

# AN OVERVIEW OF AGENT BASED PARADIGMS AND ENTERPRISE SIMULATION FOR E-LEARNING AND KNOWLEDGE TRANSMISSION

Marco Remondino  
University of Turin – Department of Computer Science  
remond@di.unito.it

## ABSTRACT

Agent Based Modeling is an approach for simulating a complex system: in a social context, the single parts and the whole are often very hard to describe in detail; by using intelligent agents as basic building blocks, there are formalisms which allow to study the emergency of social behavior through the creation of models, known as "artificial societies". The paper deals with an hybrid agent based methodology, using cognitive and reactive agents, discussing their application to simulation models and games to be used for e-learning and knowledge transmission. In particular, a management game is briefly presented and the integration with software agents is discussed.

In the last part, a model is presented of a Data Base Management System, that can be used to teach some concepts about DB Security, using reactive agents.

## INTRODUCTION

The paper deals with the development of methodologies for designing models of interactive simulation, based on software (artificial) agents, to be used for education and in general for knowledge transmission through *e-learning* techniques. The term e-learning refers to the use of computers and related technologies, such as networks, multimedia and so on, for educational purposes. A model (or *artifact*) is built, representing some real situation in which the learners can take decisions and see the corresponding results. As an example, a management game can be considered, in which some decisions must be taken month by month (e.g.: how many components to buy, how many products to build, how to manage warehouses, which markets to choose, and so on). After taking these decisions, the system simulates a month with the given values and gives a response through some tables and graphs. This kind of systems should represent the real world in the best possible way, so that learners could practice on them (learning by doing).

The agent based paradigms have three main roles in this work. The agents can act as an interactive decision support system for the users, thus enhancing their learning experience, by directly experimenting on the model. Besides the agents could interpret in a cognitive way the learner's behavior and her/his "style" of learning, in order to interpret the actions on the basis of cause/effect relations

and to consequently "explain" that to the learner herself, later (virtual tutoring). Lastly, from a structural point of view, the agents can constitute some parts of the model itself, in order to make it self adaptive when facing unforeseen decisions and contexts.

In the first case, the agents must be endowed with a knowledge about the model itself; this knowledge arises from a period of training, in which the agents themselves operate on the model and learn its response. The main topics of the present work are the formalisms behind the software agents and the problem of action selection. Two are the paradigms employed, at different levels: the reactive agents feature a very simple structure, derived from the *stimulus-reaction* (S-R) paradigm. On the other side, the cognitive agents have a symbolic and explicit representation of the environment, basing on which they can reason and foresee future events.

## TWO AGENT BASED PARADIGMS

The term agent, deriving from the Latin *agens*, identifies someone (or something) who acts; the same word can also be used to define a mean through which some action is made or caused. In the studies on software agents have been divided into two main strands: the first one, starting in 1977, is based on the studies on Distributed Artificial Intelligence and gave origin to the cognitive agents, endowed with inner symbolic models. The second one, whose origins are to be found in the 90s, focuses on a wide range of agents, defined as reactive. They do not have any internal representation of their environment and the emphasis not on the way in which they decide what to do, but simply on when to act and what action to choose, basing on the stimuli from the environment. Reactive agents simply retrieve pre-set behaviors similar to reflexes without maintaining any internal state.

Both paradigms, though quite different, have some peculiar features that make them suitable for some given situations; the main problem with a purely cognitive agent, when dealing with real-time systems, is reaction time. For simple, well known situations, reasoning may not be required at all. In some real-time domains, minimizing the latency between changes in world state and reactions is important and so reactive agents can be successfully employed. On the other end, cognitive agents should be used when artificial learning or reasoning is concerned

## Reactive (sub-symbolic) Agents

This kind of agents may be regarded as any simple (not structured) software entities which interact among them and with the environment. A multi-agent context of this kind allows the emergency of complex behavior and self-organization, with no definition of a formal rule a priori. Apparent intelligent behavior is a product of the interaction among agents and environment, and of the interaction among many simple behaviors. It can be really hard to describe the real world under every aspect: some fundamental macro-actions can thus be defined on single agents, which allow cooperation with the environment and with other agents. The concept of Multi Agent System for Social Simulations is thus introduced: the single agents have a very simple structure. Only few details and actions are described for the entities: the behavior of the whole system is a consequence of those of the single agents, but it's not necessarily the sum of them. This can bring to unpredictable results, when the simulated system is studied.

In some situations, effective results can be obtained just by building simple, sub-symbolic agents, whose behavior is randomly determined or is built by applying fixed pre defined reaction rules; this is the case, for instance, of *Heatbugs*, one of the canonical Swarm demonstrations ([www.swarm.org](http://www.swarm.org)):

*"It's an example of how simple agents acting only on local information can produce complex global behavior. As we read on Swarm main site, each agent in this model is a heatbug. The world has a spatial property, heat, which diffuses and evaporates over time. In this picture, green dots represent heatbugs, brighter red represents warmer spots of the world. Each heatbug puts out a small amount of heat, and also has a certain ideal temperature it wants to be. The system itself is a simple time stepped model: each time step, the heatbug looks moves to a nearby spot that will make it happier and then puts out a bit of heat. One heatbug by itself can't be warm enough, so over time they tend to cluster together for warmth"*

## Cognitive (symbolic) Agents

These agents' behavior is goal-directed and reasons-based; i.e. is intentional action. The agent bases its goal-adoption, its preferences and decisions, and its actions on its Beliefs. In we read that a software cognitive agent should feature the following properties:

- *autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- *social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language;

- *reactivity*: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative." The Wooldridge and Jennings definition, in addition to spelling out autonomy, sensing and acting, allows for a broad, but finite, range of environments. They further add a communications requirement.

In general, cognitive agents have some sort of behavioral pattern, that can be described through modal logic, equations, evolutionary algorithms and so forth. In order to make the agents able to learn and improve themselves, some reinforcement algorithms can be embedded into their structure.

## ACTION SELECTION AND REINFORCEMENT LEARNING

The action selection problem at time  $t+1$ , along with the goal selection, at a macro (aggregate) level, are central topics, when the agents must "learn" how the models works, by experimenting on it and being able to act as a decision support system for human users.

For action selection we do not simply mean the problem of choosing which action to take at a micro level (agent level), but also which one, among the possible goals, to select. The first level of detail is typical for reactive agents; in fact they don't have any goals, if not those imposed by the external environment. The cognitive agents feature both levels of detail. In this work, the action selection problem is crucial for the formal definition of the involved agents, especially when they are employed as a decision support system, thus requiring some learning ability.

The goals could be combined to form higher level objectives or, on the contrary, be incompatible among them. In order to decide which actions to take, it's necessary to evaluate the utility for each of them; specific *Reinforcement Learning* (RL) algorithms are used for this purpose. These transform quantitative data (the *payoff*) in behavioural patterns for the agents.

An agent endowed with some RL algorithm, when in a particular state of the world ( $x$ ), makes an action ( $a$ ) and gets a payoff ( $r$ ), calculated by a reward function based on the consequences of the action itself. Through a *trial & error* mechanism the agent learns what are the actions that maximise this numerical value. An effective RL algorithm is the one called *Q-learning* [5]: an agent "lives" in a world modelled as a *Markovian Decision Process* (MDP), that's a set of states  $X$ , in which some actions from the set  $A$  can be done. For a state  $x$ , belonging to  $X$ , and an action  $a$ , be-

longing to  $A$ , there exist a probability function  $P_{xa}(y)$ , determining the transaction probability towards a new state  $y$ . At the same time, for each couple of possible  $x$  and  $a$ , there exist another probability function  $P_{xa}(r)$ , determining the payoff – or reward –  $r$ , generated by the action. We obviously have that:

$$\sum_y P_{xa}(y) = 1 \quad (1)$$

And that:

$$\sum_r P_{xa}(r) = 1 \quad (2)$$

The Q-learning algorithm builds the so called Q-values,  $Q(x,a)$ , by considering not only the payoff of a singular action, but also the expected discounted sum of future payoffs obtained by taking action  $a$  from state  $x$  and following an optimal policy thereafter. So we have that in each time-step  $t$ :

$$R = r(t) + \lambda * r(t+1) + \lambda^2 * r(t+2) + \dots \quad (3)$$

where  $\lambda \leq 1$  is the discount factor.

By defining policy ( $\pi$ ) a set of action rules, given a state, we have that by following that policy, the total discounted reward at time  $t$  equals the previous formula with  $E(r)$  instead of  $r$ , that's its expected value defined as:

$$E(r) = \sum_y r(x,a) * P_{xa}(y) \quad (4)$$

Besides acting as a decision support system for human learners experiencing with the model, artificial agents can be used to supervise the decision taken by learners, in order to interpret them in a cognitive way. For example, in the previously mentioned enterprise accounting model, some users could immediately pursue an high profit, while others could be concerned first with the expansion of their enterprise on new markets. Others could choose to improve industrial plants, while others could want to differentiate production and invest on research & development and marketing. All these decisions are complex, since they are determined by the combination of many different variables. Sometimes the learners won't even realize that they are pursuing a strategy instead of another one, and they often won't foresee what the selected strategy could bring. That's where the *motivational model* is employed; this is based on a function called wellbeing, where the intensity of the motivation to take a certain decision comes from the

combination of two factors: internal drive and in a limited way, some external stimuli. We have that:

$$M_i = D_i + w_i \quad (5)$$

In order to give more relevance to the first term, an activation threshold can be used, such as:

$$\begin{aligned} \text{if } D_i \leq L_i &\Rightarrow M_i = 0 \\ \text{if } D_i > L_i &\Rightarrow M_i = D_i + w_i \end{aligned} \quad (6)$$

The wellbeing variable is calculated as the difference between the highest possible value and the sum of the motivational drives, weighted by a factor  $\alpha$ , which represent in a cognitive way the personality of the human agent (the weight that the learner gives to the single decisions). We have that:

$$WB = WB_{\max} - \sum_i \alpha_i * D_i \quad (7)$$

The agents can also constitute some parts of the model itself; in the considered enterprise accounting model, some reactive agents can form the supply chain, or the warehouses, or even the competitors operating on the same market.

## EVOLUTIONARY ALGORITHMS FOR REACTIVE AGENTS

When dealing with reactive agents, the action selection problem is to be found at a macro (aggregate) level, i.e.: population level. If reactive agents are the competitors of human learners in the simulated world, they could have a fixed rule of behavior over time. Some evolutionary algorithms could be embedded in the agents, so that the best players on the market could merge, to form some other artificial players with an even better behavior.

In this way it's possible to start with a population of agents with a random behavior, facing the standard decisions in the model, and select – through the various “generations” – the best ones. So it's not the single agent that selects his behavior by updating its own policy (that remains the same, being the agent a reactive one), but the population that evolves over time, through the mechanism of reproduction and mutation.

This is an approach often used when the rules of the environment are given and the main task is to observe some emerging aggregate behavior arising from simple entities, i.e.: reactive agents. Since these agents does not feature a goal based – pro-active – behavior, the way they act tends to be deeply dependent on the choices made by the designer. In order to design flexible systems, the aggregate

behavior (at population level, i.e.: macro level) can be made self-adaptive through the implementation of an evolutionary algorithm (EA). In this case the agents will have a wired random behavior at the beginning, and evolve according to the environment in which they act, through a selection mechanism.

EA derive from observations of biological evolution. Genetic Algorithms (GA) are inspired by Darwin's theory of evolution, often explained as "survival of the fittest": individuals are modelled as strings of binary digits and are the encode for the solution to some problem. The first generation of individuals is often created randomly, and then some fitness rules are given (i.e. better solutions for a particular problem), in order to select the fittest entities. The selected ones will survive, while the others will be killed; during the next step, a crossover between some of the fittest entities occurs, thus creating new individuals, directly derived from the best ones of the previous generation. Again, the fitness check is operated, thus selecting the ones that give better solutions to the given problem, and so on. In order to insert a random variable in the genetic paradigm, that's something crucial in the real world, a probability of mutation is given; this means that from one generation to the next one, one or more bits of some strings can change randomly. This creates totally new individuals, thus not leaving us only with the direct derivatives of the very first generation. GA have proven to be effective problem solvers, especially for multi-parameter function optimization, when a near optimum result is enough and the real optimum is not needed.

Classifier Systems (CS) derive directly from GA, in the sense that they use strings of characters to encode rules for conditions and consequent actions to be performed. The system has a collection of agents, called classifiers, that through training evolve to work together and solve difficult, open-ended problems. They were introduced in [7] and successfully applied, with some variations from the initial specifics, to many different situations. The goal is to map if-then rules to binary strings, and then use techniques derived from the studies about GA to evolve them. Depending on the results obtained by performing the action corresponding to a given rule, this receives a reward that can increase its fitness. In this way, the rules which are not applicable to the context or not useful (i.e. produce bad results) tend to lose fitness and are eventually discarded, while the good ones live and merge, producing new sets of rules.

## ENTERPRISE SIMULATION AND EDUCATIONAL DECISION SYSTEMS

An existing simulation framework is described in this paragraph; this will be the one to which the agent based paradigms described before will be applied, in order to obtain a virtual tutorship and a decision support system for

the learners (the users). The simulation framework (<http://e-lab.di.unito.it/SimulazioneAziendale>), developed by prof. Gianpiero Bussolin and Marco Remondino at the University of Turin, Department of Computer Science, has been conceived as a teaching platform, used for transmitting such concepts as "double-entry accounting", and the way in which the decisions taken in a real enterprise affect the synthetic results, at the end of each period (month).

In this model, the users have to take a number of core decisions at the beginning of every month; the system, based on Forrester's System Dynamics, generates a set of reports, typical for Management and Enterprise analysis. The users, by reading these reports, can track down the influence of the single decision – or even better the aggregate effects coming from two or more decisions – on the synthetic results, representing the monthly performance of the whole enterprise.

		(Range)
Lotto di acquisto materie prime (C)	150.00	(0 ~ 5000)
Lotto da mettere in produzione (C)	150.00	(0 ~ 5000)
Investimenti in impianti (euro)	32000.00	(0 ~ 500000)
Investimenti in ricerca e sviluppo (euro)	400.00	(0 ~ 5000)
Assunzioni mano d'opera diretta (p)	1.00	(0 ~ 250)
Licenziamenti mano d'opera diretta (p)	1.00	(0 ~ 250)
Prezzo unitario mercato nazionale (euro/pf)	566.00	(7 ~ 700)
Durata media del credito concesso ai clienti nazionali (gg)	30.00	(30 ~ 360)
Interesse annuo concesso ai clienti nazionali (%)	10.00	(1 ~ 20)
Promozione mercato nazionale (euro)	32.00	(0 ~ 5000)
Prezzo unitario mercato estero (euro/pf)	1000.00	(10 ~ 1000)
Durata media del credito concesso ai clienti estero (gg)	30.00	(30 ~ 360)
Interesse annuo concesso ai clienti estero (%)	10.00	(1 ~ 20)
Promozione mercato estero (euro)	32.00	(0 ~ 5000)
Tempo o dilazione richiesta per i pagamenti di fornitori (gg)	30.00	(30 ~ 240)
Prestiti richiesti alle banche (euro)	1.00	(0 ~ 500000)
Estinzione prestiti bancari (euro)	1.00	(0 ~ 500000)

Figure 1: a form for monthly decisions

Agents can have many roles in such a system; first of all, reactive agents can be used as a part of the system, in order to simulate customers or suppliers. These agents should be very simple, just reacting to some market curve. On the other hand, reactive agents could also be the production implants, with the possibility of being programmed by the users in some way, and then adapting themselves to the number of pieces to be produced, and so on. This kind of interactivity would make the model more realistic.

Cognitive agents may have different and more important roles in this kind of models used for e-learning. After a training period on the model itself, using the reinforcement learning methods discussed above, an agent can compute some strategies to be used to make profit in the simulation. That said, this agent could then be used both as a decision support system for human users – since it could foresee some results, based on its acquired experience – and as a virtual tutor, explaining the relations among certain variables (decisions) and the achieved results. This could help the learners to understand the cause/effect links.

## A DBMS MODEL USING REACTIVE AGENTS

A Data Base Management System (DBMS) is defined as a software package, designed to store and manage databases, which are very large, integrated collections of data. A DBMS allows to reach the following objectives:

- Data independence and efficient access
- Reduced application development time
- Data integrity and security
- Uniform data administration
- Concurrent access, recovery from crashes

It is then obvious that one of the fundamental goals of a DBMS is to reach a security level which could prevent users with no specific grants to read data. It's also very important to reach a satisfying level for data integrity, by preventing users without an authorization to modify them. On the other side, it's necessary to reach an high efficiency for data retrieval, when the users have the specific rights. A security policy applied to a database must specify who is authorized to do what, and a security mechanism allows to enforce a chosen security policy. There are two main mechanisms at the DBMS level: Discretionary Access Control (DAC) and Mandatory Access Control (MAC).

The former is based on the concept of access rights or privileges for objects (i.e. tables and views), and mechanisms for giving and revoking users privileges; in this model, the creator of a table or a view automatically gets all privileges on it. The DBMS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges, at the time the request is issued, are allowed. The fundamental command, in this paradigm, is GRANT:

*GRANT privileges ON object TO users [WITH GRANT OPTION]*

In this way, the specified users get the privileges on the object belonging to the DB; usually, the privileges are the following ones:

- SELECT: Can read all columns (including those added later via ALTER TABLE command).
- INSERT(col-name): Can insert tuples with non-null or non-default values in this column. Similarly, UPDATE.
- INSERT means same right with respect to all columns.
- DELETE: Can delete tuples.
- REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.

If a user has a privilege with the GRANT OPTION, he can pass it on to other users, in turn with or without passing also the GRANT OPTION. Privileges can of course be lost, through the REVOKE command; if a user loses his privileges on an object, also the ones who had them from him will lose them. A user can receive the same privileges from different subjects and, in this case, he would lose them only if all these users lose those privileges on the object.

While in SQL-92, privileges are assigned to authorization ids, which can denote a single user or a group of users, in SQL:1999, and in many current systems, privileges are assigned to roles, that can then be granted to users and to other roles. This approach reflects how real organizations work and illustrates how standards often catch up with de facto standards embodied in popular systems.

Differently from the model described above, the MAC is based on system-wide policies that cannot be changed by individual users. Each object in the database is assigned a security class and each subject, user or user program, is assigned a specific clearance for a security class. The rules based on security classes and clearances govern who can read or write which objects. The MAC was born to overcome a typical flaw of the discretionary system, known as Trojan Horse. In fact, user A could create a table, on which he has all the privileges, and then can grant to user B the INSERT privileges on it. User B has privileges on, and thus can access, another table, containing secret data, which are forbidden to user A; then, user A modifies the code of an application program used by user B to additionally write those secret data to the newly created table, and so user A can now access these secret data. Bell-LaPadula model defines the main rules for the management of MAC. In this model we find:

- Objects (e.g., tables, views, tuples)
- Subjects (e.g., users, user programs)
- Security classes: Top secret (TS), secret (S), confidential (C), unclassified (U)
- An order for the classes: TS > S > C > U
- Each object and subject is assigned a class:
  - Subject S can read object O only if class(S) class(O) (Simple Security Property)
  - Subject S can write object O only if class(S) class(O) (\*-Property)

The main idea is to ensure that information can never flow from a higher to a lower security level. This, obviously, avoids the Trojan Horse problem. The MAC rules are usually applied in addition to any discretionary controls that are in effect.

### Model Description

In the following description of the model, we will represent a database organized according to the DAC, which is simpler and easier to port to a programming language. The use of an Object Oriented Language (OO) is assumed; this kind of languages (C++, Java) allows the creation of many independent objects, without having to write specific code for each of them. Besides, in an OO language, there are proprieties such as inheritance and polymorphism, useful for this model.

We can think of a set of agents, which are the users of a database, organized into a hierarchy; in general, a community of agents which can access data according to specific rules. When the single agent needs a datum, he first looks for it and, if he can't access it directly, he asks other agents, who have the specific permission on it. Hierarchy and proximity relations are defined among the agents: there are n levels, organized into a pyramid. Level 1 is the upper one, while Level n is the bottom. Besides, on the same level, proximity relations can be defined: a list could also exist, called Project Colleagues, containing the IDs of the agents working on the same project, so that if one of those creates a table or an object, the other agents will automatically have the access to them. Data inside the database are modelled as very simple objects, which have: a unique number, so that they can be called and retrieved; an identifier, signalling if the datum is corrupt or fine; the ID of their creator. Besides, there is a variable, associated with the datum, which signals whom accessed it for the last time; this is used to keep track of whom damaged it, if the datum is not fine anymore. Each user agent has his own unique ID, representing the name of the subject; a number, identifying the level to which he belongs and a list of the privileges on data; each element of the list is an array with the following elements:

- 1) datum number (code)
- 2) read privilege (yes/no)
- 3) ID of the subject that granted the privilege at point 2
- 4) write privilege (yes/no)
- 5) ID of the subject that granted the privilege at point 4
- 6) delete privilege (yes/no)
- 7) ID of the subject that granted the privilege at point 6

Besides, each agent has another list, containing the privileges he granted to others. Again, each element of the list is an array, with these elements:

- 1) datum number (code)
- 2) IDs of the subjects to whom read permission has been granted
- 3) IDs of the subjects to whom write permission has been granted
- 4) IDs of the subjects to whom delete permission has been granted

Obviously, an agent can't grant a privilege on a datum if he hasn't got it himself. When a subject creates an object, an array is automatically inserted in his list, with the new datum number (code) and all the privileges on it. Besides, Colleagues List described above could be implemented and in this case, when an object is created, all the subjects in this list will automatically have the privileges on it. Each user has also an unreliability index, which is increased each time he damages data, after a write operation. When an agent must complete an operation on a certain datum, he tries to access it directly, by looking in its list if he has the needed privileges on it. If he has the privileges, he access the datum in a single time unit, t. This process is shown in Figure 2: the user B, belonging to the third level of the pyramidal hierarchy, wants to access datum\_8, in the central database. When he contacts the DBMS (1), the datum attributes (2a) are compared to the list of the privileges of the user (2b). If the user has the needed privileges, he can immediately access the datum (3), and the operation is finished in a time t.

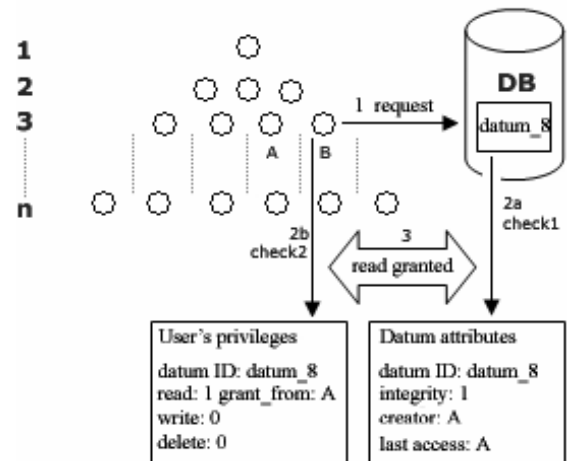


Figure 2 – access attempt, with privileges

If he hasn't got the needed privileges, he asks the datum who its creator is; this requires a time equal to that of retrieving the datum directly, that is  $t$ ; as a reply, the creator ID is returned. According to the adopted security policy, which in the simulation can be changed by the user, the agent will ask for the grant directly to the creator or, in the most inflexible case, he will have to move up in the hierarchy, asking for the grant to an user at the lever which is immediately upper, and so on, till when he meets one that has the needed privileges. In the worst case he will need to go all the way up to the creator; obviously, each request will consume some time, which can be considered equal to  $t/2$ . When the user meets an agent with the required privileges, he asks him to pass them to himself, consuming again a time  $t/2$ .

According to the inflexibility of the security policy, selectable before the simulation starts, and according to the unreliability index of the subject, the privileges will or won't be granted. If they are granted, the user will be able to access the datum, in a time  $t$ ; again, according to the security policy, the privileges can be kept by the user or can be immediately revoked after the operation has been completed. If the privileges are not granted, the user which owns them will access the datum on behalf of the requesting agent in a time  $2t$ . Of course, if the same user has to access the datum again, he will have to pass again through all these steps. In Figure 3, we show an example of the described case: user B must access datum\_8 (1), but after verification (2a and 2b), he realizes he can't do that directly. The datum then returns the ID of its creator, that is user A (3), who is on the same level as B can thus be contacted directly (4) by B.

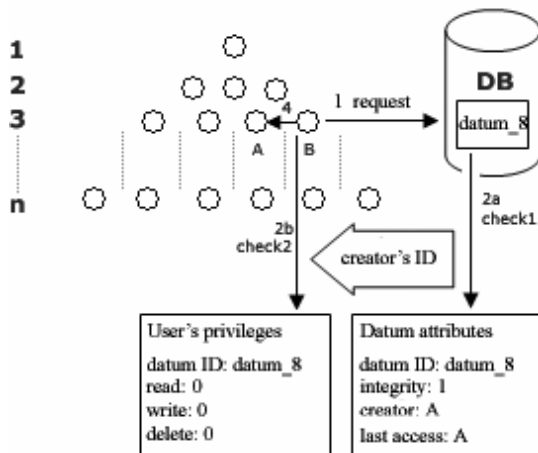


Figure 3 – access attempt, without privileges

With some simple calculations, we see that in the most inflexible situation, if the agent accessing the datum is not its creator, the access time, for each operation, is:

$$3t + t \left( \frac{\Delta + 1}{2} \right) \quad [8]$$

where  $\Delta$  is the distance between the level of the user requesting privileges and the creator of the object that is being accessed. This time must be compared to time  $3t$ , needed for data access in the most flexible situation, in which privileges are always granted. When an agent has the write privilege on a datum, there is a probability function which determines the possibility that this operation compromises its integrity. During the write operation, the user could modify the flag variable of the datum from 1 to 0.

The next time this datum will be necessary, the problem will emerge: this will increase the index of system unreliability and will waste a time  $2t$ , to restore the datum. The user who damaged it, whose ID is stored in the Last Access variable of the datum, will have its personal unreliability index increased and, according to the security policy adopted, will lose or not the privileges on that datum. When a user loses his privileges on a certain datum, also the agents who had the privileges from him will lose them. It could be possible that an agent had received the same privileges from more than one user: in this case, he will keep the privileges, unless all the granting agents lose them. The need of the users to access data is controlled by random functions, and so it the probability function for data corruption.

This probability increases with the growth of the delta between the level of the creator and that of the user accessing the data. During the execution of the simulation, two real-time graphs will be created: one will represent the average time for data retrieval; the other one will represent the general unreliability index of the system, derived from the average of corrupted data. By varying the security policy, through the initial parameters, it will be possible to compare different situations, after the same number of simulation steps and with the same random seed. The security policy affects the probability for an agent to grant the privileges to another user, on certain data. A probability equal to 0 means that no agent will receive the privileges, so that only the owners can access the data they created. In this case, the general unreliability index will be very low, but the time for data retrieval will reasonably be very high. A probability equal to 1 means that the privileges are always granted, no matter who asks for them: of course this will bring to an opposite situation. The intermediate cases, i.e. a probability between 0 and 1, are the most interesting and difficult to predict, but also the most useful to model real situations.

Also the creation of new data is managed in a random way, and at the beginning of the simulation some steps will be dedicated to this activity. We can think of a number of data with an inverse proportion in respect of the level or, in alternative, we can put in the simulation the exact situation that we observe in the organization we want to model.

The last case that needs to be considered is the request of privileges on certain data by an user who is at an upper level than their creator; there could be an automatic grant or, if we want to be more realistic, the request could be addressed directly to the creator, without needing to move down in the hierarchy, but using the rules defined in the security policy, that consider the personal unreliability of the requesting agent.

## CONCLUSION

In the paper an e-learning technique is discussed, using different kinds of agents as a support for learners that use a simulator for practicing some real situation. Two agent based paradigms are discussed, namely cognitive and reactive ones, and the action selection problem for both of them is studied in detail. The purpose is that of having cognitive agents acting the role of support decision system in the model, as long as that of virtual tutors for the learners, indicating both the possible outcome of the taken decisions, and helping to determine the global strategy selected by the human user. A management game is introduced, which is used to teach some concepts of Management and Economics.

On the other end, reactive agents are employed as part of the model itself, where it's important to have a real-time response from the system. Reactive agents can also be used as artificial competitors for human learners, in environments like markets or enterprise modelling. As an example of reactive agents used as parts of the model itself, a framework is described, which could be used to teach the different security policies for a Data Base Management System (DBMS).

## ACKNOWLEDGMENTS

I would like to gratefully acknowledge the key support of *prof. Gianpiero Bussolin*, who designed the Enterprise Simulator used as the starting point for many ideas described in this paper. I would also like to thank e-business LAB and in particular *prof. Marco Pironti* (University of Turin, Italy) for his precious support. I'd also like to acknowledge *the managerial board of Fontazione CRT*. Last but not least a big thanks to *Stefano Gabutti*, for his precious help and ongoing support.

## REFERENCES

- Singh, H., 2003: Building Effective Blended Learning Programs, in Issue of Educational Technology, Volume 43, Number 6, Pages 51-54.
- Simon, H. A., 1996: The Sciences of the Artificial, (third ed.). Cambridge, MA, MIT Press
- Nwana, H.S., 1996: Software Agents: an Overview, in Knowledge Engineering Review, Vol. 11, N. 3, pp.1-40, Cambridge University Press.
- Sutton, R. S., 1998: Reinforcement Learning: an Introduction, MIT press
- Watkins, C. J. C. H., Dayan P., 1992: Q-Learning, Springer ed.
- Gadanh S. C.,2003: Learning behavior-selection by emotions and cognition in a multi-goal robot task. The Journal of Machine Learning Research. Volume 4 Pages: 385 – 412. MIT Press Cambridge, MA, USA.
- Holland J.H., 1975: Adaptation in natural and artificial system, Ann Arbor, The University of Michigan Press
- Woolridge, M., and Jennings, N., 1995: Intelligent agents: Theory and *practice*. Knowledge Engineering Review 10(2). pp. 115-152

## BIOGRAPHY

**MARCO REMONDINO** got his **Master Degree in Economics** at the beginning of 2001, with *110/110 cum Laude et Mentione*. He then started a **PhD in Computer Science**, during which he delved into theoretical studies about the structure of different software agents, namely the reactive and deliberative (BDI) ones. He completed his PhD in January 2005.

After that, he was awarded a **two-year research scholarship** from ISI Foundation, for the *Lagrange Project on Complex Systems*, during which he applied agent based paradigms to design several models in different scientific fields, namely Game Theory, Biology, Economics and Enterprise Simulation.

At present, he holds a **post-DOC research grant** from University of Turin, Department of Computer Science.

His main research interests are Agent Based Modelling, different paradigms for agents and their integration, computer simulation, Social Systems, emergent properties for Complex Systems, Social Networks, Action Selection, agent learning through Neural Networks, Genetic Algorithms, Classifier Systems, paradigms of Reinforcement Learning, Data Mining, validation of models.