Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 5, pp. 852-861

Agent Technology in Monitoring Systems

B. Pătruţ, C. Tomozei

Bogdan Pătruţ, Cosmin Tomozei
"Vasile Alecsandri" University of Bacău,
Romania, 600115, Calea Mărăşeşti, 157
E-mail: bogdan@edusoft.ro, cosmin.tomozei@ub.ro

Abstract: The aim of this paper is to make a brief presentation of the results obtained by the authors regarding agent technology application in distributed monitoring systems development. Consequently, we would like to present MAgeLan and ContTest as monitoring systems based on intelligent agents technology. Formal aspects regarding intelligent agents will be mentioned. Quantitative issues concerning efficiency, maintenance and reengineering are also to be taken into account.
Keywords: intelligent agents, distributed systems; monitoring systems; hyper-encyclopedia, reengineering.

1 Intelligent agents in distributed computing

Software applications are nowadays in position and necessity to solve some problematical situations by their own, in order to save time and reduce cost. Intelligent agent integration in distributed computing proved to be an important way of achieving this objective in the last years. Furthermore, the union distributed computing with intelligent agents theoretical perspective offers a good practical basis to distributed artificial intelligence [2]. Intelligent agents have to be reliable, robust in order to offer accuracy in the results, in dissimilar, open or unpredictable environments [7], [8]. Software agents are situated in particular environments and capable of autonomous actions, in order to fulfill their objectives. The concept of autonomy and the fact that intelligent agents are enriched with it presume that human action is minimized. In [8] it is affirmed that the agent is an entity which can perceive the environment by sensors and acts in order to realize its objectives by effectors. On the other hand, distribution of hardware, software and data offers the possibility for the agents to be replicated on diverse nodes on the computer network.

2 Developing Students' Metacognitive Competences

Metacognitive skills development is an important formative intellectual object in education of the students, as reaching this level involves a route through effective education, appropriate to each one in particular [7]. Metacognitive skills suppose that students are aware of their own cognitive activity, i.e. learning activity, and self-adjustment mechanisms consisting in cognitive controls (rules, procedures, strategies).

Next we will introduce the notions of environment, agent, s-agent, necessary to develop some intelligent systems as those described in chapter 5 of this article. The following theoretical concepts are different from the FIPA standard [9] ones or from the concepts defined by authors like Wooldridge in [8], and are necessary for the implementation of the systems MAgeLan and ContTest.

The following statements are to introduce theoretical concepts regarding intelligent agents and distributed technology.

Definition 1. We use the name of environment for a set of elements $E = \{e_0, e_1, e_2, e_3, ..., e_n\}$ among which there is a relation of partial order marked with "$<$". We use the notation $e \leqslant f$ for the fact that $e < f$ or $e = f$, respectively $f > e$ if $e < f$.

The environment can be, at a certain point, in a certain state e, which we will express by $st(E) = e$. At first, the environment leaves an initial state e_0, for which $e_0, e_1, \forall i \in \{1, 2, ...n\}$. The state e_n is called final state for which it is considered that $e_i, e_n, \forall i \in \{1, 2, ...n\}$.

Definition 2. We use the name of **agent** for a triple of the type (3.1) $A = (S, s_0, R)$ where S is a finite set of states, s_0 in S is called the agent's initial state, and R is a set of evolution rules.

If agent A is in state s, then we express this by $st(A) = s$. Among the states of S there is a special state marked with λ. At first $st(A) = s_0$, and when $st(A) = \lambda$ we say that the agent is inactive. For the rest, the agent is active. The rules in R are of the type (1) or (2):

$$r_1 = (A, s, e) \rightarrow (A, t, f) \tag{2.1}$$

$$r_2 = (B, s, e) \rightarrow (B, t, f) \tag{2.2}$$

Rule (1) states that if $st(A) = s$, $st(E) = e$, then $st(A)$ becomes t and $st(E)$ becomes *f*, if $e \leqslant f$. The second rule (2) states that agent A ceases its implementation ($st(A)$ becomes λ), transferring the control to agent B, for which $st(B)$ becomes t, and the environment remains in the same state.

If $st(E) = e$ and there are two agents A and B with $st(A) = a \neq \lambda$ and $st(B) = b \neq \lambda$ and $(A, s, e) \rightarrow (C, z, f)$ and $(B, t, e) \rightarrow (D, z', f')$, then we will consider $st(E) = max(f, f')$ if f and f' are comparable, one of them respectively, if f and f' are not comparable, and $st(A) = 0$ and $st(C) = z$ if $f < f'$, respectively $st(B) = 0$ and $st(D) = z'$, if $f < f'$.

This can be generalized for more active agents.

Definition 3. We use the name of **s-agent** for an n-uple of the type (3)

$$S = (C, A_1, A_2, ..., A_n) \tag{2.3}$$

where C is an agent called **coordinating agent**, and $A_1, A_2, ..., A_n$ are agents corresponding to the definition above that are called **effectors** or **atomic agents**. The coordinating agent will interact directly with the user and the architecture and its functionality depends on the concrete implementation (the examples will be offered in the following chapters).

Because inside an s-agent, the agents transfer the control form one to another, an s-agent behaves like a communicative multi-agent system.

Definition 4. Let there be $S = (C, A_1, A_2, ..., A_n)$ an s-agent. If $s_1, s_2, ..., s_n$ are the states of the atomic agents that make up S (without the coordinator C), we then say that $(s_1, s_2, ..., s_n)$ is the state of S (at a given moment).

Definition 5. Let there be $S = (C, A_1, A_2, ..., A_n)$ an s-agent in the environment E, that cannot be modified by the user. If the initial state of S is of the type $(0, \lambda, \lambda, ..., \lambda)$ we say that S is a normal s-agent (s_{01} marks the initial state of the agent A_1).

Directly, we obtain the following lemma:

Lemma 1. In a normal s-agent, $\{s_1, s_2, ..., s_n\} = \{s, \lambda, \lambda, ..., \lambda\}$ is enacted, with $s \neq \lambda$, at any point of time t.

Proof. The s-agent being normal, it follows that its initial state (at the moment t_0) is $(0, \lambda, \lambda, ..., \lambda)$, therefore $\{s_{01}, s_{02}, ..., s_{0n}\} = \{s_{01}, \lambda, \lambda, ..., \lambda\}$, and $s_{01} \neq \lambda$.

Let us suppose that the state in the moment t_i is $\{s, \lambda, \lambda, ..., \lambda\}$, with $s \neq \lambda$. This means that an active A_i agent exists and that it is unique, with and $st(A_i) = s \neq \lambda$ and $st(A_j) = \lambda, \forall j \neq i$. If there is a relation of evolution of the type $(A_i, s, e) \rightarrow (A_j t, f)$, then by applying this relation of evolution, we will obtain in the moment $t_{i+1} : st(A_i) = \lambda$ and $st(A_j) = \lambda$, with $t \neq \lambda$. Therefore the state of the s-agent will become (no matter the situation) $(\lambda, \lambda, ..., \lambda, t, \lambda, ...\lambda)$, with $t \neq \lambda$, on the position j, therefore. If there is no relation of evolution with (A_i, s, e) on the left side, then the s-agent will remain in the state $\{s, \lambda, \lambda, ..., \lambda\}$, with $s \neq \lambda$.

The fact that the agent will remain in that certain state will be called **blockage**, a notion that we will eventually formally define.

Definition 6. We use the name of **multi-agent monitoring system (MAMS) of the environment** E, based on s-agents (or on groups) for a triple of the type (4)

$$MAMS = (Sa, L, E) \tag{2.4}$$

where Sa is a set of s-agents, having the same structure, E is the environment within which these exist and are implemented, and L are communication or linkage rules of the type $C_i \rightarrow C_j$, where C_i and C_j are coordinating agents of some s-agents from Sa.

The communication relations among the s-agents form an oriented graph depending on the architecture and the concrete implementation of the multi-agent system, as we will see in the following chapters. Our interest lies in some evolution rules of the environment within an s-agent and the structure and graphic representation of an s-agent.

A MAMS containing only normal s-agents is called a **normal MAMS**.

Definition 7. The purpose of an s-agent is to take the environment E to a state as close as possible to its final e_n state. If the relation (5) can be obtained, we then say that the aim of the s-agent can be carried out.

$$St(E) = e \wedge \neg \exists f \in E, f \neq e_n : e < f \tag{2.5}$$

By extension, we can say that the aim of MAMS is reached if all the targets of the component s-agents are achieved.

Definition 8. Two rules of evolution of the type $r_1 = a \rightarrow b$ and $r_2 = b \rightarrow c$, where a, b, and c are triples of the type of those in (1) or (2), they are called adjacent.

Definition 9. Let there be $r_1, r_2, ..., r_k$ a line of adjacent rules of evolution, two by two. We will use the notation (6) and we will call this relation as the derivation relation (see (6)).

$$r_1 \Rightarrow r_k \tag{2.6}$$

Within an s-agent, the following results are obvious:

Proposition 1. If st(E)=e, there is an agent A with st(A)=$s \neq \lambda$, (A, s, e) $\Rightarrow$ (A', s', e') and there is no $f \neq e_n$, so that $e' < f$, then the aim of the s-agent can be reached (with the environment in state e'). (From the very beginning, we have noted the final state of the environment with e_n). Most of the times, set E is not fully ordered. If, however, a fully ordered relation "<" is found, then the following sentence is enacted.

Proposition 2. If the relation "<" is fully ordered, then the purpose of the s-agent can be reached with the environment in the final state (e_n) if there is a derivation of the type $(A, s_0^A, e_0) \Rightarrow (B, t, e_n)$. Proposition 2 affirms that the purpose of the s-agent can be reached if there is a derivation which leads the environment to the state en, starting from the environment's initial state e_0 and the initial state of any agent (A), without the user's intervention (in the case of a fully ordered relation "<").

If $|\{A; st(A) \neq \lambda\}| = 1$, for any $e = st(E)$ (therefore a single agent is active at a given moment, as in lemma 1), then the MAMS operates sequentially. This is what happens most of the times.

We consider that the environment E modifies its current state in two cases:

• as a result of applying an evolution rule, by one of the system's agents;

• as a result of the direct intervention of a human user. We can also consider, in some exceptional cases, that the state a certain agent is in can be modified by the evolution rules as by the user as well. Therefore, we cannot hold control over what is going to happen, or how the state of the environment is going to evolve within a certain interval of time. If, ideally, the human user cannot randomly modify neither the environment E nor the current state of the agents, then the system is entirely deterministic.

3 Blockages and Infinite Cycles

Our interest does not lie simply in building absolutely sequential systems or deterministic systems, but in specifying in the best possible way the evolution rules so that blockages cannot occur.

Definition 10. If relation (7) is certified, then we say that the s-agent is under **blockage**.

$$\exists A \in S : st(A) = s, st(E) = e, e \neq e_n \wedge \neg\exists(A, s, e') \rightarrow (B, s', f), f, e' \neq e_n \tag{3.1}$$

Therefore, we say that an s-agent is under blockage when an atomic agent of the system gets into a state s, and the environment is in a non-final state e, and there is no evolution relation that can allow for the agent's passing out from the state s, although the environment is suddenly modified by the user, into another non-final state e'. In (7) f and e' are certain non-final states of the environment (e' and f may also be even e), and s' a certain state of a certain agent B from the s-agent, being even possible for B to be A.

In other words, there is no evolution rule, with s on the left side, which can lead to another state of A or within another agent B, no matter the evolution of the environment E.

Definition 11. In an s-agent a derivation of the type $(A, s, e) \Rightarrow (A, s, e)$ is called **infinite cycle**.

This occurs when, if the user does not intervene through modifying the environment, the execution of the s-agent's agents cycles infinitely, without the environment reaching the final state. In this case, the user may intervene to take the MAMS out of the cycle.

Blockages and infinite cycles can be identified easier if we represent the s-agents and the MMS. Graphically, a multi-agent monitoring system (MAMS) can be represented in the shape of a graph oriented thus (figure 1):

• optionally, more s-agents are represented in the shape of some polygons or other geometrical figures, containing more s-agents, the internal structure being represented only for one of them;

• optionally, the links among the s-agents will have the shape of some curves; graphically only one s-agent will be represented, because all s-agents are considered to have the same internal structure;

• the generic s-agent will be represented through a geometrical figure where all atomic agents are represented;

• the atomic agents are represented through some rectangles labeled with their names; inside those squares there will be circles representing the different states of the respective agent;

• each state will represent a point in an oriented graph, where the edges are the evolution relations, labeled with a pair of clues for the states of the environment: the starting state and the state that is finally reached.

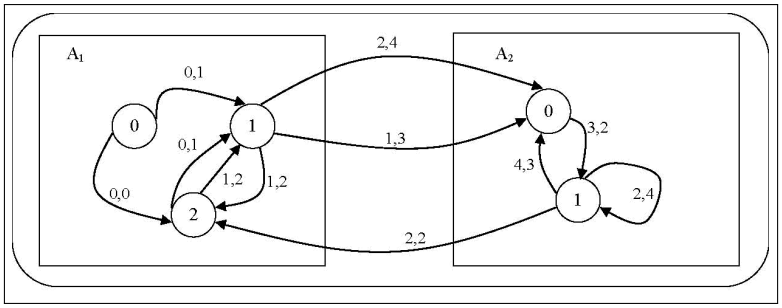

Figure 1: A simple s-agent containing a blockage in A_1 and an infinite cycle in A_2

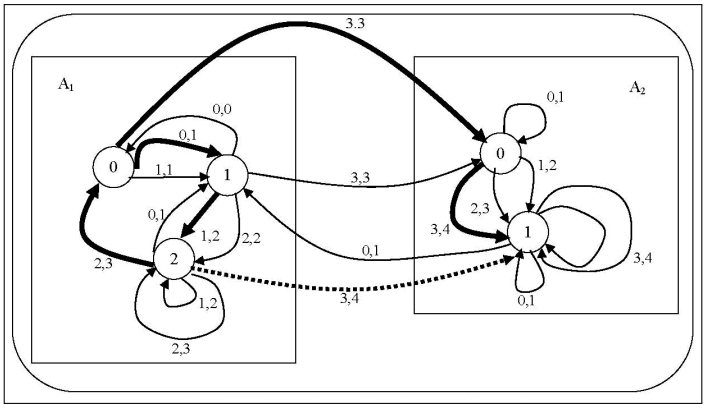

Figure 2: A normal s-agent and its reduced s-agent

This is an ideal example of MAMS functioning, illustrated by the bold arcs in figure 2. Obviously, the other arcs except the bold ones are useless in this graph, because they will never be passed through. If, however, the user interferes, for example the moment the MAMS is in state 2 from the atomic agent A_1, modifying the state of the environment from e_2 to e_3, then the evolution $(A_1, 2, e_3) \rightarrow (A_2, 1, e_4)$ will occur, expressed in figure 3 through the dotted arc. We can obviously "endow" an s-agent with many evolution rules. Inside a co-operating working environment, different users will introduce different rules of evolution. The question is, if under the circumstances of some normal agents, some of those rules will ever be applicable, will ever come into action. It is as if we had a program with functions or pieces of codes which are not resorted to by anywhere, cannot be touched, or an expert system with production rules whose part of the premises will never be fulfilled.

Thus, we will show that normal agents can be reduced to other normal agents, on the basis of an algorithm, so that certain evolution rules could become useless and could be eliminated from the system, the later behavior being not affected. On the contrary, the systems become more simple. By an s-agent' behavior we mean the applicability of the rules of its component agents. Therefore, if a certain evolution rule could ever be applied, it will be a part of the s-agent's

behavior, and if not, then it will not be a part of this behavior.

Definition 12. Let there be S_1 and S_2 two normal s-agents, functioning simultaneously within the same environment E. If the behavior of S_1 is identical with the one of S_2 at all moments, meaning that the same evolution rule is applied both in S_1 and S_2, we say that the two agents are **equivalents**. Equivalence also refers to the situations of blockages or infinite cycles.

4 Reducing the Normal S-agents

We will further consider a normal s-agent. We will show that the following theorem is enacted.

Theorem 1. A normal s-agent S can be reduced to a normal s-agent T that is equivalent with S, by eliminating some evolution relations, according to the following algorithm.

Reduction algorithm(s-agent S, s-agent T);
{Local Variables: A, s, e, A_i, blockage, obj_realized, M, $T = \emptyset$;
STEP 1: for each A_i *from S*;
if $st(A_i) \neq \lambda$ *then* $(A, s, e) \leftarrow (Ai, st(A_i), st(E))$;
end for;
STEP 2: $M = \emptyset$; *cycle = False; blockage = False; obj_ realized = False*;
while (not cycle) and (not obj_ realized) and (not blockage);
{*M = M* $\cup$ *(A, s, e)*;
if in S exists edge (A, s)$\rightarrow$ *(B, t) labeled with (e, f) then* {;
Add T in Vertex s in agent A; Vertex t in agent B;
Add edge (A, s) $\rightarrow$ *(B, t) between s and t; Label edge (A, s)* $\rightarrow$ *(B, t) with (e, f)*;
if f = e_n *then obj_ realized = True else*;
if $\neg \ni f' > f$ *and* $f' \neq e_n$ *then obj_ realized = True*;
else if (B, t, f) $\in$ *M then cycle=True* };
else blockage=True } }.

Proof. According to lemma 1, the s-agent S has always a state made up of n-1 inactive states (λ) and a single active state belonging to one single agent. This means that the I^{st} part in an algorithm will determine a unique agent A with a singular active state s, and e will be the current state of the environment. The cycle "while" from the second part ends because inside it all possible cases are brought into discussion on the "if-then-else" branches. Building the s-agent T is done using the instructions in the framed part. We have used the graph representation of S and T. The instructions inside the dotted frame realize the inclusion in T of the agents A and B (unless they are already there), of their states s and t (unless they are already there) and of the relation of evolution $(A, s, e) \rightarrow (B, t, f)$.

The algorithm will extract out of the graph of S a chain T, until the final state has been reached, the "largest" final state, a blockage will occur or an infinite cycle will be entered into. T will thus be a normal s-agent, because if, at a given moment, according to lemma 1, $\{s_1, s_2, ..., s_n\} = \{t_1, t_2, ..., t_n\}$ and $s_1, s_2, ..., s_n = \{s, \lambda, \lambda, ..., \lambda\}$ is enacted, it also results that $\{t_1, t_2, ..., t_n\} = \{s, \lambda, \lambda, ..., \lambda\}$.

Observation 1. The previous result is equivalent, in the theory of formal and automatic languages, with the elimination of inaccessible and unusable states from a finite deterministic automate [1].

Corollary 1. An s-agent S gets blocked if and only if its reduced agent T gets blocked.

Corollary 3.2. An s-agent S has an infinite cycle if and only if its reduced agent T has an infinite cycle.

According to the algorithm in theorem 1, the s-agent T is equivalent to the s-agent S, therefore the implementations of S and T are identical. If S gets blocked, then T gets blocked and reciprocally. If T has an infinite cycle, then S has the same cycle too (though it may have some others as well).

The algorithm in theorem 1 can be completed with a sequence of pseudo code, in order to display the configuration in which both S and T get blocked or enter an infinite cycle.

5 Practical Implementations

One first example of MAMS is that where the environment is given by an intelligent hyper-encyclopedia. We described in [4] the concept of intelligent hyper-encyclopedia, which, intuitively means an encyclopedia distributive developed by several users and which uses artificial intelligence tools. For the time being, we only mention the fact that the environment E is made up of the content of the hyper-encyclopedia. By the content of hyper-encyclopedia, we understand the totality of its entries. Each entry is defined by a word, a definition of this and the references to the other related entries. The states of the environment are states of the content of this hyper-encyclopedia at a certain moment, that is the totality of the entries introduced by the respective moment. We certainly have an initial state e_0 when the hyper-encyclopedia contains nothing, but also a final state e_n, when the hyper-encyclopedia is considered by the users to be definite, complete. The order relation "<" between the states of the hyper-encyclopedia is a relation of partial order, because we can say that one state is smaller than the other only if the contents of the encyclopedia at a certain moment is included in the contents of the encyclopedia at another (later) moment. However, we cannot compare two states in which the hyper-encyclopedia has completely different contents, or has only one common part. The aim of the MAMS is to develop the hyper-encyclopedia as much as possible.

The s-agents will be local multi-agent systems. Each user who contributes to the enlargement of the hyper-encyclopedia has his/her own s-agent. In [3], [4] we presented a real architecture of such an s-agent. Within each s-agent there is a series of atomic agents bearing different tasks. We presented the agents responsible for monitoring the Microsoft Word editing activity of the content of the hyper-encyclopedia, while we have dealt with the agents that monitor the web search by using the Internet Explorer browser.

The encyclopedia behaves intelligently in these directions: it raises certain questions in order to generate new information, based on the existing ones; it answers certain questions from the users in order to provide them with the information necessary to generate new knowledge; it adapts itself to the users' language and to their way of working (consults and modifies the databases or interacting with the others; it provides an intelligent mediation in the communication among users, so that these may be able to learn from each other, and the encyclopedia from them. The intelligent hyper-encyclopedia - having so many "human" traits, even in its limited alternative - will have a network structure where each knot will detain a local data basis, a human user and a system of intelligent agents, thus: an agent which observes the user's behavior while consulting the encyclopedia and adapts itself to it; an agent which observes the user's behavior while updating the local data basis (modifying, deleting, adding data or generating rules) and acts accordingly; an agent which alone modifies the local data basis in order to adapt it to the user from that knot (in order to ease his/her work). All the above mentioned agents communicate among themselves and cooperate to learn from each other and to modify their own local data bases, but also the way in which they interact with the users from the knots. Communication among agents and human users will often be achieved by natural language, through certain

conversational agents. They are also capable of "metamorphosis' in order to adapt themselves to the users, or, in case this cannot be done, agents can move from one knot to another. The MAgeLan [9] operates to solve tasks characteristic of artificial trainers which assist (monitor, help and manage) several students (users) in developing an intelligent hyper-encyclopedia.

One rule of evolution of the type $(A, s, e) \rightarrow (A, t, f)$ can be thus interpreted: if the hyper-encyclopedia is in state e, and the agent A notices that the user searches the web for the term s, then the hyper-encyclopedia passes to the state f (its content is improved by creating some new links between the articles or by some other means) and the user is suggested to search for the term t. Obviously, the user may or may not follow the system's piece of advice.

A second example is given by an instructive system, ContTest [5]. The environment is represented by the knowledge of the learner. In this case, first of all we assume that the student knows nothing, therefore the environment lies in the state e_0. When the student has achieved what we had had in mind the environment is in its final state. The aim of the MAMS is to make the student know everything, or at least as much as possible.

If we assume that the s-agents are multi-agent systems, each of them responsible for one lesson or one learning unit, then each s-agent will contain atomic agents which will have specific pedagogical tasks. For example, an agent will be responsible for teaching some content, another will be responsible for offering examples, another one for generating tests. The states of an s-agent are stages in teaching, illustration, testing and so on. Let us consider the agents as being didactic actions. For example, agent A can mark the teaching, and agent B, the testing.

One rule of evolution of the type $(A, s, e) \rightarrow (A, t, f)$ will communicate to us something like this: if we have taught (A) the notion s, and the student's level is e, then we are to teach the notion t next and the student will reach level f. One rule of evolution of the type $(A, s, e) \rightarrow (B, t, f)$ will communicate to us that if we have taught the notion s and the student's level is e, we can then pass on to testing (agent B) the student through exercise t and we expect the student to reach the (superior) level f.

6 Reengineering MMS Systems

Multi-agent monitoring systems are being submitted to the process of reengineering in order to achieve new objectives. Software reengineering proves to be [6] valuable in large-scale application development. By applying this procedure, we will keep a valuable software system, such as MAgeLan hyper-encyclopedia in working condition for a longer period of time. The cost is significantly reduced and the new tasks are efficiently accomplished.

If we take into consideration the application's functionalities, the following formula (8) will describe the elements regarding the connection of the modules that remain unchanged. Elements which are to be modified or deleted will be identified as well, for the entire application to become a powerfully connected system.

$$Ir = \bigcup_{i=1}^{n} Funct_{i0} + \bigcup_{i=1}^{n} Funct_{i1} - \bigcup_{i=1}^{n} DelFunct_i \tag{6.1}$$

Ir represents the indicator of reengineering, $Funct_{i0}$ is the functionality in the initial moment, $Funct_{i1}$ is the functionality in the present moment added due to reengineering, consequently $DelFunct_i$ is the functionality eliminated by reengineering, and n represents the number of functionalities.

ContTest (figure 3) significantly evolved due to reengineering, the agents being enriched with new teaching and learning methods and procedures. Therefore, evolution in objectives brought about evolution in the functionalities of agents. Pedagogical tasks of agents have been

considerably improved. Consequently, higher educational objectives and elevated results in the evaluation process have been obtained.

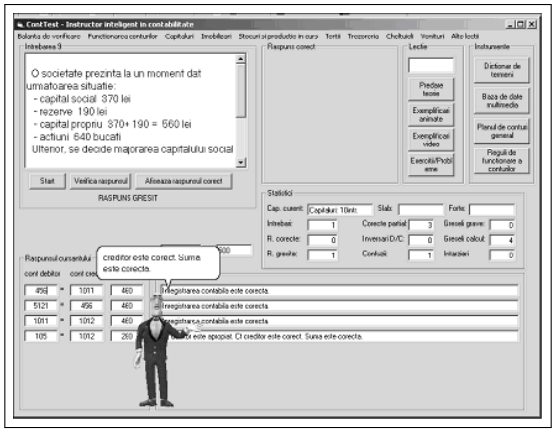

Figure 3: Validation of answers

We may state that each software project has been enhanced with a higher level of quality, evolving correspondingly with the evolution in demands, by integrating reengineering.

7 Conclusions

Our aim was to summarize the results obtained in development and maintenance of distributed applications and multi-agent monitoring systems. Practical examples, such as MAgeLan and ContTest have been given, in order to implement and to test the theoretical approaches.

Multi-agent monitoring systems bring quality and efficiency in almost every area of activity, including scientific research, education, healthcare or defense.

By implementing the solid theoretical concepts of artificial intelligence and reuniting them to the infinite resources of distributed systems and the Internet, we get the possibility to increase the quality of the results and reduce the duration of reaching the objectives.

Reengineering maintains the multi-agent systems and expands their utility in time and functionality. Consequently, software which proved to be precious in the activity continues to be efficient in service for a longer period of time.

Bibliography

[1] Atanasiu, A., 2007, *Formal languages and automata*, InfoData Publishing House, Cluj-Napoca, Romania

[2] Dzitac, I., Bărbat, B.E., Artificial Intelligence + Distributed Systems = Agents, *Int. J. of Computers, Communications & Control*, ISSN 1841-9836, E-ISSN 1841-9844, 4(1):17-26, 2009

[3] Pătruţ, B., Pandele, I., 2008, How to Compute the References Emergencies in a Hyper-encyclopedya, in "Recent Advances in Systems Engineering and Applied Mathematics. Selected papers from the WSEAS conferences in Istanbul, May 27-30, 2008", ISBN 978-960-6766-91-6, ISSN 1790-2769, Istanbul, Turkey, pp.72-75

[4] Pătruţ, B., Socaciu, T., 2008, *Constructing a Hyper-enciclopedya: How to Transfer the Emergencies Between the Nodes*, in Proceedings of the Fourth International Bulgarian-Greek Conference (Computer Science' 2008), 18-19 September 2008, Kavala, Greece, pp. 468-473

[5] Pătruţ, B., Vârlan S. E., Socaciu, T., 2008, *ContTest - a Multiagent System for Accounting Education*, in Proceedings of the Third International Multi-Conference on Computing in the Global Information Technology, ICCGI 2008, July 27 - August 1, 2008, Athens, Greece

[6] Tomozei, C., 2009, *Security Engineering and Reengineering on Windows 2008 Server Based Distributed Systems*, Journal of Information Technology & Communication Security, SECITC 2009, Bucharest, pp. 63-73

[7] Vlassis, N., 2007, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Synthesis Lectures on Artificial Intelligence and Machine Learning*, Department of Production Engineering and Management, Technical University of Crete, Morgan and Claypool Publishers

[8] Weiss, G. (ed.), 1999, *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, MIT Press, ISBN 978-0-262-23203-6

[9] FIPA Standards Specifications: http://www.fipa.org/specifications/index.html