2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Applying Evolutionary Computation for Evolving Ontologies

Oliviu Matei, Diana Contraş, Petrică Pop

***Abstract*— In this paper, we describe a novel application of evolutionary computation, namely for evolving ontologies. The general algorithm of evolutionary ontologies follow roughly the same guidelines as any other genetic algorithms. However, we introduced a new genetic operator, called repair, which is needed in order to make the offspring viable. Experiments for the generation of user centered automatically generated scenes demonstrate the performance of the proposed approach.**

I. Introduction

Artificial Intelligence (AI) has become nowadays more critical and more pragmatic. Due to the fact that the actions are becoming more complex and are taking more time, a lot of interest falls on automated knowledge representation.

The genetic algorithms (GAs), which were formally introduced in the 70th by John Holland at the University of Michigan [1], represent an important part of AI with a wide applicability in various fields: bioinformatics, computational science, engineering, chemistry, economics, etc. Since their introduction, GAs evaluated during time: if at the beginning required binary strings as individuals, recently the float representation of the chromosomes are claimed, if at the beginning GAs used only one mutation operator in producing the next generation, it was shown that each optimization problem may require different mutation operators in order to obtain better results, etc.

In computer science, an ontology formally represents knowledge as a set of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts [2]. AI uses ontology as a formal conceptualization of knowledge through representation of the basic concepts and the relationships between them.

In AI, the GAs and the ontologies followed separate paths so far. We consider that one of the next steps in AI is represented by evolutionary ontologies, meaning the use of ontologies as individuals of GAs. The result will be a new ontology, which means enriched knowledge, more entities, more and more complex relations.

In this article we focus on a precisely ontology in order to mark out the importance of ontologies in evolutionary computation.

Charles Darwin proposed in his book, *The Origin of Species* the concept of evolution based on the system of natural selection. The American zoologist, russian-born, Th. Dobzhansky showed that the selection works at micromutation level which leads at population's genes variation, therefore at their evolution. Dobzhansky succeeded in this way to make the connection between genetics and Darwin's book, laying the foundation (with other biologists) of synthetic theory of evolution.

The theory of evolution was taken over informatics appearing evolutionary computation. Nils Aall Barricelli, a Norwegian-Italian mathematician laid the foundation of evolutionary computation research in 1954. There are four directions of evolutionary computation: genetic algorithms, genetic programming, evolution strategies and evolution programming. However, GAs have become popular only in the 70s thanks to John Holland and his colleagues at the University of Michigan [1].

In GAs a potential solution to an optimization problem is represented as a set of parameters known as genes. These parameters are joined together to form a string of values known as a chromosome or individual. In the initial algorithms, a gene was a sequence of bits.

Like in nature, the population evolves by applying three genetic operators (used in any evolutionary technique):

- Crossover (recombination) between two random individuals;
- Mutation of a gene with a certain probability;
- Selection of a new population (epoch).

GAs have been developed further into evolution strategies (ESs) by Ingo Rechenberg and Hans-Paul Schwefel. Ingo Rechenberg is a specialist in aerodynamics who found that the mechanism of evolution and natural selection can be applied to technical problems such as aerodynamic wing design. Unlike the GAs, the ESs make use the floating numbers as chromosomes, the mutation and crossover operators are adaptive and depend on the fitness of each individual and the selection is deterministic.

Based on GAs, John R. Koza [3] developed the genetic programming (GP). The main difference between genetic programming and genetic algorithms is the representation of the solution, namely, genetic programming creates computer programs in Lisp or scheme computer languages as the solution while genetic algorithms create a string of numbers that represent the solution.

All these evolutionary techniques have been developed to use different representations for individuals, such as character strings, coordinates, vectors, matrices, etc.

However the current research in the field of AI imposes the use of ontologies for formally representing knowledge as a set of concepts within a domain and the relationships between concepts. They can be used also for reasoning about

Oliviu Matei and Petrică Pop are with the North University Center of Baia Mare, Technical University of Cluj-Napoca, Romania (email: oliviu.matei@holisun.com, petrica.pop@ubm.ro).

Diana Contraş is with Automation Department, University of Cluj-Napoca, Romania (email: pdia17@yahoo.com

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement No609143 Project ProSEco.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

concepts. They allow a complex representation of the entities and their relationships, including logical (AND, OR, NOT), universal (ANY) and existential (EXISTS) operators. Therefore the knowledge within an ontology is significantly more powerful than the data within any classical data structure as so is their expressive power.

An ontology consists of a number of different components. According to Lord [4], these components can be divided usually in two parts: those describing the entities of the domain and those describing the ontology itself. Common components of ontologies include:

- Individuals: are the base unit of an ontology and may model concrete or abstract objects;
- Concepts: also called classes, types or universals are the core component of most ontologies and represent a group of individuals that share common characteristics;
- Relations: describe the ways in which the individuals or the concepts can be related to one another;
- Attributes: describe the aspects, properties, features, characteristics, or parameters that the individuals or the concepts may have;
- Function terms: are complex structures formed from certain relations that can be used in place of an individual term in a statement;
- Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input;
- Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form;
- Axioms: represent the knowledge given as input described in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of *axioms* in generative grammar and formal logic. In those disciplines, axioms include only statements asserted as a priori knowledge. As used here, *axioms* also include the theory derived from axiomatic statements;
- Events: are changes that occur to attributes or relations. While the previously described components are static, the events are dynamic.

We believe that the evolutionary ontologies are the next step of the evolutionary computation, making use of ontologies as individuals, rather than numbers, programs or other data structures. The result is creation of new ontologies, which means enriched knowledge, i.e. more entities, more (complex) relations. This is specifically important for exploring the knowledge space by creating new concepts.

The aim of this paper is to apply the evolutionary principles on ontologies. The rest of the paper is organized as follows: section II presents some relevant articles about evolutionary computation and ontologies, but we must to emphasize that the two concepts have never been used together. Section III shows the general concepts of genetic algorithms applied on ontologies and section IV describe the experiments used for testing the concepts.

II. Related work

This section gives an overview on the main results on ontologies: types and operations and as well some GA approaches designed especially to ontology matching problem and optimal ontology alignment.

Daum and Merten [5] identified four types of ontologies:

- top-level (also known as upper ontology or foundation ontology) - describes very general concepts and are serving the scopes of a large comunity;
- domain-related - are formal descriptions of classes of concepts and the relationships between those concepts that describe a specific area of knowledge (for example computer industry or academia);
- task-related (also known as horizontal ontologies) - are applied to a task or to a group of related tasks (for example, the software requirement analysis) and provide a mechanism to organize and standardize information content;
- application-related - describe the concepts involved in an application, referring to specialization of an domain-related or task-related ontology.

In some cases, one application may use multiple ontologies, therefore some operations on ontologies are necessary to be introduced so far. Some of the main operations on ontologies summarized by Obitko [6] are described in what it follows:

- *Merge* of ontologies - the creation of a new ontology from other ontologies that already exist. The new ontology may contain all the knowledge from the original ontologies or only a part of it if the ontologies are not consistent. Also the new ontology may introduce new concepts and relations destined to create a bridge between terms from original ontologies.
- *Mapping* from one ontology to another - represents the translation of concepts and relations from one ontology to another one.
- *Alignment* - there are situations in which reality is configured by different persons in different ways. Thus appear heterogeneous ontologies which describe the same domain of knowledge, but they are created independently by experts that do not communicate with each other. It requires that such ontologies to be mapped resulting global ontology. This mapping process is called alignment.
- *Refinement* - is the process of mapping from one ontology to another one such that every concept from the original ontology has equivalent in the other ontology. This operation defines a partial ordering of ontologies.
- *Unification* - represents the alignment of all of the concepts and relations in ontologies so that inference in one ontology can be mapped to inference in other ontology and vice versa. Usually, this operation is made as refinement of ontologies in both directions.
- *Integration* - is the creation of a new ontology from two existing ontologies. Depending on the number of

changes between the given ontologies during development of the new ontology the level of integration can range from alignment to unification.
- *Inheritance* - means that an ontology inherits everything: concepts, relations and restrictions or axiom, from another ontology.

In the literature, there have been introduced some GA approaches to different ontologies: Wang et al. [7] presented a genetic algorithm based approach for solving the ontology matching problem, by modeling the problem as an optimization problem of a mapping between two compared ontologies. Qazvinian et al. [8] used GA in order to find the best mapping between concepts of two ontologies. Martinez-Jil et al. [9] presented a mechanism in order to compute the optimal ontology alignment function for a given ontology input set, based on genetic algorithms. Naya et al. [10] introduced a GA-based approach for automatically align ontologies. Recently, Romero et al. [11] developed a GA based approach for optimizing similarity aggregation in ontology matching.

III. Evolutionary ontologies

An ontological evolutionary algorithm is a genetic algorithm in which the individuals are ontologies rather than any other data structure. That is why we call them *evolutionary ontologies*.

The evolutionary ontologies undergo an evolutionary process, described in algorithm 1. The first step will be the definition of the ontological space (see section III-A for more details).

Algorithm 1 The Evolutionary Ontologies - pseudo code showing the overall algorithm for evolutionary ontologies

```
1: OS = createTheOntoSpace()
2: population = createInitialPopulation()
3: currentEpoch = 0
4: while (currentEpoch ≤ maxNumberOfEpochs) and (isSuboptimalSolution) do
5:    newPopulation = recombine(population)
6:    mutate(newPopulation)
7:    validate(newPopulation)
8:    repair(newPopulation)
9:    Population = select(population, newPopulation)
10:   currentEpoch = currentEpoch + 1;
11: end while
```

The ontological space is defined in section III-A. The individuals, their representation and initial creation are described in section III-B. The genetic operators, including the pseudo code for them, are detailed in section III-E, III-D, III-F, III-C.

Like for any evolutionary program, a set of parameters needs to be defined for the evolutionary ontologies:

- ontological space
- representation of the individuals
- initial population
- genetic operators
  - selection
  - mutation
  - crossover
- fitness function.

A. Ontological space

We define the ontological space (onto-space) as an ontology describing a domain specific knowledge, containing all the concepts along with their allowed and denied relationships. The onto-space defines the degrees of freedom as well as the boundaries of the solution space to be searched by the evolutionary process. Quite often, the solution space is infinite and special algorithms are needed for exploring it efficiently. An onto-space would be the ontology about all electronic appliances and the solution required to be found is a possible arrangement of a kitchen given some restrictions.

Formally, an onto-space $OS = (C, P, I)$, where C is the set of classes, P is the set of properties and I is the set of instances.

Within the ontology OS, there are two disjunctive sub-ontologies $OS_e = (C_e, P_e, I_e)$ and $OS_f = (C_f, P_f, I_f)$, with

$$OS_e \cup OS_f = OS \tag{1}$$

OS_e is the sub-ontology which will undergo the evolutionary process and OS_f is the fixed sub-ontology, e.g. which will not change under the evolutionary process.

The concepts described in this article are exemplified on an ontology used for scene generation. This way, the relationships defined in the ontology are very visible and tangible. On the other hand, the ontology has the power of dealing with various and complex relations between individuals, such as `above`, `top`, `left`, `top_left`.

Every scene has a frame and a content. Therefore the superclasses of the ontology are *Content* and *Frame*.

Any scene can take place either in an indoor area or in an outer one. For the class **Frame** should be chosen subclasses *Exterior* and *Interior*. Consistent with them will be chosen as subclasses of the class **Content**: *ExteriorContent* and *InteriorContent*.

The main elements found in an outside area are land and water. Based on this, the class **Exterior** has two subclasses: *Land*, *Water* and the class **ExteriorContent** has two subclasses: *LandContent*, *WaterContent*.

Each item, whether it is land or water, can be populated with objects, plants, animals. So, the class **LandContent** has as subclasses *Car*, *LandAnimal*, *LandPlant* and the class **LandWater** has as subclasses: *Algae*, *WaterAnimal*, *WaterObject*.

Customizing the class **LandAnimal** with the most representative living animals on earth we choose subclasses: *LandBird*, *Mammal*, *Reptile*.

Proceeding the same way for the class **LandPlant** we establish subclasses: *HighPlant* and *SmallPlant*. Further, we devide both classes in another two classes: *Bush* and *Tree* for class **HighPlant**, *Flower* and *Grass* for class **SmallPlant**.

We use the same pattern to customize the class **WaterAnimal**, which has two subclasses *Fish*, *WaterBird* and the class **WaterObject** which has two subclasses *Boat*, *Ship*.

An inner area can be populated with objects and specific plants. Therefore, the class **InteriorContent** has two subclasses: *InteriorObject* and *InteriorPlant*.

We go ahead and we choose two types of **InteriorObjects**: *Furniture* and *Picture*. For the class **Furniture** we establish three subclasses: *Chair*, *Locker* and *Table*.

The hierarchical structure of the classes lead to a tree structure and figure 1 illustrates such a structure.

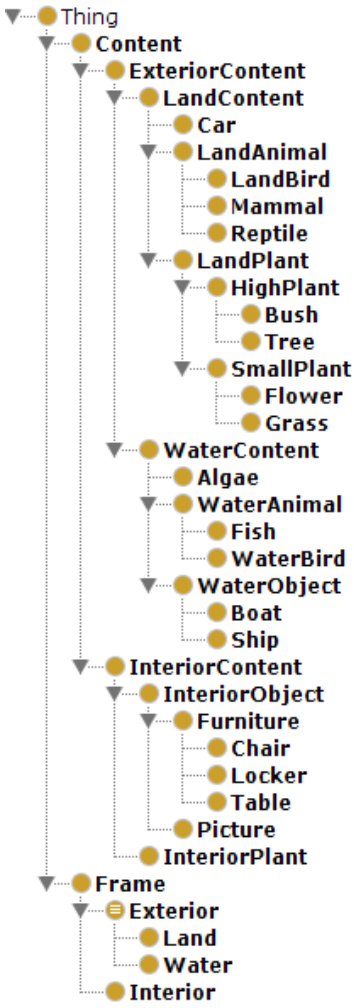

Fig. 1. Class hierarchy

B. Representation of individuals

An individual is a a subset of the ontology, represented as $Ch = (C_i, P_i, I_i)$, where $C_i \subset C$ is a subset of classes in OS, $P_i \subset P$ is a subset of properties in OS and $I_i \subset I$ is a subset of instances in OS. Further on, a genetic individual consists of an evolving part $Ch_e = (C_ie, P_ie, I_ie)$, which will be changed during the evolutionary process, and a fixed part $Ch_f = (C_if, P_if, I_if)$. The two parts fulfill the following relations:

$$Ch_e \cup Ch_f = Ch \tag{2}$$

$$\tag{3}$$

Moreover:

$$Ch_e \subset OS_e \tag{4}$$

$$Ch_f \subset OS_f \tag{5}$$

$$Ch \subset OS \tag{6}$$

A population consists of a given (μ) such individuals and does not necessarily cover the entire onto-space, therefore $\bigcup_i Ch_i \subset OS$.

In our case, an individual is a possible scene represented as an ontology, which has the same class structure as the onto-space presented in section III-A. However, with respect to the instances, there is a difference: unlike the onto-space, which contains all the instances, a scene may contain only several of them, based on the object properties defined in the onto-space. For instance, a valid scene cannot have an instance belonging to `Frame/Interior` class as well as instances belonging to `Content/ ExteriorContent/ WaterContent/ WaterObject/ Ship` class due to the property `isInterior(x, y)={∃ x ∈ InteriorContent ∃y ∈ Interior}` defined in the onto-space, which not include a ship in an interior frame.

The class tree for individuals is illustrated by the Figure 2 and Figure 3:

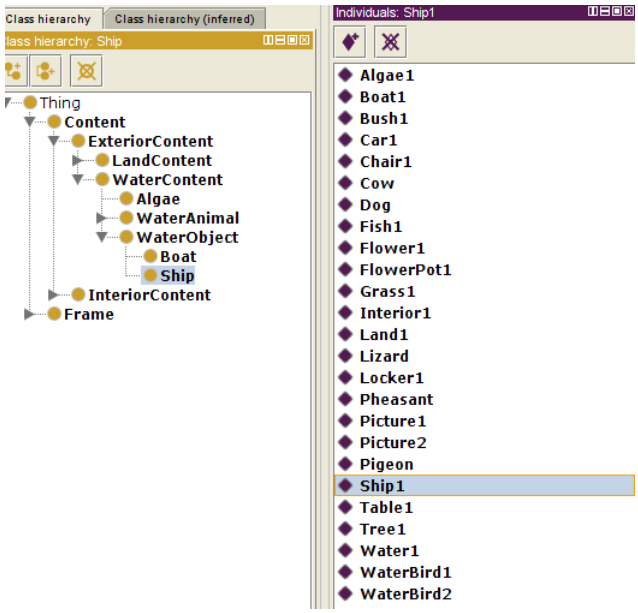

Fig. 2. Individuals

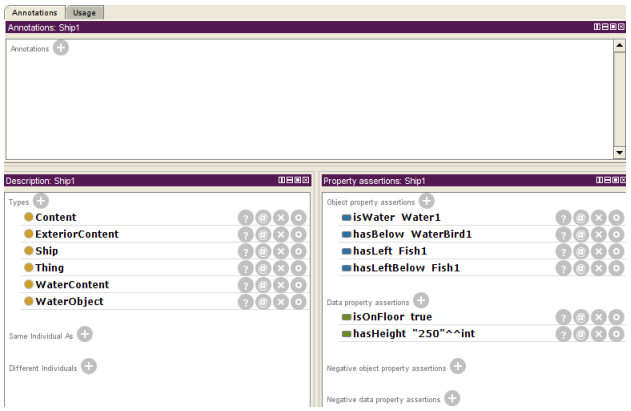

Fig. 3. Individual Ship1

In Figure 2, the second column represents the instances belonging to the classes.

A semantic network representation of the properties of the individual *Interior*, *Table1*, *Chair1*, *FlowerPot1*, *Locker1*, *Picture1*, *Picture2* would be like the one presented in the next figure.

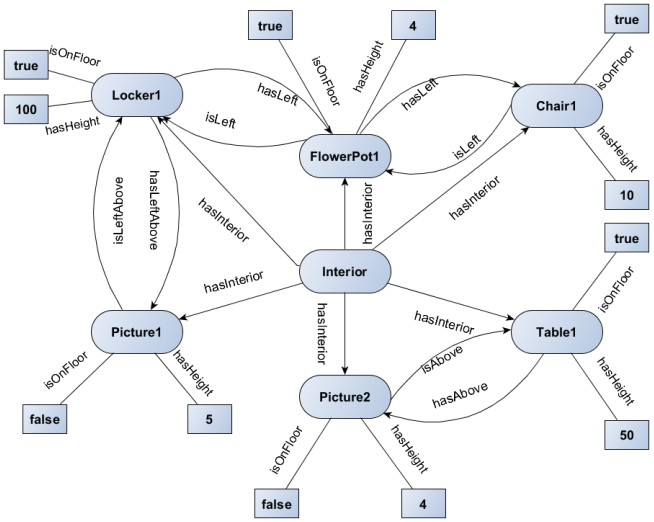

Fig. 4. Semantic network representation

C. Selection

Selection is the stage of a genetic algorithm in which individuals are chosen from a population for later breeding (crossover or mutation). Any selection operator may be used within the process, whether it is based on Monte Carlo technique or it is deterministic. Monte Carlo (or roulette wheel) technique assumes that the parents are selected according to their fitness: the better the chromosomes, the better chances to be selected they have.

The deterministic selection process is strictly based on fitness of the population. In the case of (μ, λ)-selection, μ parent produce λ ($\lambda > \mu$) and only the offspring undergo selection. In other words, the lifetime of every individual is limited to only one generation. The limited life span allows to forget the inappropriate internal parameter settings. This may lead to short periods of recession, but it avoids long stagnation phases due to unadapted strategy parameters.

Choosing a $(\mu+\lambda)$-selection, the population may get stuck into local optima. However, the convergence is significantly improved.

For a more detailed discussion about the selection operator we refer for example to [12].

D. Mutation

The mutation operator is applied for each individual with a probability p_m and requires different treatment for classes, for instances, respectively for properties.

The pseudo code for mutation is shown in algorithm 2.

Algorithm 2 The Mutation - pseudo code describing the mutation procedure

```
1:  procedure mutate(newPopulation)
2:  for all ind ∈ newPopulation do
3:      choose a random number r ∈ [0, 1)
4:      if r < p_m then
5:          choose a random integer number
            r_m ∈ 1, 2
6:          if r_m = 1 then
7:              applyInstanceMutation(ind)
8:          else
9:              applyClassMutation(ind)
10:         end if
11:     end if
12: end for
13: end procedure
```

1) Instance Mutation: Instance mutation means replacing a randomly selected instance i belonging to a class C ($i \in C$) with another individual $i' \in C$ from the onto-space OS. This operator preserves the number of ontological instances in an individual.

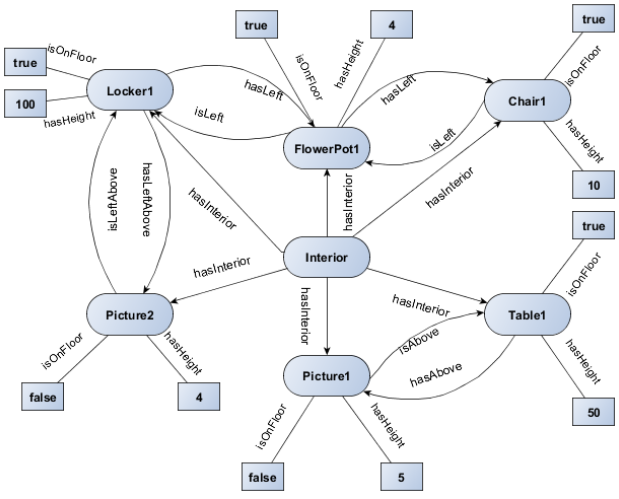

Fig. 5. Instance mutation

2) Class Mutation: A class means replacing all the instances in a class C by other individuals belonging to a random subclass $SC \subseteq C$ in the onto-space. The process is described in algorithm 3:

Algorithm 3 The Class Mutation - pseudo code showing the class mutation

```
procedure mutate(indidivual)
select a random class C in the individual
select a random class SC ∈ OS and SC ⊆ C
select r random individuals i'_k ∈ SC, k = 1,r
replace all individuals i_p ∈ C with the individuals i'_k
end procedure
```

This operator alters the number of ontological individuals in an individual.

In Figure 6 is used a semantic network to represent some classes of the ontology.

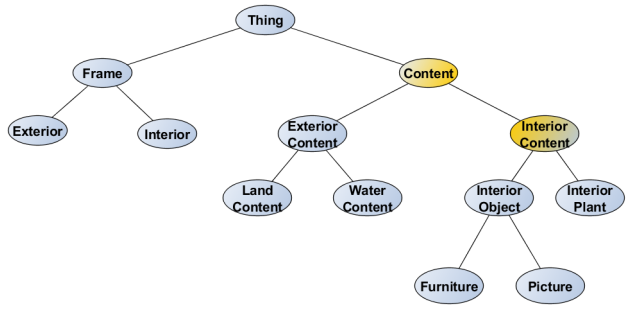

Fig. 6. Class representation

The two classes marked in Figure 6 occur in the class mutation algorithm. The result can be seen in Figure 7.

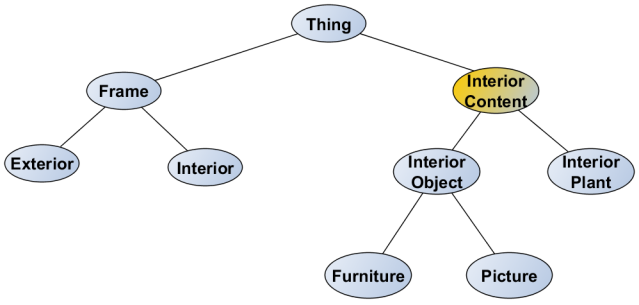

Fig. 7. Class mutation

3) Property Mutation: The property mutations may be approached separately for data properties, respectively for object properties.

Data properties are relations between instances and RDF literals or XML schema datatypes. An example would be `Picture1 hasHeight 20 cm`. Mutating a data property implies, actually, to apply a classical mutation operator suitable for the datatype. For instance, mutating a boolean data property means flipping data from TRUE to FALSE or from FALSE to TRUE. Mutating real valued data properties means adding (normally or uniformly distributed) random values [13].

E. Crossover

In algorithm4 we describe the pseudo code for the crossover operator.

Algorithm 4 Crossover - pseudo code showing the crossover operator

```
procedure recombine(population)
newPopulation = ∅
parentPopulation = select λ individuals randomly
for all ind_1 and ind_2 ∈ parentPopulation do
    choose a random cutting point (in the tree formed by the classes)
    create two offspring by preserving the ordering position of symbols in the corresponding sequences of the parents
    adjust the object properties according to the new class structure
    add the two offspring to the newPopulation
end for
end procedure
```

F. Repair

Ontologies represent a complex structure and they are very susceptible of being corrupted by the classical genetic operators: crossover and mutation. We introduced a new operator, called repair, which is a deterministic operator. It can be applied on the population each time another genetic operator is used or only once, after crossover and mutation. Repairing an individual means adjusting its instances and properties so that they respect all the rules defined in the onto-space.

The repair algorithm is depicted in what it follows:

G. Fitness function

Every solution has a fitness value assigned to it, which measures its quality. This function depends on the problem to be solved. However, such a complex representation of the individuals allows not only numerical fitness functions, but also complex relationships.

1) the first one represents the number of user preferences met by each individual
2) in the second stage, the fitness is given by the user herself, meaning how much she likes a specific scene

IV. Experimental results

The concept of *evolutionary ontologies* has been applied for the generation of user centered automatically generated scenes. Gero [14] used the genetic algorithms for creating Mondrian-styled paintings, but they are different in the way that they are contained colors rectangles and lines, rather than scenes.

Algorithm 5 The **repair** operator

```
1: procedure repair(population)
2: for all ind ∈ population do
3:   for all property ∈ ind.objectProperties do
4:     if property needs to be repaired then
5:       let obj1 P obj2 be the property between object
         obj1 and obj2
6:       if obj1 needs to be repaired then
7:         let C1 be the most specific super-class of obj1
           which holds the property R
8:       end if
9:       if obj2 needs to be repaired then
10:        let C2 be the most general super-class obj2
           which holds the property R
11:      end if
12:      if C1 = ∅ or C2 = ∅ then
13:        remove property
14:      end if
15:    end if
16:  end for
17: end for
18: end procedure
```

The scenes are generated randomly at the beginning and represent the initial population. Based on some user preferences (such as inside or outside, with or without humans etc.) the scenes evolve until the preferences are met. Then the user marks each scene with a level of satisfaction, on a scale from 0 to 10. Based on these marks, the population evolves until at least one scene is graded 10.

The experiments have been carried out on 10 subjects, based on the following methodology. The size of the population has been set to 10, which means that the subject is presented 10 scenes each generation.

1) a list of scenes are generated randomly
2) the user grades them on a scale from 1 to 10, based on how much she likes them
3) the evolutionary process runs for maximum 20 epochs, using a Monte Carlo-based selection operator, in which the grade plays the role of the fitness function.
4) the process stops when at least one scene is graded 10.

The results of the experiments are described in table I. Column **Initial** shows the results for the initial population (representing the randomly generated scenes) and column **Final** shows the results at the end of the evolutionary process, which is done when at least one scene is graded 10 or the maximum number of 20 epochs is reached. Columns **Best** represents the best grades, whereas **Avg** are the average grades for each generation (initial, respectively final).

TABLE I
THE EXPERIMENTAL RESULTS

Subject	**Initial**		**Final**		
	Best	Avg	Best	Avg	# epochs
Subject 1	8	7.4	10	8.9	5
Subject 2	9	7.2	10	9.3	7
Subject 3	10	8.6	10	8.6	0
Subject 4	6	3.6	9	7.9	20+
Subject 5	7	5.5	10	8.7	14
Subject 6	7	6.3	10	8.8	18
Subject 7	8	6.9	10	7.8	8
Subject 8	9	8.0	10	9.0	9
Subject 9	5	2.4	7	4.4	20+
Subject 10	5	4.0	10	6.7	17

As can be seen in table I, one subject (*Subject 3*) graded a scene with 10 and there was no evolution. Other two subjects (*Subject 4* and *Subject 9*) have graded 10 none of the scenes, not even 20 epochs. All the others trials ended between 5 and 18 generations. We want to emphasize that this has not been a psychological experiment, but it was rather meant for researching the evolutionary ontologies.

V. CONCLUSIONS

In this article we present the approach of applying the evolutionary computation for evolutionary ontologies. A general algorithm of evolutionary ontologies follow roughly the same guidelines as any other genetic algorithm. However, we consider that a new genetic operator - **repair** - is needed to making the offspring viable. Otherwise the risk of having no viable individuals after only a few epochs is high.

Although the evolutionary ontologies seem very similar to genetic programming because the hierarchy of the classes represented as a tree structure, there are significant differences between the two concepts, such as:

- the only relationship between a class and a subclass within an ontology is a `is a` relationship, whereas the GP works with programs (structured as trees). In this case, the nodes are operations or instructions, except for the leaves, which are operands;
- the genetic operators are completely different in the case of EO, respectively GP;
- more operators may be further defined and derived the case of EO's. For instance, a very powerful operator would be the inference, which is not at all the case of GP's;
- the genetic operators do not deal with semantics in the case of GP.

The evolutionary ontologies have been applied for automatically generating scenes. The concept have been applied on a specific ontology built in this scope. The advantage of using scene for researching evolutionary ontologies is that the ontologies as well as the results of the genetic operators have a visual equivalent.

REFERENCES

[1] J. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[2] T.R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5(2), pp. 199-220, 1993.

[3] J.R. Koza, *Genetic Programming.* The MIT Press, 1992.

[4] P. Lord, "Components of an Ontology," *Ontogenesis*, http://ontogenesis.knowledgeblog.org/514, 2010.

[5] B. Daum, U. Merten, *System Architecture with XML.* Elsevier Science, 2003.

[6] M. Obitko, *Ontologies and semantic web. Translations between Ontologies in Multi-Agent Systems*, PhD thesis,Faculty of Electrical Engineering, Czech Technical University in Prague, 2007.

[7] J. Wang, Z. Ding and C. Jiang, "GAOM: Genetic Algorithm based Ontology Matching," in Proc. of IEEE Asia-Pacific Conference on Services Computing, 2006.

[8] V. Qazvinian, H. Abolhassani, S.H. Haeri and B. Hariri, "Evolutionary coincidence-based ontology mapping extraction," *The Journal of Knowledge Engineering*, vol. 25(3), pp. 221-236, 2008.

[9] J. Martinez-Gil, E. Alba and J. Aldana-Montes, "Optimizing Ontology Alignments by Using Genetic Algorithms," in Proc. of Workshop on nature based reasoning for the semantic Web, Karlsruhe, Germany 2008.

[10] J.M.V. Naya, M.M. Romero, J.P. Loureiro, C.R. Munteanu and A.P. Sierra, "Improving Ontology Alignment through Genetic Algorithms," Soft Computing Methods for Practical Environment Solutions: Techniques and Studies, pp. 241-259, 2010.

[11] M.M. Romero, J.M.V. Naya, F.J. Novoa, G. Vasquez and J. Pereira , "A Genetic Algorithms-Based Approach for Optimizing Similarity Aggregation in Ontology Matching," *Lecture Notes in Computer Science*, vol. 7902, pp. 435-444, 2013.

[12] O. Matei, *Evolutionary Computation: Principles and Practices* Risoprint, Romania, 2008.

[13] N. Hansen, A., Ostermeier and A. Gawelczyk, "On the Adaptation of Arbitrary Mutation Distributions in Evolution Strategies: The Generating Set Adaptation," in Proc. of 6th Int. Conf. on Genetic Algorithms, pp. 57-64, 1995.

[14] J.S. Gero, "Extensions to evolutionary systems in design from genetic engineering and developmental biology," in Proc. of Evolutionary Computation, Vol. 1, 1999.