Towards a New Inheritance Definition in Multi-Agent Systems

Antonino Ciuro, Massimo Cossentino, Giuseppe Fontana, Salvatore Gaglio, Riccardo Rizzo and Monica Vitali

***Abstract*—Growing complexity of software systems leads some researchers to explore new paradigms like self-organization and genetic programming. We regard this problem as a new occurrence of a need that has been partially solved in the past with the introduction of object-orientation whose most innovative feature can probably be agreed to be inheritance. In this work, the authors propose a different approach for solving the initially discussed process. The definition of a new "nature-inspired" inheritance is discussed. Agents according to this approach can reproduce by mixing their genome and generate new agents that can better fit a specific problem.**

***Index Terms*—Agent-oriented software engineering, Agents and objects, Relationships between agents and other development technologies.**

I. Introduction

SOFTWARE complexity continuously grows up and pushes researchers towards the definition of new techniques for facing this complexity.

In 1970s, the well-known software crisis encouraged the adoption of new programming paradigms (object-orientation) and new design philosophies (waterfall design approaches were fully exploited and then overcome by more modern evolutionary and iterative/incremental approaches). Despite the relevant advantages offered by modern technologies (service-oriented architectures, model driven engineering, and so on) new challenges have to be faced. Several researchers think that a solution to growing software complexity is in the definition of evolutionary paradigms for software behaviour. Some of them adopt self-organization [7] as the key strategy for solving complex problems. The idea of designing a rather simple system (usually composed by autonomous entities called agents) that can dynamically evolve its behaviour towards the achievement of a goal is for sure very fascinating. The designer needs to define the goal (for instance by using a formal logics or a kind of fitness function) and the entities composing the system will reorganize their individual behaviours and collaborations in order to collectively achieve the goal. The research presented in this paper starts from a similar motivation, it is largely inspired by evolutionary concepts and it aims at exploring the following issue: how can system entities evolve themselves in order to achieve the best fitness for solving a problem? This question has usually been faced by adopting evolutionary criteria of the information manipulated by an algorithm (this is the typical application context of genetic algorithms [6] [5]). If the concept of agent as a highly encapsulated, autonomous, proactive and social entity is introduced in this scenario, a different strategy should be identified because object-oriented inheritance cannot be applied because it violates the high encapsulation of agents. Since agents are frequently regarded as *live entities* the obvious path for achieving such an evolution is to look at natural evolutionary processes and the problem becomes defining an agent-oriented inheritance as the propagation of parents knowledge and abilities to their child with the necessary changes that may ensure evolution. Modern agent-based architectures are mostly based on diffused object-oriented languages and therefore do not offer a specific (agent-oriented) inheritance feature. Sometimes, delegation is adopted as a surrogate for that. In order to define a natural evolution paradigm for agents, it is possible to look at the biological inheritance among individuals. The proposed approach consists in defining a genetic representation of the agent (a DNA-like representation of agent physical and behavioural features), and in allowing the reproduction of couples of individuals by mixing portions of their DNA. Casual changes introduced in the DNA mixing phase also ensure the possibility of pursuing an evolution of species just like it happens in nature. The paper is organized as follows: section 2 introduces the proposed approach, section 3 discusses the Genoma framework we developed to implement the agents' reproduction process, section 4 introduces a case study where the proposed approach is used to approximate some geometrical shapes. Finally some conclusions are drawn in section 5.

Antonino Ciuro is with EMC Corporation, aciuro@gmail.com.

Massimo Cossentino and Riccardo Rizzo are with CNR-ICAR, cossentino@pa.icar.cnr.it, ricrizzo@pa.icar.cnr.it

Giuseppe Fontana is with Accenture S.P.A., gfontana13@gmail.com

Salvatore Gaglio is with Universita' di Palermo, CNR-ICAR, gaglio@unipa.it

Monica Vitali is with Universita' di Palermo, monicavit164@gmail.com

II. The Proposed Approach

In object oriented approaches using inheritance makes portions of useful and general code available to a large set of other classes [9]. Such a code can be easily specialized by programmers in order to fulfil a new goal.

In the agent oriented world such an inheritance process doesn't exist. Some attempts have been proposed in the past (for instance in [4][8]).

In our approach the focus of inheritance is not the code but knowledge and ability of the agents, and the "user" is not a programmer but another agent (the CrosserAgent) that manages the deployment of new agents with new knowledge and abilities. We realize this inheritance feature by means of a reproduction process that starting from two parent agents (that are not able to solve a specific problem), generates a new

agent that (hopefully) better fits the problem. The solution is ensured by the generation of several new individuals that in turn reproduce themselves until the proper result is achieved.

Knowledge and abilities are stored in chromosomes and modified using techniques coming from genetic programming (crossover, mutation, and so on). Using the genetic algorithm point of view, we are looking to a solution in the space obtained by joining the ability and knowledge spaces. In so doing, we suggest an agent reproduction process characterized by two different phenomena:

- from an individualistic point of view: it is highly probable that a part (probably the most useful) of parents' knowledge and ability is forwarded to their children;
- from a social point of view: the behaviour of the new agents "moves" towards the achievement of the assigned goal. This means that plans and knowledge evolve in order to fulfil the desired service.

The first phenomenon concerns the conservation of the useful knowledge and abilities of the agent in the reproduction process. The second phenomenon is about the whole point of genetic programming: the desired behaviour emerges from the agent society during evolution and it can be considered as a "specialization" of the agent.

In order to realize our ideas, we developed the Genoma Framework; this framework allows us to obtain a society of agents capable to increment their efficiency and/or to learn new capabilities. A Genoma agent (an agent that belongs to the framework) is composed of chromosomes defining its knowledge about the world and the plans (addressed as Abilities) that define its behaviours. All the knowledge and the abilities of the agent are coded in a genome structure. Using the information in the genome structure it is possible to implement a reproduction process for the agent. The agent is developed using the JADE platform so it is able to send and receive messages to other JADE agents. The knowledge of the agent is implemented by using Java and it is an instance of ontological elements. In the Genoma framework, the agent society is composed by one or more Genoma agents that can make available many services, and by one agent that manages the reproduction process: the CrosserAgent. Once activated, each Genoma agent sends a chromosome containing all the information that defines the agent to the CrosserAgent.

Inside the agent society all the agents that need a service can ask for that service to all the others inside the society. If the required service is not available, or not satisfying, the agent can ask to the CrosserAgent to activate an evolution procedure. The CrosserAgent will form a society composed by descendants of all agents that can deliver a similar service and it will start the crossover procedure among the selected individuals. The generation zero will contain a lot of duplicate agents, if there are not enough individuals. The following generation will contain many agents of the preceding generation and new agents obtained by crossing genome of the original agents. This evolution process will continue until a suitable agent is obtained (according to some fitness criterion). This resulting agent will now be introduced in the agent society and the CrosserAgent will send to the agent that requested the unsatisfied service its name in order to properly fulfil the

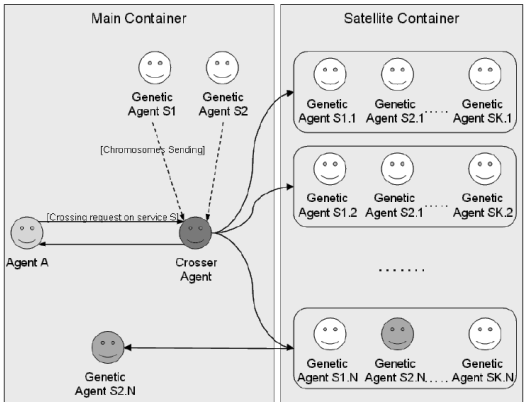

Fig. 1. A sketch of the agent crossover procedure, Genoma agents are white, selected agents are light gray.

service request (Figure 1). The evolution process operates at two different levels: knowledge and plan. The knowledge level is related to instances of (knowledge) objects that belong to the agent, plans are related to how the agents behave and execute the required service. The evolution procedure is different for knowledge and plan and some details will be reported later.

An example will help to explain the full mechanism. Imagine we want to develop an agent capable to escape from a labyrinth. A simple (but not efficient) way to escape from a labyrinth is to go ahead and turn on the same side each time a wall or a turn is met. This plan can be expressed as a graph where each node corresponds to a choice or an action. In the upper part of Figure 2 we have two plans for agents that are trying to solve the labyrinth: the former (agent A on the left) changes its direction whether it finds a wall or not. The latter (agent B on the right) tries to move forward in every circumstance. The nodes of the plans refer to the knowledge of the agent. This is a consequence of the ontology structure adopted in PASSI [2] that is composed of: concepts describing categories of the world, predicates asserting the status of the previous cited concepts and finally actions that can affect concepts status. For instance, the action "turn" is referred to a concept "direction" of the agent. This knowledge piece, once instantiated in the agent, can have four values: right, left, forward or backward. We can suppose it has value left. Both of these agents are unable to solve the problem. The crossover of these agents generates a new individual who has acquired the ability to solve the proposed problem as shown in the lower part of Figure 2. The new agent in Figure 2 inherits from agent B the ability to go ahead if there are not walls (part of the plan on the *no* branch of the second decision node) and from agent A the ability to turn according to the actual value of "direction". This value actually depends from the crossover process applied to the knowledge of the two agents. In a second scenario, we can suppose agent A having the value "back" for its direction and agent B having the value "left" (even if it doesn't use it in its plan). The new agent can inherit one or both parents' knowledges and it will be able to solve the problem only if it inherits the knowledge direction with value left. Figure 3 shows a possible crossover of the

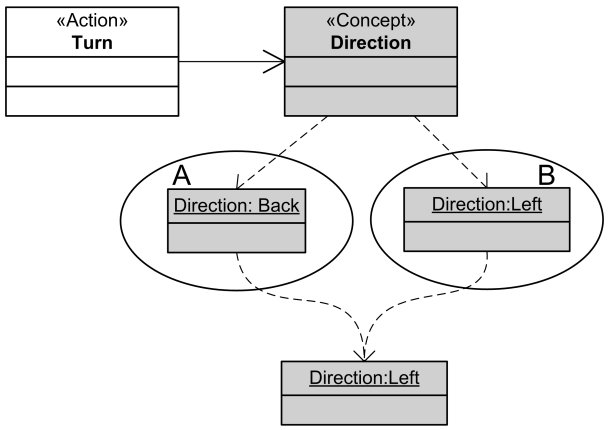

Fig. 3. Diagram which shows the relations between the action "Turn" and the concept "Direction". The instance on the left belongs to the upper left agent in Figure 3, the one on the right belongs to the upper right agent. The instance in the lower part is the knowledge of the new agent generated by crossover.

knowledge "direction". The instance of "direction" on the left belongs to agent A in Figure 2, the one on the right belongs to agent B. Different tecniques for crossing knowledge will be explained further on.

III. The Genoma Framework

The Genoma Framework blends genetic programming to the JADE agent platform[1][3]. All the information required to define an agent and its behaviour are coded in a data structure that is comparable to a genome, as it will be described later. Genomes belonging to two different exemplars of agents are mixed in order to obtain a new agent with some new behaviours.

The basic structure of the framework, as reported in this paper, is applied to the realization of services delivered by single agents, but because of the generality of the approach it can be used for other broader scope tasks. The Genoma Framework is based on the GenomaAgent agent that is a specialization of the JADE Agent. The reproduction process, governed by the CrosserAgent, is applied to couples of these agents and produces new individuals. In the following a description of the GenomaAgent agent will be provided in terms of its structure (subsection III-A) and chromosome (subsection III-B); after that, subsection III-C provides a description of the CrosserAgent.

A. The Genoma Agent Structure

The GenomaAgent agent structure is based on a class that is an extension of the JADE Agent class. The GenomaAgent class is composed of:

- a global plan representing the ability of the agent;
- a set of knowledge items (instances of ontology elements);
- a set of tasks that are used in the global plan;

These elements represent the chromosomes of the agent; more specifically we can talk of an Ability chromosome (representing the agent global plan and composed by activities and control nodes), a knowledge chromosome and a set of task chromosomes (representing the agent activities). Task chromosomes can, in turn, be represented as the composition of an Ability and Knowledge chromosome. The Ability chromosome is composed of activities and control nodes while the knowledge chromosome is composed of referred knowledge (that is the portion of the agent knowledge that is referred in the plan). The global plan (stored in the agent's Ability chromosome) manages the agent's life, defines the roles the agent can assume during its life-cycle, and manages the interactions with the environment and the service delivery. The activities used in the Ability chromosome are elementary pieces of behaviour that constitute the whole set of the agent capacities (usually implemented by using tasks in the JADE platform). Each activity is the representation in the Ability chromosome of an ontology action (stored in the Knowledge chromosome as an Action gene). Tasks realizing activities can have an internal plan that leads the agent towards the achievement of the related sub-goal. Tasks refer to knowledge items for manipulating entities of the environment. As a consequence, Tasks, even though they constitute one of the agent's chromosomes, contain an Ability and a Knowledge chromosome themselves.

All of these elements (ability, knowledge and tasks) define the agent chromosome structure. In order to simplify the operations related to the reproduction process, a plan is represented by a tree structure that is composed by nodes. A node is the basic element of the tree and can contain an action/activity or a predicate. It can have one or more predecessors and one or more successors. If the action or the predicate related to the node are satisfied, all the successors nodes are activated.

B. Chromosomes

The agent structure can be defined using three chromosome categories: a Knowledge chromosome, an Ability chromosome and a Task chromosome. Each chromosome is made by genes; in the first chromosome, each gene describes an instance of the ontology (predicates, concepts and actions). The ability chromosome contains a plan that represents what the agent is able to do.

Figure 4 reports the genome structure in form of a UML class diagram.

1) Knowledge Chromosome Crossing: The generated agent will have a set of knowledge elements derived by the two parents. If parents own a similar knowledge they will generate a new knowledge that will be in relation to both the knowledge of the parents. If one of the parents has a knowledge that does not have any correspondent to the knowledge of the other parent, the new knowledge will be in relation only will the knowledge of one parent (the right parent). In this way, for each knowledge piece of the parents there will be only one knowledge piece in the child. On the other side, the knowledge of the child will be in relationship with at least one knowledge of the two parents.

The evolution process is realized by applying several different crossover techniques to the parents' genome. This process

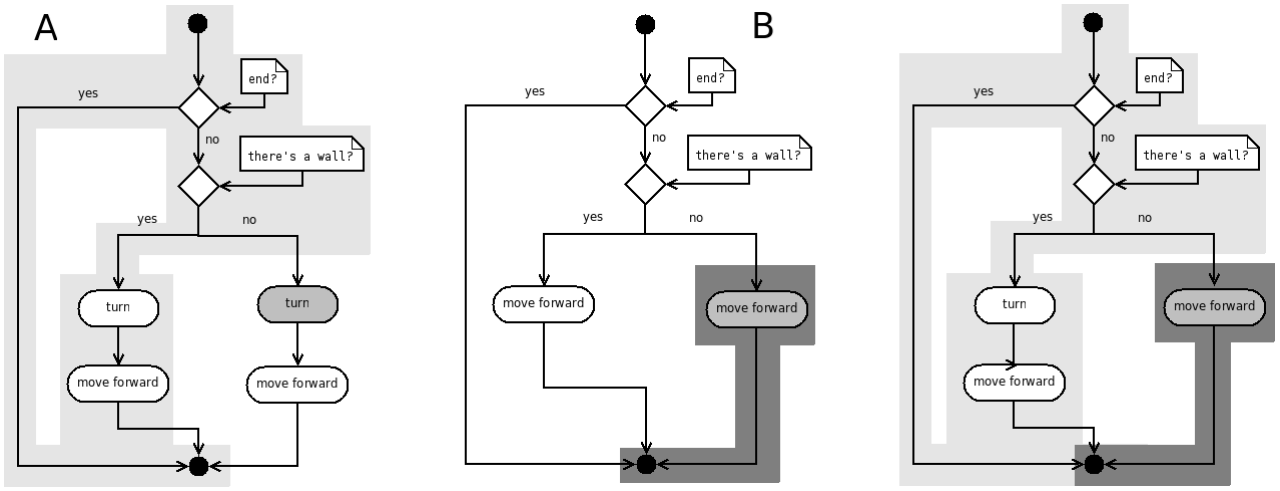

Fig. 2. On the left part of the figure the two plans of the parent agents and on the right the resulting plan. The parts of the plan selected for the crossover procedure are highlighted.

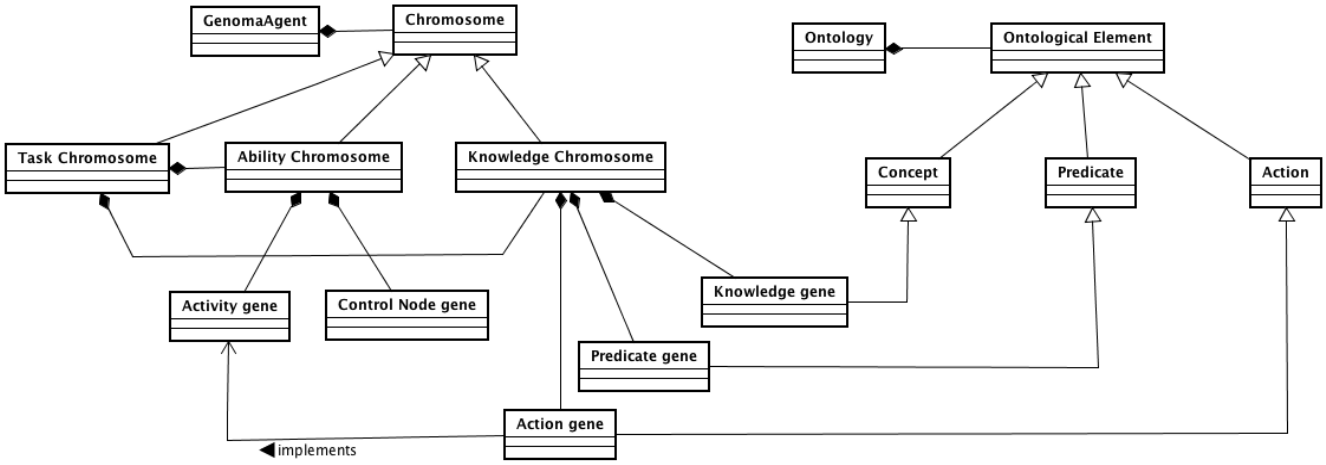

Fig. 4. A UML class diagram representing the agent genome

is strongly conditioned by the need of obtaining a working agent. In order to achieve this objective, it is necessary that knowledge crossover will happen only between similar knowledge elements. There are four techniques to obtain a new knowledge:

- **fusion:** the two parents' knowledge is unified into a single body of knowledge that will contain a weight or algebraic average of the parent knowledge
- **selection:** as in fusion, the two parents' knowledge originate a unique knowledge in the child but, in this case, one of the two is copied and the other one is discarded
- **union:** the new individual's knowledge will be composed by the union of the two parents' bodies of knowledge. A frequent use of this technique may produce a very redundant agent
- **copy:** it is used if one of the two parents has a knowledge the other parent has not. In this case the knowledge is simply copied from the parent to the child.

As it is obvious, we assume that all the knowledge pieces of an agent may have a reference to other knowledge pieces or may be referred by portions of the agent plan. Of course, it is necessary to maintain these references in the child agent during the reproduction process; otherwise it will contain some knowledge portions without any reference (and therefore useless).

An example is in Fig. 5 where the concept "brush" is linked to the concept "grid" (an agent can draw in the grid by using a brush). If we remove this link, the crossover may generate

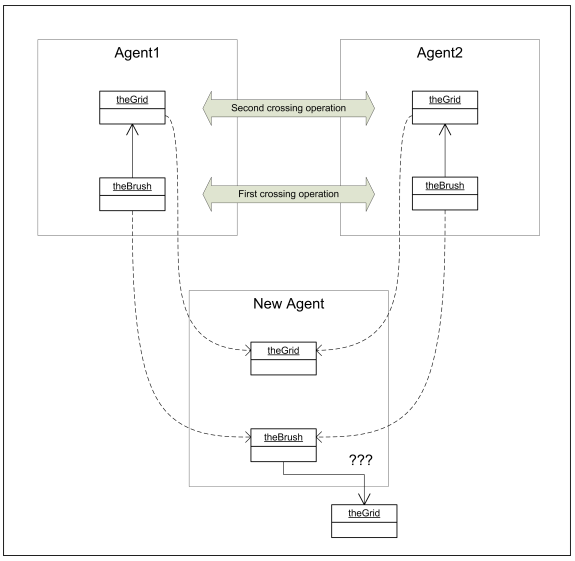

Fig. 5. Crossover where there are referred knowledge pieces

a concept "grid" without any reference to the object "grid" referred by the two parents.

The algorithm applied for the knowledge chromosomes crossing is the following:

1) A crossover operation (fusion, selection, union, copy) is assigned to each knowledge gene element contained in the agent chromosome;
2) An item, containing the selected operation and the parents' knowledge, is added to the list of crossover operations to be done.
3) The first item of the list is selected and the corresponding operation is performed. Then the resulting knowledge is verified:
   a) if both the objects referred by the parents' knowledge were not marked then a new mark is created with the same crossover operation and it is added to the bottom of the crossover list
   b) if at least one of the knowledge was marked for the crossover operation, a reference is created.
4) if the list is empty then end, else go to step 3.

2) Plan Chromosome Crossing: Plan crossing is the operation that allows obtaining the child plan from the two plans of the parents. Plan crossing is some way less complex than the previous discussed knowledge crossing, because all the referred knowledge genes have been already crossed at this point. Generic tasks composing the plan (and referred in the knowledge as actions) are crossed by using the same techniques explained in the previous subsection. In this case it is necessary to distinguish the two parents' agents in, a randomly labelled, "mother" and "father" agent. The plan fragment obtained from the "mother" agent will always contain the starting node and will be obtained as follows:

1) randomly select a cut node
2) cancel all the links that start from the cut node
3) cancel all the nodes that are not reachable

An example of "mother" agent plan is represented in the top-left part of Figure 2. The plan fragment obtained from the "father" agent will always contain the end node and will be obtained as follows:

1) randomly select a cut node
2) cancel all the nodes that are not reachable from the cut node

An example of "father" agent plan is reported in the top-right part of Figure 2. The "child" agent plan is obtained substituting the cut node of the "mother" with the cut node of the "father" (see bottom plan in Figure 2).

3) Mutation: Mutation allows obtaining a new individual from a single parent by modifying in a random way one of its characteristics. If mutation is applied to a knowledge item, an attribute is selected and a random value is given to the node. If mutation is applied to a plan a node is replaced with an equivalent one or a link is added in a random way. The probability that an efficient individual is obtained after mutation is low but mutation adds an unpredictable variation that sometimes allows obtaining unexpected improvements.

C. The Crosser Agent

The CrosserAgent is responsible for managing the reproduction process that includes the creation of agent generations, the execution of new agents, and the evaluation of the results achieved by each agent. The behaviour of this agent is composed by a set of tasks executed according to a specific plan (Figure 6 represents it as a PASSI [2] Task Specification diagram). More in details, the most relevant components of this behaviour are:

- **Listener:** this task is responsible for message receiving and management. Messages containing agent genome and crossing requests are usually received by this agent.
- **ManageRequest:** this task is responsible for creating the necessary file system structure (namely directories) for storing next agent generations.
- **GenerateChildren:** it is responsible for child agent creation and chromosome crossover operations.
- **CompileChildren:** it compiles the classes (agents) built by the GenerateChildren task.
- **PollChildren:** it instantiates agents of the current generation and introduces them in the agent platform. Then it requests them the service that is to be evaluated to esteem the agent fitness to solve the assigned problem. All the agents of this generation have a fixed time for performing the assigned duty. If they complete the work in time, their fitness is evaluated, otherwise they are discarded.
- **EvaluateChildren:** it calculates the fitness function value for each agent and stores it.
- **ManageGeneration:** it selects the agents that will compose the next generation. This new generation will be composed of elite agents (the best agents of the current generation) and the children of this generation that better realised the required service. If the crossover process is

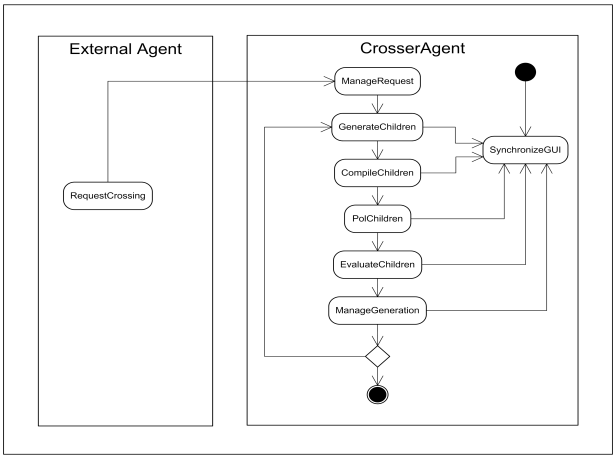

Fig. 6. The CrosserAgent plan represented as a PASSI Task Specification diagram

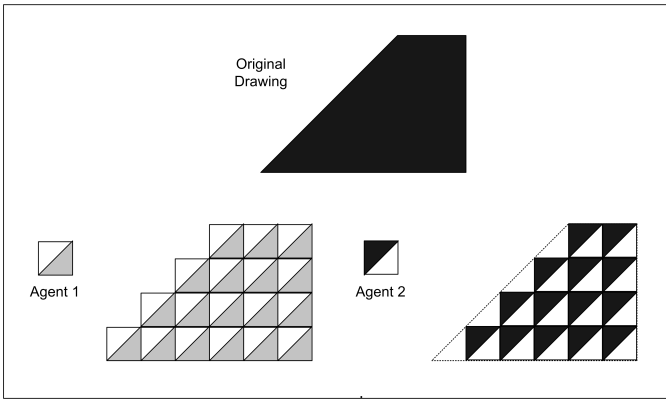

Fig. 7. The objective of the case study is reproducing the shape at the top of this figure. Initial agents are able to achieve a limited approximation of the required goal.

completed (maximum number of generations or maximum value of fitness function reached) then it selects the best agent.

In the current implementation of the framework, crossover operations can be done by only considering agents providing the same service.

IV. Experimental Results

The discussed approach has been applied to a simple case study where the goal is to be achieved by a single agent; the goal consists in reproducing a design in shape and colour (see the shape at the top of Figure 7).

A. Reproduction of a Graphical Structure

This case study uses an agent-oriented system for solving a problem characterized by several different variables. The problem consists in reproducing a shape in both colour and geometry; the fitness function considers both shape similarity and colour proximity.

Figure 7 reports the original shape to be reproduced (experiments have been done by adopting different shapes, sizes and filling colours). Each agent of the first generation (composed by the two individuals depicted in Figure 7) was able to draw by using elementary cells (brushes) of square shape. Inside the brush, it was able to draw a triangle with a vertex on the lower-right corner (the other agent was able to draw a triangle with a vertex on the higher-left corner). Each agent designed its brush in a different colour. An optimal solution to this problem requires that agents cross their genome in order to learn how to draw brushes that can produce a good reproduction of the goal picture in both shape and colour. During the evolution process, brushes of different size can emerge and the triangle can change its orientation and even its shape. The fitness function calculates the percentage of the shape that is filled (and not left blank) and the similarity in the filling colour to the goal picture. Figure 8 reports results obtained by individuals of the fourth generation; they are interesting because they show the different evolution paths that can be undertaken during

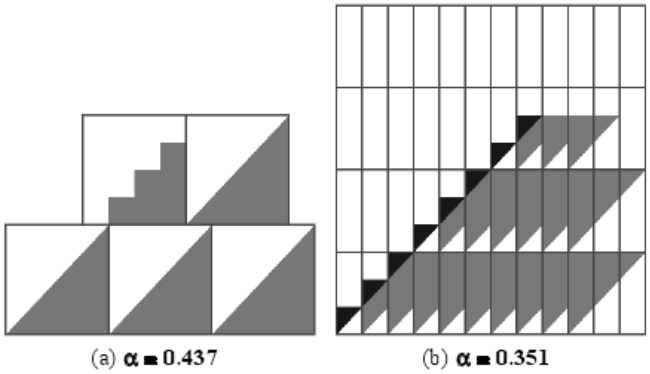

Fig. 8. Results obtained by individuals of the fourth generation and the corresponding fitness function

the process. We found that the number of generations that is necessary to build in order to find the optimal solution to the problem depends (as it was expected) on the complexity of the goal picture. In the reported example, nine generations have been necessary in order to obtain some individuals (more than one obtained the same score) achieving the optimal solution. A few examples of good and optimal solutions obtained from individuals of the ninth generation are reported in Figure 9.

V. Conclusions and Future Works

We proposed a new inheritance approach that focuses on agent knowledge and abilities as a subject of reuse.

In our approach the agents do not inherit code or classes but behaviours and pieces of knowledge; genetic programming techniques modify and specialize them in order to fulfil a defined goal or delivery a requested service. In principle, the management of this procedure is delegated to an agent so that this process is automatic and transparent to the user: agents are evolved by reusing and modifying their knowledge and abilities without the programmer help.

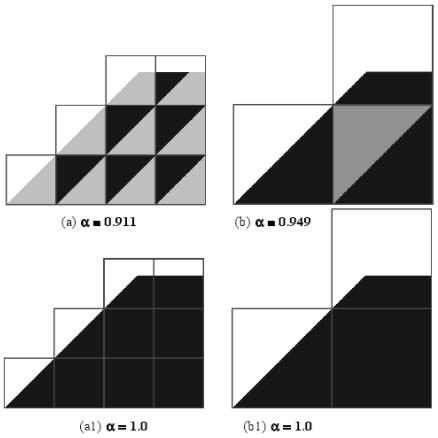

Fig. 9. Results obtained by individuals of the eight (subfigures (a) and (b)) and ninth generations (subfigures (a1) and (b1))

Many aspects of the proposed method need further investigations: for example we noticed that there are also part of the plan of the agent that are not used but merely transmitted from parent to children and they remain present in the agents that reach the final goal. These parts of plan and knowledge can be considered a sort of "memory" of the agent society and they can be useful if another service is requested. In fact from an optimized set of knowledge pieces and set of abilities it can be difficult to obtain something new. It is possible to think that if we optimize the agent by deleting unused knowledge and abilities future generations will cover a smaller area of the "solution" space. The impact on the agent functionality, and on the agent society, of these parts of plan and knowledge that are not used are worth of more investigations.

REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa2000 compliant agent development environment. *Agents Fifth International Conference on Autonomous Agents (Agents 2001)*, 2001.

[2] M. Cossentino. *From requirements to code with the PASSI methodology. In Agent Oriented Methodologies.* Idea Group Publishing, june 2005.

[3] M. Cossentino and L. Sabatucci. *Agent System Implementation*. CRC Press, April 2004.

[4] L. Crnogorac, A.S. Rao, and K. Ramamohanarao. *Analysis of inheritance mechanisms in agent-oriented programming*. Morgan Kaufmann Publishers Inc., 1997.

[5] R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, 1992.

[6] Nils J. Nilsson. *Artificial Intelligence, a New Synthesis*. Morgan Kaufmann Publishers, Inc., 1998.

[7] G. Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self organisation and emergence in mas: An overview. *Informatica Journal*, 2005.

[8] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Agent cloning: An approach to agent mobility and resource allocation. *IEEE Communication Magazine*, 1998.

[9] A. Snyder. Encapsulation and inheritance in object-oriented programming languages. In *In proc. of Conference on Object Oriented Programming Systems Languages and Applications*, pages 38–45, 1986.