The classifier is presented in Figure 7. The decision tree has 53 nodes. Part of the knowledge can now be analyzed. In line 3 we see the leaf with FAST category, which means that for small multimedia files, EDGE connection and local processing the computation is fast. The prediction is correct for 35 examples fulfilling these conditions and incorrect for three of them (see the numbers in parentheses). For longer multimedia files and local execution of tasks 0-5 the processing is slow (line 24). For such files, cloud execution and HSPA+ the connection time for task number 0 is fast (line 51), while the time for the other tasks is medium (line 52). The model can be further simplified by applying discretization (with equal frequency) to all numerical attributes (task and file length as well). The learned decision tree is presented in Figure 8. The accuracy decreases to 77.3, but the number of nodes is much smaller (19). Therefore the entire classifier may be analyzed. In the first line, all processing of small files is classified as fast (180 examples are classified correctly and 22 incorrectly). The speed of processing medium files locally is medium (line 3). The speed of processing medium files in the cloud depends on the connection type and task number. For EDGE it is medium for tasks 0–7 (line 6) and slow for the others (lines 7–8). For HSPA, HSPA+ and LTE connections the processing is fast (lines 9–11). The local processing of large files is slow (line 13). The processing of such files in the cloud is slow in the case of the EDGE connection (line 15) or medium in the other cases (lines 16–18). Generally, the knowledge represented by the decision trees is quite intuitive.

We have also checked the complexity of the learning process by measuring how much time and battery it consumes. CPU energy consumption was measured by the Power Tutor package [34] integrated with our project. Three learning algorithms (C4.5, Random Forest and Naïve Bayes) were executed on the mobile device for 25,

```
fileLength <= 61128
| connectionType = EDGE
| | engine = mobile:  FAST (35/3)
| | engine = cloud
| | | task <= 5
| | | | task <= 0:  FAST (6/1)
| | | | task > 0:  SLOW (6)
| | | task > 5:  MEDIUM (13/4)
| connectionType = HSPA: FAST (22)
| connectionType = LTE: FAST (60/1)
| connectionType = HSPA+:  FAST (38/1)
fileLength > 61128
| fileLength <= 305530
| | engine = mobile:  MEDIUM (117/34)
| | engine = cloud
| | | connectionType = EDGE
| | | | task <= 5:  MEDIUM (11)
| | | | task > 5:  SLOW (10/1)
| | | connectionType = HSPA: FAST (12/2)
| | | connectionType = LTE: FAST (18/2)
| | | connectionType = HSPA+:  FAST (11/1)
| fileLength > 305530
| | engine = mobile
| | | task <= 5:  SLOW (80/5)
| | | task > 5
| | | | fileLength <= 458566
| | | | | connectionType = EDGE: SLOW (25/12)
| | | | | connectionType = HSPA: SLOW (8/2)
| | | | | connectionType = LTE: MEDIUM (18/8)
| | | | | connectionType = HSPA+:  MEDIUM (9/1)
| | | | fileLength > 458566:  SLOW (53/11)
| | engine = cloud
| | | connectionType = EDGE
| | | | task <= 0
| | | | | fileLength <= 458566:  MEDIUM (5)
| | | | | fileLength > 458566:  SLOW (8)
| | | | task > 0:  SLOW (39/1)
| | | connectionType = HSPA
| | | | task <= 0:  FAST (5)
| | | | task > 0:  MEDIUM (18/3)
| | | connectionType = LTE
| | | | task <= 0:  FAST (10)
| | | | task > 0
| | | | | fileLength <= 458566
| | | | | | task <= 10
| | | | | | | task <= 5:  MEDIUM (2)
| | | | | | | task > 5:  FAST (6/1)
| | | | | | task > 10:  MEDIUM (12/2)
| | | | | fileLength > 458566:  MEDIUM (27/2)
| | | connectionType = HSPA+
| | | | task <= 0:  FAST (6)
| | | | task > 0:  MEDIUM (27/6)
```

Figure 7. Classifier predicting the calculation time discretized into 3 ranges and learned by the C4.5 algorithm (the numbers of examples covered by every leaf with correct/incorrect predictions are stated in parentheses)

```
fileLength = small:  FAST (180/22)
fileLength = medium
| engine = mobile:  MEDIUM (117/34)
| engine = cloud
| | connectionType = EDGE
| | | task = 0-7:  MEDIUM (11)
| | | task = 8-20:  SLOW (7)
| | | task = 21-25:  SLOW (3/1)
| | connectionType = HSPA: FAST (12/2)
| | connectionType = LTE: FAST (18/2)
| | connectionType = HSPA+:  FAST (11/1)
fileLength = large
| engine = mobile:  SLOW (193/48)
| engine = cloud
| | connectionType = EDGE: SLOW (52/6)
| | connectionType = HSPA: MEDIUM (23/8)
| | connectionType = LTE: MEDIUM (57/19)
| | connectionType = HSPA+:  MEDIUM (33/12)
```

Figure 8. Classifier predicting the calculation time learned by the C4.5 algorithm, with all attributes discretized into 3 ranges (the numbers of examples covered by every leaf with correct/incorrect predictions are stated in parentheses)

50, 75 and 100 % of the training data collected in one of the previous experiments, which consisted of 720 examples. It was repeated 100 times. Average execution time and power consumption are presented in Tables 2, 3 and 4. The lowest learning time and energy consumption were observed for the Naïve Bayes algorithm and the highest values were for Random Forest. Values in both sets increase when more training examples are processed but remain acceptable in all cases. Learning time is well below one second with the only exception of the Random Forest, for which the average learning time is about 1.2 seconds. Energy consumption for all learning algorithms is very low (below 0.06 % of battery capacity).

Where experience is collected for a long time, multiple examples may be stored in the *Training Data* and as a result, the learning time and energy consumption may become excessive. In this case, the oldest examples can be removed to save time and energy.

Number	Time (ms)			Energy (% of the battery)		
of examples	min	max	average	min	max	average
181	60.8	137.6	77.71	0.0001	0.0008	0.0003
361	134.8	163.1	146.37	0.0006	0.0017	0.0010
541	198.4	212.6	206.2	0.0015	0.0022	0.0018
720	303.6	339.1	324.65	0.0030	0.0062	0.0045

Table 2. Execution time and energy consumption of the C4.5 machine learning algorithm executed on the mobile device

Number of examples	Time (ms)			Energy (% of the battery)		
	min	max	average	min	max	average
181	277.4	328.9	299.24	0.0024	0.0051	0.0036
361	554.1	598.7	576.65	0.0108	0.0170	0.0144
541	846.4	886.7	871.33	0.0262	0.0349	0.0312
720	1177.6	1229.4	1197.65	0.0509	0.0670	0.0592

Table 3. Execution time and energy consumption of the Random Forest machine learning algorithm executed on the mobile device

Number of examples	Time (ms)			Energy (% of the battery)		
	min	max	average	min	max	average
181	32.6	83.2	42.15	0.00006	0.00050	0.00018
361	57.7	67.3	64.03	0.00021	0.00039	0.00030
541	87	109.8	96.29	0.00032	0.00085	0.00052
720	119.8	174.6	137.38	0.00045	0.00198	0.00092

Table 4. Execution time and energy consumption of the Naïve Bayes machine learning algorithm executed on the mobile device

6 CONCLUSIONS

In this research, we have investigated recommender systems and proposed a novel solution, which includes a formal model of agent-based architecture using supervised learning designed for service recommendation in a heterogeneous environment. The experiments related to recommending video file processing services have demonstrated that the main objective of our studies has been achieved and after the learning process, the optimal service in the MCC environment is selected.

The results demonstrate a statistically significant decrease in the time required for the execution of conversion services owing to the automatic selection (using learning methods) of the location (mobile device or cloud) where the conversion of multimedia data is to be performed. The learning algorithm was executed by the learning agent autonomously and online on the mobile device. As a result, agent strategy was updated and recommendations were improved. We have also measured the time taken, and the energy consumed, by the learning process executed on the mobile device. The time is acceptable and energy consumption is very low.

Further research in this area could involve more extensive testing of both recommendation stages for different types of services, taking into account battery usage and context data from the wide range of sensors installed in mobile devices such as GPS modules, accelerometers and light sensors. We also plan to apply other knowledge representation techniques and learning algorithms, e.g. neural networks or support vector machines.

Acknowledgements

The research presented in this paper was supported by the Polish Ministry of Science and Higher Education under AGH University of Science and Technology Grant 11.11.230.124. We would like to thank Karol Bukowski for assistance with the implementation and testing.

REFERENCES

[1] Abbar, S.—Bouzeghoub, M.—Lopez, S.: Context-Aware Recommender Systems: A Service-Oriented Approach. Proceedings of the 3rd International Workshop on Personalized Access, Profile Management and Context Awareness in Databases (PersDB), in conjunction with the International Conference on Very Large Data Bases (VLDB), Lyon, France, 2009, pp. 1–6.

[2] Abidar, R.—Moummadi, K.—Medromi, H.: Mobile Device and Multi Agent Systems: An Implemented Platform of Real Time Data Communication and Synchronization. Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS), Ouarzazate, Morocco, 2011, pp. 1–6, doi:10.1109/ICMCS.2011.5945595.

[3] Anagnostopoulos, T.—Anagnostopoulos, C.—Hadjiefthymiades, S.: Mobility Prediction Based on Machine Learning. Proceedings of the 12th IEEE International Conference on Mobile Data Management, Lulea, Sweden, 2011, pp. 27–30, doi: 10.1109/MDM.2011.60.

[4] Angin, P.—Bhargava, B.: An Agent-Based Optimization Framework for Mobile-Cloud Computing. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), Vol. 4, 2013, No. 2, pp. 1–17.

[5] Bachara, P.—Brzoza-Woch, R.—Długopolski, J.—Nawrocki, P.—Zaborowski, W.—Zieliński, K.—Ruta, A.: Construction of Hardware Components for the Internet of Services. Computing and Informatics, Vol. 34, 2015, No. 4, pp. 911–940.

[6] Balakrishnan, P.—Tham, C. K.: Energy-Efficient Mapping and Scheduling of Task Interaction Graphs for Code Offloading in Mobile Cloud Computing. Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), Dresden, Germany, 2013, pp. 34–41, doi: 10.1109/UCC.2013.23.

[7] Bouneffouf, D.: Improving Adaptation of Ubiquitous Recommander Systems by Using Reinforcement Learning And Collaborative Filtering. Computing Research Repository (CoRR), 2013, arXiv:1303.2308.

[8] Braunhofer, M.—Elahi, M.—Ricci, F.: STS: A Context-Aware Mobile Recommender System for Places of Interest. In: Cantador, I., Chi, M. (Eds.): Proceedings of the 22nd Conference on User Modeling, Adaptation, and Personalization Co-Located with the 22nd Conference on User Modeling, Adaptation, and Personalization (UMAP) – Posters, Demos, Late-Breaking Results and Workshop, Aalborg, Denmark, 2014, pp. 75–80.

[9] Brzoza-Woch, R.—Długopolski, J.—Nawrocki, P.—Zieliński, K.: Application of FPGA Integrated Circuits for Acquisition and Providing Information in Compliance with Web Service Model. Przeglad Elektrotechniczny, Vol. 89, 2013, No. 7, pp. 340–347.

[10] Brzoza-Woch, R.—Czekierda, Ł.—Długopolski, J.—Nawrocki, P.—Psiuk, M.—Szydło, T.—Zaborowski, W.—Zieliński, K.—Żmuda, D.: Implementation, Deployment and Governance of SOA Adaptive Systems. In: Ambroszkiewicz, S., Brzeziński, J., Cellary, W., Grzech, A., Zieliński, K. (Eds.): Advanced SOA Tools and Applications, Springer Berlin/Heidelberg, 2014, pp. 261–323, doi:10.1007/978-3-642-38957-3_6.

[11] Brzoza-Woch, R.—Nawrocki, P.: FPGA-Based Web Services – Infinite Potential or a Road to Nowhere? IEEE Internet Computing, Vol. 20, 2016, No. 1, pp. 44–51, doi: 10.1109/MIC.2015.23.

[12] Colombo-Mendoza, L. O.—Valencia-García, R.—Rodríguez-González, A.—Alor-Hernández, G.—Samper-Zapater, J. J.: Recommetz: A Context-Aware Knowledge-Based Mobile Recommender System for Movie Showtimes. Expert Systems with Applications, Vol. 42, 2015, No. 3, pp. 1202–1222, doi:10.1016/j.eswa.2014.09.016.

[13] Czarnowski, I.—Jedrzejowicz, P.: Machine Learning and Multiagent Systems as Interrelated Technologies. In: Czarnowski, I., Jedrzejowicz, P., Kacprzyk, J. (Eds.): Agent-Based Optimization. Studies in Computational Intelligence, Springer Berlin/Heidelberg, Vol. 456, 2013, pp. 1–28, doi:10.1007/978-3-642-34097-0_1.

[14] Eom, H.—Figueiredo, R.—Cai, H.—Zhang, Y.—Huang, G.: MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training. Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), San Francisco, CA, USA, 2015, pp. 51-60, doi: 10.1109/MobileCloud.2015.19.

[15] Kosinski, J.—Nawrocki, P.—Radziszowski, D.—Zielinski, K.—Zielinski, S.—Przybylski, G.—Wnek, P.: SLA Monitoring and Management Framework for Telecommunication Services. In: Bi, J., Chin, K., Dini, C., Lehmann, L., Pheanis, D. C. (Eds.): Proceedings of the 4th International Conference on Networking and Services (ICNS), Gosier, Guadeloupe, 2008, IEEE Computer Society, pp. 170–175, doi:10.1109/ICNS.2008.31.

[16] Nawrocki, P.—Mamla, A.: Distributed Web Service Repository. Computer Science, Vol. 16, 2015, No. 1, pp. 55–73, doi:10.7494/csci.2015.16.1.55.

[17] Nawrocki, P.—Jakubowski, M.—Godzik, T.: Analysis of Notification Methods with Respect to Mobile System Characteristics. Proceedings of the Federated Conference on Computer Science and Information Systems, Lodz, Poland, 2015, pp. 1183–1189, doi: 10.15439/2015F6.

[18] Panait, L.—Luke, S.: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems, Vol. 11, 2005, No. 3, pp. 387–434.

[19] Resnick, P.—Varian, H. R.: Recommender Systems. Communications of the ACM, Vol. 40, 1997, No. 3, pp. 56–58, doi:10.1145/245108.245121.

[20] Ricci, F.: Mobile Recommender Systems. Information Technology and Tourism, Vol. 12, 2010, No. 3, pp. 205–231, doi:10.3727/109830511X12978702284390.

[21] Sankaranarayanan, S.—Cuffe, K.: Intelligent Agent Based Scheduling of Student Appointment-Android Environment. Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), Seoul, Korea, 2010, pp. 46–51, doi:10.1109/ICCIT.2010.5711027.

[22] Shabtai, A.—Elovici, Y.: Applying Behavioral Detection on Android-Based Devices. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (Eds.): Mobile Wireless Middleware, Operating Systems, and Applications. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin/Heidelberg, Vol. 48, 2010, pp. 235–249, doi:10.1007/978-3-642-17758-3_17.

[23] Shi, C.—Pandurangan, P.—Ni, K.—Yang, J.—Ammar, M.—Naik, M.—Zegura, E.: IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud. Technical Report GT-CS-13-01, Georgia Institute of Technology, USA, 2013.

[24] Singh, D.—Sardina, S.—Padgham, L.—Airiau, S.: Learning Context Conditions for BDI Plan Selection. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) – Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, Toronto, Canada, 2010, pp. 325–332.

[25] Śnieżyński, B.—Dajda, J.: Comparison of Strategy Learning Methods in Farmer-Pest Problem for Various Complexity Environments without Delays. Journal of Computational Science, Vol. 4, 2013, No. 3, pp. 144–151.

[26] Śnieżyński, B.: Agent Strategy Generation by Rule Induction. Computing and Informatics, Vol. 32, 2013, No. 5, pp. 1055–1078.

[27] Śnieżyński, B.: Comparison of Reinforcement and Supervised Learning Methods in Farmer-Pest Problem with Delayed Rewards. In: Badica, C., Nguyen, N. T., Brezovan, M. (Eds.): Computational Collective Intelligence. Lecture Notes in Computer Science, Springer Berlin/Heidelberg, Vol. 8083, 2013, pp. 399–408, doi:10.1007/978-3-642-40495-5_40.

[28] Śnieżyński, B.: Agent-Based Adaptation System for Service-Oriented Architectures Using Supervised Learning. Procedia Computer Science, Vol. 29, 2014, pp. 1057–1067

[29] Sun, Z.—Han, L.—Huang. W.—Wang, X.—Zeng, X.—Wang, M.—Yan, H.: Recommender Systems Based on Social Networks. Journal of Systems and Software, Vol. 99, 2015, pp. 109–119, doi:10.1016/j.jss.2014.09.019.

[30] Tripathy, A.—Patra, A.—Mohan, S.—Mahapatra, R.: Designing a Collaborative Filtering Recommender on the Single Chip Cloud Computer. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Salt Lake City, UT, USA, 2012, pp. 838–847, doi: 10.1109/SC.Companion.2012.118.

[31] Tuyls, K.—Weiss, G.: Multiagent Learning: Basics, Challenges, and Prospects. AI Magazine, Vol. 33, 2012, No. 3, pp. 41–52.

[32] Sensoy, M.—Vasconcelos, W. W.—Norman, T. J.—Sycara, K.: Reasoning Support for Flexible Task Resourcing. Expert Systems with Applications, Vol. 39, 2012, No. 2, pp. 1998–2010, doi:10.1016/j.eswa.2011.08.041.

[33] Witten, I. H.—Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[34] Zhang, L.—Tiwana, B.—Qian, Z.—Wang, Z.—Dick, R. P.—Mao, Z. M.—Yang, L.: Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS), ACM, New York, NY, USA, pp. 105–114, doi: 10.1145/1878961.1878982.

Piotr Nawrocki is Assistant Professor in the Department of Computer Science at the AGH University of Science and Technology, Poland. His research interests include distributed systems, computer networks, mobile systems, service-oriented architectures and mobile cloud computing. He received his Ph.D. degree in computer science from AGH University of Science and Technology in Krakow, Poland.

Bartłomiej Śnieżyński received his Ph.D. degree in computer science in 2004 from AGH University of Science and Technology in Krakow, Poland. In 2004 he worked as Postdoctoral Fellow under the supervision of Professor R. S. Michalski at the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA. Currently, he is Assistant Professor in the Department of Computer Science at AGH. His research interests include machine learning, multi-agent systems, and knowledge engineering.

Jakub Czyżewski is computer science graduate student at the AGH University of Science and Technology in Krakow, Poland. His fields of interests include web technologies, machine learning and cloud computing.