A Multi-agent Approach to Unstructured Data Analysis Based on Domain-specific Onthology [⋆]

Natalia O. Garanina, Elena A. Sidorova, and Evgeny V. Bodin

A.P. Ershov Institute of Informatics Systems,
Lavrent'ev av., 6, Novosibirsk 630090, Russia,
{garanina,lena,bodin}@iis.nsk.su

Abstract. The paper presents a multi-agent algorithm implementing semantic analysis of unstructured data based on ontology. In this multi-agent model agents of two kinds interact: the instance agents correspond to meaningful units of the information being retrieved, and the rule agents implement rules of a given ontology. An original solution for the termination detection of this multi-agent algorithm is suggested.

1 Introduction

At present most organizations deal with large quantity of documents, licenses, manuals, emails, business letters, financial and technical reports etc. It is already impossible to process all these documents by hand, without automatic assistance. Ontological knowledge bases are a good solution for storing information from these documents, and automatical completing the ontology is necessary.

The essence of an ontological approach to information retrieval is to use knowledge represented by an ontology for extraction of data interpreted as the ontology instances. For example, semantic-oriented approach to text analysis without complete linguistic analysis can be used for ontological data generation. The standard productional approach to semantic-oriented analysis is to sequentially apply given rules of instance retrieval to data. This process takes a long time and causes such specific problems as information duplication, variability of results, etc. Using the multi-agent approach allows to create good alternatives to the data analysis systems with the sequential architecture. The main feature of the approach is that the system being developed is considered to be a set of autonomous entities (the agents) where the agents have the abilities to interact with the environment and with other agents. By means of this interaction the system works. The traditional benefits of the multi-agent approach is that the operations of the system are parallelized, due to independent agents and their ability to interact with each other, so that some system tasks are solved locally and therefore the result is obtained significantly faster. Besides, our multi-agent approach handles some of the above difficulties of productional approaches.

[⋆] The research has been supported by Siberian Branch of Russian Academy of Science (Integration Grant n.15/10 "Mathematical and Methodological Aspects of Intellectual Information Systems").

The proposed approach is in the framework of modern investigations of automatic processing and analyzing huge amount of unstructured data. Multi-agent approach for information retrieval from heterogeneous data source for completing ontology is widespread, in particular, it is used for natural language processing [1, 2, 11, 6] and web processing [3–5]. Agents in these works have different behavior. Usually in web processing, agents are high-level agents that manage rather data flows, using standard algorithm for knowledge retrieval, than data itself. In natural language processing, agents are either associated with conventional linguistic levels (morphological, syntactic, semantic) or targeted to recognize specific linguistic phenomena such as ellipsis, anaphora, parataxis, homonymy. These agents do not use ontological knowledge substantially. Thus they are computing processes which may speed up information retrieval due to their parallel work but they do not affect the retrieval qualitatively.

Unlike all the above works, in our approach we use two kinds of agents, collectively possessing complete information about both the data being investigated and the domain-specific ontology. Agents of one kind can analyze ontological (and linguistic) features. They do not use data directly, but they process information provided by requesting agents of the other kind. The latter agents are the most close to the ones from [10]. In cited paper every agent representing some word from a text has to determine correspondence between its word and an element of a given ontology. The authors do not use special agents for ontological (and linguistic) properties. Instead, they exploit statistical methods of text clustering.

Our variant of ontology-based approach for processing unstructured data contains the following stages. First, a proper domain ontology has to be selected. We suppose that rules for completing the ontology are defined formally. Then initial ontology instances (their classes and some attributes) have to be identified by some preliminary algorithm. Then other instances' attributes are evaluated by the ontology rules using our algorithm.

The idea of multi-agent aspect of the our approach is that a set of different data items is aggregated into an agent considered as an ontology instance. This process is assisted by special support agents corresponding the ontology. First, objects significant for the ontology are recognized preliminary in given data. We call these objects *instance agents*. Belonging to an ontology, the instance agents have attributes. The values of some of these attributes are evaluated as the result of the preliminary analysis. Non-evaluated attributes can be specified as a result of communication of instance agents and the support *rule agents*. In the process of interaction, the agents establish a correspondence between the ontology concepts and the instance units, and thus complete the ontology with specific instances of concepts and relationships.

This paper presents a multi-agent algorithm for arbitrary unstructured data processing. This algorithm improves and generalizes the algorithm for information retrieval from natural language text suggested in [7]. We estimate the time upper bound of the algorithm and prove the properties of termination and correctness of the termination controller agent.

The rest of the paper is organized as follows. The next section 2 describes the main agents in our systems and gives a simple example. The following section 3 presents protocols for the instance, rule and controller agents and sketches some properties of the systems. Finally, we conclude in the last section 4 with a discussion of further research topics.

2 Agent Model

Outline of the approach and multi-agent system follows. There is an ontology, a set of rules for completing it, and a finite set of data to extract the information for the ontology. The preliminary phase partially assigns values to *instance* agents attributes. The *rule* agents implementing the ontology rules (the rules depend on the ontology only, but not on the data), according to data received from various instance agents, generate new attribute values of the instances, send the obtained result to all agents interested in it, or generate new instance agents. Eventually, the instance agents assign values to all their attributes that can be evaluated with the information from the data, and the system stops. A special *controller* agent keeps track of system stopping.

Let the result of data pre-processing be the set IA of instance agents, where each $I \in IA$ is a tuple $I = (id; Cl; RO_0; a_1(RI_1; RO_1), ..., a_k(RI_k; RO_k))$, where

- id is a unique agent identifier;
- Cl is an ontological class of the agent;
- RO_0 is set of rule agents that use this instance agent as an argument;
- for each $i \in [1..k]$, a_i is the attribute of the agent, which value is determined by some rule agent from RI_i, every rule in set of rule agents RO_i requires the value of attribute a_i to get the result; let us denote the set of rule agents for incoming values as $RI = \cup_{i=1..k} RI_i$, the set of rule agents for outcoming values as $RO = \cup_{i=1..k} RO_i$.

The values of attributes of an instance agent are usually only partially determined before the algorithm starts. When the algorithm terminates, the initially unvalued attributes should be provided values with help of rule agents.

Let us define the set of rule agents RA, where each $R \in RA$ is a tuple $R = (id; arg_1(Cl_1), \ldots, arg_s(Cl_s); make\_res(args), a_{res})$, where

- id is a unique agent identifier;
- for each $i \in [1..s]$: arg_i is a set of argument values determined by the corresponding instance agent from ontological class Cl_i; let us denote the set of vectors of argument values as $args$, where each value is provided with the identifier of the defining instance agent, the set of these agents is $args.Ag$; let us consider the argument values vector non-empty, if all its values are non-empty;
- $make\_res(args)$ is a function for computing the result from argument vector $args$;
- a_{res} is the result of function $make\_res(args)$ which can be

- null, if the argument vector is inconsistent;
- a new value of some attribute[1] for instance agents;
- a new instance agent (there should not be an agent with the same attribute values in the system).

As a simple example let us consider the following multi-agent system for natural language text processing. Let the given ontology includes classes $Person$, $Organization$, and relation $Employee$. The corresponding instance agents have the following form:

- $A_1 = (id; Person; \{CWork, CPersonDeg, \ldots\};$
  $surname(\{CPerson, CPersonIni\}; \emptyset), first\_name(\{CPerson\}; \emptyset),$
  $degree(\{CPersonDeg\}; \emptyset), ...; Employee(\{CWork, CWorkPos\}).$
  $Person$ has the following attributes: $surname$, $first$ $name$, (academic) $degree$ which can be used and evaluated by the corresponding rule agents $CPerson$, $CPersonIni$, $CPersonDeg$.
- $A_2 = (id; Organization; \{CWork, CPersonDeg, \ldots\};$
  $name(COrg; \emptyset), type(COrg, COrgType; CWork), \ldots;$
  $Employee(CWork, CWorkPos);$
  $Organization$ has attributes $name$ and $type$, and the corresponding input-output rule agents $COrg$ and $COrgType$.
- $A_3 = (id, Employee; \{CWork, CWorkPos\};$
  $arg_1(CWork; \emptyset), arg_2(CWork; \emptyset), pos(CWorkPos; \emptyset)).$
  Relation $Employee$ can be evaluated by rule agent $CWork$ directly connecting $Person$ and $Organization$, or by rule agent $CWorkPos$ which connects them using a position.

As an example of an ontology rule agent let us consider agent $CWork$:

$$CWork = (id, arg_1(A_1), arg_2(A_2);$$
$$\{Sentence(\{arg_1, arg_2\}), BracketSegment(\{arg_2\}),$$
$$Preposition(arg_1, arg_2), Contact(arg_1, arg_2)\};$$
$$Employee.arg_1 = CWork.arg_1, Employee.arg_2 = CWork.arg_2;$$
$$A3 = newEmployee()).$$

This agent recognizes sentences where an organization is enclosed in brackets after a person. For example:

> Academician Genrikh Aleksandrovich Tolstikov (Novosibirsk Institute of Organic Chemistry) is a prominent chemist, recognized authority in synthetic organic chemistry.

The following evaluation of attributes of the above agents is the result of analysis of the given text fragment:

$$A_1 = Person(id = 1, surname = Tolstikov, first\_name = Genrikh,$$
$$degree = Academician);$$
$$A_2 = Organization(id = 2, name = Institute\ of\ Organic\ Chemistry,$$

[1] This attribute is defined a priory.

$type = Institute$);
$A_3 = Employee(id = 3, arg_1 = A_1, arg_2 = A_2, pos = \emptyset)$.

In the next section, the algorithms of the instance, rule and controller agents are described in pseudocode.

3 Multi-agent Algorithm for Data Analysis

Let $IA = \{I_1, \dots, I_n, \dots\}$ be an instance agents set, and $RA = \{R_1, \dots, R_m\}$, be a rule agents set. The result of executing of the following algorithm is data analysis, when the instance agents determine the possible values of their attributes. Let `Ii` be a protocol of actions of instance agent I_i, and `Rj`, be the protocol of actions of rule agent R_j, `C` be the protocol of actions of an agent-controller C. Then the multi-agent data analysis algorithm MDA can be presented in pseudocode as follows:

```
MDA::
  begin
   parallel {I1} ...{In} ...{R1} ...{Rm} {C}
  end.
```

Here we assume that the `parallel` operator means that all execution flows (threads) in the set of braces are working in parallel. That is, all agents act in parallel until either all attributes of the instance agents are evaluated or it happens that none of the rule agent can proceed. These events are determined by the controller agent. The system is dynamic because rule agents can create new instance agents. Let N be the maximal number of instance agents that can be obtained from a given data.

The agents are connected by duplex channels. The controller agent is connected with all agents, and every instance agent is connected with several rule agents (and vise versa). Messages are transmitted asynchronously and stored in FIFO channels until being read. The messages are transmitted in a fast and reliable medium.

We consider an agent *active* iff it does not complete its work (is not at the label "`end`" of the algorithms below) and either it processes some message or its queue of input messages is not empty. Otherwise, the agent is *passive*. We say that a multi-agent system *terminates* iff every system agent (possibly) except the agent-controller is passive.

In the agent protocols below the function `get_head(queue)` removes the first element from the queue and returns that element.

3.1 Protocol of Instance Agents

Let us comment a notation of the instance agent protocol. A message for an instance agent has two fields: `name` of sender (`name` $\in [1..m] \cup C)$[2] and a_i with value of attribute i. The pseudocode of the protocol follows.

[2] The agent receives messages only from the controller agent and from the rule agents.

Protocol of instance agents.

```
I::
  $C$: Controller Agent;
  $R, R_{ij}$: Rule Agent;
  $RI$, $RO$: set of Rule Agents;
  data_wait: set of Rule Agents = $\emptyset$;
  $a_i$: Attribute;
  mess: message;
  $In$: queue of incoming messages;
begin
1.   send $|RI \cup RO| + 1$ to $C$;
2.   forall $R \in RI \cup RO$ send evaluated_data to $R$;
3.   forall $a_i \in Atr$ forall $R_{ij} \in RI_i$ {
4.      send request($a_i$) to $R_{ij}$; add $(ij)$ to data_wait;}
5.   send $-1$ to $C$;
6.   while (true){
7.    if $In \neq \emptyset$ then {
8.       mess = get_head($In$);
9.       if mess.name = $C$ then break;
10.      if mess.name $\in RI_i$ then {
11.        if $a_i = \emptyset$ then {
12.          upd($a_i$);
13.          forall $R_{ij} \in RI_i$ {
14.            send cancel($a_i$) to $R_{ij}$; remove $(ij)$ from data_wait;}
15.          forall $R_{ij} \in RO_i$
16.            send data($a_i$) to $R_{ij}$;
17.          send $|RO_i| - 1$ to $C$;}
18.          if $a_i \neq \emptyset$ then send $-1$ to $C$;} }
19.      if data_wait = $\emptyset$ then break;}
end.
```

Let us informally describe the protocol of an instance agent. First, the agent (1) notifies the controller agent that it started working and the number of rule agents that will process its data (line 1), (2) sends the available data (line 2), (3) sends the requests for evaluating and adds the corresponding rule agents to its waiting list (lines 3-4), and then (4) tells the controller agents that it is now passive. From the beginning of the work of the agent, its channel is open for incoming messages. As soon as a message arrives, it begins processing it (line 8). If it is from the controller agent, the agent terminates (line 9). If it is from a rule agent (line 10) then if the corresponding attribute is empty (line 11) the agent evaluates it with the obtained data (line 12) and notifies other rule agents related to this attribute that a value of this attribute is no more required from them, and then the instance agent deletes these agents from its waiting list (lines 13-14). Then the obtained attribute value is sent to those agents that require it (lines 15-16), and the controller agents is notified about the agent finishing its work and about the number of rule agents that will process sent attribute value (line 17). If the message contains the value of an attribute that is already evaluated, the agent does not handle it and notifies the controller agent about it (line 18). If it turns out that all attributes are evaluated, the agent finishes its work (line 19).

Let us estimate the time upper bound of the instance agent protocol. In the first phase of its activities (line 2) the instance agent sends the evaluated data to all rules agents interested in this data. The complexity of this phase $C_1^{IA} = O(|RI| + |RO|) = O(m)$. Sending the activation messages to rule agent from RI (lines 3-4) is estimated as $C_2^{IA} = O(k \times m)$ and the size of the queue of incoming messages In is the same. The agent processes the received data (lines 8-19). It takes time $C_4^{IA} = O(|In| \times (|RI| + |RO|)) = O(k \times m^2)$. Thus, the upper bound of the protocol actions of each instance agent is $C^{IA} = O(k \times m^2)$.

3.2 Protocol of Rule Agents

In the algorithm of the rule agent's actions protocol, the following functions and notation are used. The rule agents receive messages only from instance agents and from the controller agent. The messages have (1) the `name` of the sender, (2) the `type` $\in$ {`data, request, cancel`} that means that it has received an attribute, a request for a result, or a cancelation request, respectively, and (3) the *value* of the attribute. The function `make_arg`(a, I) creates vectors of arguments with received values of attributes at the positions corresponding to ontology classes. The function `make_res`$(args)$ creates the output result: (1) the values of attributes that have sent a request to the rule agent, (2) a new instance agent which starts working immediately, or (3) the null result in a case of inconsistency of the argument vector. The pseudocode of the protocol follows.

Protocol of rule agents.

```
R::
  $C$: Controller Agent;
  $I$: Instance Agent;
  $a$: Attribute;
  $args$: vector of Argument = $\emptyset$;
  $Arg$: queue of vector of Argument = $\emptyset$; // set of tuples
  res_send: set of Instance Agents = $\emptyset$;
  $In$: queue of incoming messages;
begin
1.  parallel
2.   { while (true) {
3.       if $In \neq \emptyset$ then {
4.         mess = get_head($In$);
5.         if mess.name=$C$ then goto end;
6.         if mess.name=$I$ then {
7.             if mess.type=request then add $I$ to res_send;
8.             if mess.type=cancel then remove $I$ from res_send;
9.             if mess.type=data then {
10.                $a$ = mess.val; Arg = Arg $\cup$ make_arg($a, I$);
11.              send $-1$ to $C$; }}}}}
12.  { while (true) {
13.      if Arg $\neq \emptyset$ then {
14.         send 1 to $C$;
15.         $args$ = get_head($Arg$);
16.         $a_{res}$ = make_res($args$);
17.         if $a_{res}$.type = attr then {
18.            forall $I \in args.Ag \cap res\_send${
19.               send $a_{res}$ to $I$; remove $I$ from res_send;}
20.            send $|args.Ag \cap res\_send| - 1$ to $C$;}
21.         if $a_{res}$.type = new_agent then send $a_{res}.val$ to $a_{res}$.name;
22.         if $a_{res}$.type = null then send $-1$ to $C$;} } }
end.
```

Let us informally describe the protocol of a rule agent. The agent can perform in parallel both processing of incoming messages(lines 2-10) and the generating of the outcome (lines 12-21). If it has received a message from the controller agent, it finishes the work (line 5). If the agent receives a request from the agent I for a result (line 7), it adds I to the recipients list; and it removes I from this list if it was the cancelation request (line 8). If it receives a value of the attribute a from the agent I, then using the procedure *make_arg* it tries to create a vector of arguments (set of vectors) (line 10). In such vectors the received value of the attribute is one of the elements and other elements are values of attributes received earlier. Then the agent tells the controller agent about becoming passive (line 11). If the vector (or the set of vectors) is formed, the agent immediately begins to process it/them (line 13). The result of processing is obtained using the function *make_res*. It may be (1) an attribute which is later sent to those agents that have requested it (lines 16-18), then the controller agent is informed about the number of the agents to process the data and about this agent has completed processing the vector of arguments, (2) a new instance agent, that starts working immediately as soon as it gets the attribute values from the rule agent, (3) no result, due to the vector of arguments is not consistent, and the controller agent is notified that the argument vector processing is finished.

Let us estimate the time upper bound of the rule agent's protocol. The time complexity depends on the time bounds of the parallel actions of the rule agents. Let Ag be the set of agents have sent attributes and Arg be the set of arguments of the rule agent. The complexity of requests and cancels is $C_2^{RA} = O(|Ag|) = O(N)$ (lines 7-8). Retrieving and storing data from the instance agents (lines 9-11) is a very time-consuming process with the estimate $C_1^{RA} = O(|Ag|^{|Arg|}) = O(N^s)$, since the obtained data generate a set of vectors of the argument values. The complexity of parallel data processing (lines 13-22) is $C_3^{RA} = O(C_1^{RA} \times (||make\_res|| + |Ag|) = O(N^s \times (s+N))$, where $||make\_res||$ is the time complexity of the function, which is linear with respect to the size of the argument. Thus, the overall time upper bound of the actions of each rule agent is $C^{RA} = O(C_3^{RA}) = O(N^s \times (s+N))$.

3.3 Protocol of the Controller Agent

A special agent-controller handles the Distributed Termination Detection problem [8]. We suggest an algorithm for the problem which fits to our multi-agent system more than known termination detection algorithms, the credit/recovery algorithms in particular [9, 12]. The main feature of this agent-controller is to sequentially calculate other agents' activities by using variable *Act*. Instance and rule agents send information about their activities to the agent-controller. After system termination the agent informs others about this fact.

Protocol of agent-controller C.

```
C ::
 Act, num: integer;
 messages: queue of integer;
begin
1.  Act = 0;
2.  while(true){
3.   if messages $\neq \emptyset$ then { num =get_head(messages); Act=Act+num;}
4.   if messages $= \emptyset$ and Act = 0 then break; }
5.  send STOP to all;
end.
```

Let us estimate the time upper bound of the controller agent's protocol. The size of the queue of incoming messages for the controller agent C^{CA} is less then $N + \sum_{i\in[1..N]} |k_i| + \sum_{j\in[1..m]}(N + N^s_j)$, where k_i is the number of attributes of instance agent i and s_j is the number of attributes of instance agent j.

3.4 MDA Protocol Properties

The time complexity of the multi-agent analysis algorithm MDA follows from the above estimations: $C^{MDA} = O(max\{C^{IA}_1, ..., C^{IA}_N, C^{RA}_1, ..., C^{RA}_m, C^{CA}\})$, where C^{IA}_i and C^{RA}_j are the complexities of the protocols of the instance and rule agents, respectively, for all $i \in [1..N]$, $j \in [1..m]$.

Correctness (completeness and soundness)[3] of information retrieval algorithms is rather a notion of data analysis theory than the theory of multi-agent algorithms, thus it is out of the scope of this paper. But this multi-agent algorithm has some properties to be proved.

Proposition 1. *Multi-agent system* **MDA** *terminates and the agent-controller determines the termination moment correctly.*

Sketch of the proof. First, an analyzed data contains a finite number of information objects for a given ontology. Hence the number of corresponding agents and their attributes is finite. Hence (1) every instance agent determines values of all its attributes and goes to a passive state or (2) some attributes can not be evaluated because there is no appropriate information in the data and after

[3] Completeness means that all relevant information has been retrieved from data. Soundness mans that this information has been retrieved correctly.

determining evaluable attributes an instance agents goes to a passive state also. Every rule agent (1) gets enough information from instance agents to process received data and goes to a passive state after that or (2) goes to a passive state after receiving messages and never processes data. After processing data, the generation of new instance agents does not duplicate agents. Hence, there is no infinite loop because the number of information objects in the data is finite.

Second statement of the proposition follows from the fact that the value of variable *Act* becomes 0 no earlier than the termination moment. Let $active(t)$ be the number of active agents. For every time moment t the following holds: $\sum_{i \leq |mess(t)|} mess(i,t) + Act = active(t)$, because agents influence (1) increase of *Act* when after their local termination they send to the controller the number of meaningful messages sent to instance/rule agents (lines 1,17/20), and (2) decrease of *Act* when they informs about their passive state (lines 5,18/11,22).■

4 Conclusion

The proposed approach aims at taking advantage of the agent-based approach to knowledge representation and processing. Thus, using the agent-based technology allows to avoid unnecessary information retrieval, since at any given time, only information required for an agent is being searched for. Furthermore, due to the agents working in parallel, the speed of data processing significantly increases.

Note that this paper presents only a basic formal model of agents' interaction that implements a simplified model of data analysis, which does not yet take into account specific problems related to ambiguity of input data. For example, let us consider a case of data ambiguity when the different ontology instance agents correspond to the same data ("toast" as "fried bread" and as "a tribute or proposal of health"). In order to handle such ambiguities competitive instance agents acquire points which characterize their connections with other instance agents. These connections are defined by agents' attributes that could be the *linked* agents themselves or their attributes. The more links some agent has and the more points its linked agents have, it becomes more probable that this agent is the most accurate data based instance of a given ontology.

These problems can be solved by increasing the expressive power of the proposed agent-based models by giving the agent the ability to work cooperatively, to compete (as above), to keep the history of its creation and development, etc.

Acknowledgments. We would like to thank Dr. I.S. Anureev for discussions.

References

1. Aref, M.M. *A Multi-Agent System for Natural Language Understanding* International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2003, 36

2. Ariadne Maria B.R. Carvalho, Daniel S. de Paiva, Jaime S. Sichman, João Luís T. da Silva, Raul S. Wazlawick & Vera Lúcia S. de Lima *Multi-Agent Systems for Natural Language Processing* In Francisco J. Garijo & Cristian Lemaitre (eds.), Multi Agent Systems Models Architecture and Appications, Proceedings of the II Iberoamerican Workshop on D.A.I. and M.A.S(Toledo, Spain, October 1-2 1998), pp. 61-69.
3. Banares-Alcantara R., Jimenez R., Aldea L. *Multi-agent systems for ontology-based information retrieval* // European Symposium on Computer-Aided Chemical Engineering-15 (ESCAPE-15),2005, Barcelona, Espaa
4. Cheng X., Xie Y., Yang T. *Study of Multi-Agent Information Retrieval Model in Semantic Web* // In Proc. of the 2008 International Workshop on Education Technology and Training and 2008 International Workshop on Geoscience and Remote Sensing (ETTANDGRS'08), 2008, Vol. 02, P. 636-639.
5. Clark K.L., Lazarou V.S. *A Multi-Agent System for Distributed Information Retrieval on the World Wide Web* // In Proc. of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises (WET-ICE '97), 1997, P. 87-93.
6. Danilo Fum, Giovanni Guida, Carlo Tasso *A Distributed Multi-Agent Architecture for Natural Language Processing* // In Proc. of the 12th conference on Computational linguistics (COLING '88), 1988, Vol. 2, P. 812-814.
7. Garanina N., Sidorova E., Zagorulko Yu. *Multi-agent algorithm of text analysis based on domain-specific onthology.* // Proc. of The 13th Russian Conference on Artificial Intelligence (CAI-2012), October 16-20, 2012, Belgorod, Vol.1, P. 219-226. (In Russian)
8. Matocha J., Camp T. *A taxonomy of distributed termination detection algorithms* // The Journal of Systems and Software, 1998, Vol. 43, P. 207-221
9. Mattern, F. *Global quiescence detection based on credit distribution and recovery* // Inform. Process. Lett. 30 (4), 1989, P. 195-200.
10. Minakov I., Rzevski G., Skobelev P., Volman S. *Creating Contract Templates for Car Insurance Using Multi-agent Based Text Understanding and Clustering*// In Proc. Holonic and Multi-Agent Systems for Manufacturing, Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2007, Regensburg, Germany, September 3-5, 2007. Springer, Lecture Notes in Computer Science, 2007, Vol. 4659, P. 361-370.
11. Cássia Trojhan dos Santos, Paulo Quaresma, Irene Rodrigues, Renata Vieira *A Multi-Agent Approach to Question Answering* // In Renata Vieira, Paulo Quaresma, Maria da Graça Volpes Nunes, Nuno J. Mamede, Cláudia Oliveira & Maria Carmelita Dias (eds.), Computational Processing of the Portuguese Language: 7th International Workshop, PROPOR 2006. Itatiaia, Brazil, May 2006 (PROPOR'2006) LNAI 3960, 13-17 de Maio de 2006, Berlin/Heidelberg: Springer Verlag, pp. 131-139.
12. Rokusawa, K., Iciyoshi, N., Chikayama, T., Nakashima, H. *An ecient termination detection and abortion algorithm for distributed processing systems* // In: Proceedings of the International Conference on Parallel Processing, pp. 18-22.
13. Zagorulko Yu.A., Sidorova E.A. *Document analysis technology in information systems for supporting research and commercial activities* // Optoelectronics, Instrumentation and Data Processing, 2009. Volume 45, Number 6. -pp. 520-525.