Software Entities as Bird Flocks and Fish Schools

Giuseppe Scanniello and Ugo Erra
Dipartimento di Matematica Informatica e Economia,
University of Basilicata, Viale Dell'Ateneo, Macchia Romana
{giuseppe.scanniello, ugo.erra}@unibas.it

***Abstract* — In this paper, we present a novel approach based on the distributed behavioral model proposed by Reynolds to simulate animal motion such as bird flocks and fish schools. Our proposal has been used to group similar source code classes and has been implemented in a prototype of a supporting system. The approach and the software prototype have been preliminarily assessed on six open source object-oriented software systems implemented in Java. The results indicate that our proposal is promising in source code comprehension and could be successfully applied in the architecture recovery field.**

Keywords - Comprehension; Distributed Behavioral Model; Reynolds's Model, Reverse Engineering; Software Clustering; Visualization

I. INTRODUCTION

Distributed behavioral models have been widely used for the simulation of aggregate motion of flocks of birds and fishes as well as fire, smoke, clouds, spray, and foam of ocean waves [7], [14]. In the flocking behavior models, each individual agent, called *boid* (bird-oid), is implemented as an independent actor that navigates according to its local perception and a set of local rules. The synchronized aggregated motion of the flock is the result of the dense interaction of the relatively simple local behaviors of each boid considering the neighbors from its fields of view.

A number of approaches have been proposed to support the comprehension of existing software systems (e.g., [1], [2], [6], [10]). To this end, well known clustering algorithms (e.g., K-means [9]) have been widely exploited to group software entities (e.g., source code classes or programs) that are similar with respect to a given concern: structural information [13], or key principles of the software development (e.g., high-cohesion and low-coupling [1]). The new trend in software clustering is to group entities on the basis of their lexical similarity (e.g., [5], [10], [11], [16]).

In this paper, we propose a novel approach to comprehend object-oriented software. This approach is based on the distributed behavioral model proposed by Reynolds [14]. Each software entity (a source code class) represents a boid, while its dense interaction depends on its similarity with the closest entities. The similarity is computed by using lexical information extracted from the source code, namely the source code comment. We explored this idea since developers generally put great care in the choice of the source code comments [5]. To assess our proposal, we have implemented a prototype of a supporting system and have conducted a preliminary empirical evaluation on five open source object-oriented software implemented in Java.

II. THE DISTRIBUTED BEHAVIORAL MODEL

Our approach is based on a customization of the distributed behavioral model proposed by Reynolds [14]. In this section, we first introduce the general formulation of that model and then we detail on how we customized it.

A. Boids and Distributed Behavioral Models

Reynolds [14] proposes a computer distributed behavioral model to simulate animal motion such as bird flocks and fish schools. The model consisted in three simple local rules, which describe the behavior of a boid i according to its current position p_i and velocity v_i and respect to all its neighbors $Neighs(i)$. The behaviors and corresponding forces of the Reynolds model are shown in Figure 1 and can be summarized as follows:

a. *Separation:* a boid avoids collisions with nearby boids. For this rule a separation force f_s is computed as follows:

$$f_s = -\sum_{j \in Neighs(i)} \frac{p_i - p_j}{d(p_i, p_j)^2}$$

where $d(p_i,p_j)$ is the distance between current boid i and all the neighbors j.

b. *Cohesion:* boid moves toward the average position of local boids. This rule is to give an aggregated aspect to a flock and its cohesion force f_c, is computed as follows:

$$f_c = \frac{1}{n} \sum_{j \in Neighs(i)} p_j - p_i$$

c. *Alignment:* a boid goes towards the average heading of the local boids. This rule is to bring the flock toward the same direction and its alignment force f_a is computed as follows:

$$f_a = \frac{1}{n} \sum_{j \in Neighs(i)} v_j - v_i$$

The behavior a boid at a given interaction step is obtained combining the forces of the model. Although each boid has direct access to the information of all the boids, the aggregate motion of a given boid requires that it only reacts to the closer neighbors and to its field of view (an angle measured from its fly direction). The neighbors of the boid i (i.e., $Neighs(i)$) can be identified in two ways: *(I)* all the boids within a given distance measured from the center of

978-1-4799-1457-9/13/$31.00 ©2013 IEEE

the boid are neighbors; *(II)* a fixed number of the closest boids are neighbors (e.g., 7 is an appropriate value to simulate animal motions [3]). For example, Figure 1 shows the distance and the field of view of the boid in the middle.

B. Our Customization

In our proposal, each source code class represents a boid. To model the boid motions, we added the *cluster-cohesion* and *cluster-alignment* behaviors to the Reynolds's model. The cluster-cohesion behavior enables similar boids to stay close to each other or to stay away from boids not similar. The force f_{cc} associated to this behavior, for a given boid i, is computed as follows:

$$f_{cc} = \sum_{j \in Neighs(i)} Sim(i,j) \times Seek(i,j) + (1 - Sim(i,j)) \times Flee(i,j)$$

where the function *Seek*(*i, j*) is used to compute the seeking force between the boids i and j as $(p_j - p_i) - v_i$, while the function *Flee*(*i, j*) indicates the fleeing force between i and j and it is computed as $(p_i - p_j) - v_i$. The function *Sim*(*i, j*) computes a similarity factor between the features vectors associated to the boids i and j. This function is computed by using the features vectors associated to each boid. We use here the cosine similarity between two features vectors:

$$Sim(i,j) = \frac{c_i \cdot c_j}{\sqrt{\|c_i\| \cdot \|c_j\|}}$$

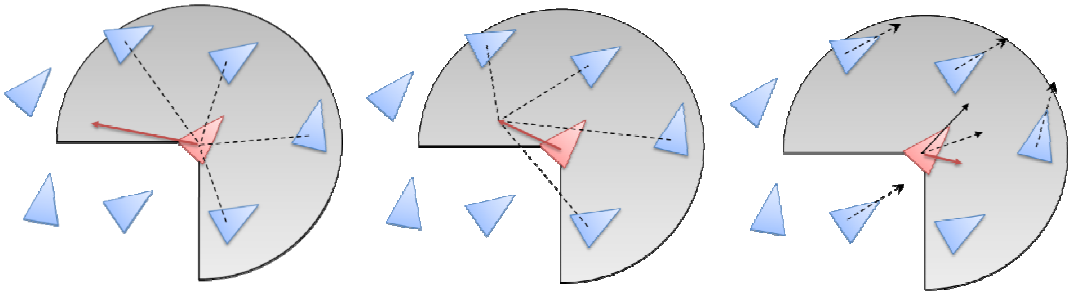
This function is widely used in the information retrieval field [12] to quantify the similarity between two documents (entities in our case) in a vector space. This function assumes values in the interval [0, 1]. The value 0 indicates that the documents are completely different. The *Sim* function returns 1 as the value if the documents are identical. We used the cosine similarity because our goal was to allow boids (i.e., software entities) to stay close if similar or away if different. The definition of different measures could ease the study of different phenomena in existing software. The use of different similarity measures is subject of future work.

The cluster-alignment behavior f_{ca} of a boid i is computed as follows:

$$f_{ca} = \sum_{j \in Neighs(i)} Sim(i,j) \times v_j$$

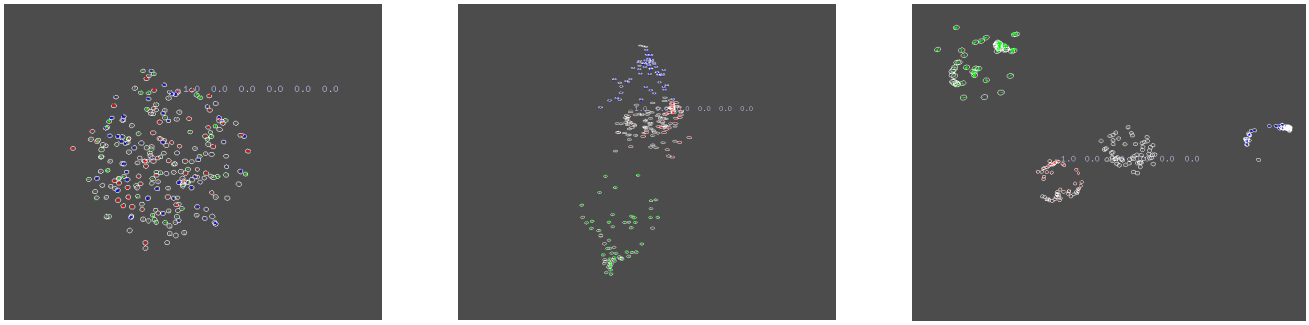
To let boids move and enable the creation of groups based on the similarity function *Sim(i, j)* the forces f_{cc} and f_{ca} are summed with the Reynolds's forces by using vector sum.

Our algorithm groups software entities by projecting them on a three-dimensional space. Figure 2 shows this space as it has been implemented in the 3D environment of our tool prototype. The temporal evolving of the movements of clusters and entities can be followed in this space. In addition, the prototype allows the software engineer to easy retrieve the clusters (see Figure 2.c) as well as to get information on how the boids within each group have been reached the equilibrium (i.e., entities stay close to each other or to stay away and no entities move from a group to another). In case of an impact between a boid and a group, they can either bounce and follow different paths or join in a new group and follow the same path. Boids move in the fixed boundaries of the 3D environment of our tool prototype. In case a boid bumps into the boundary its forward force is reflected.



(a) separation **(b) cohesion** **(c) alignment**

Figure 1. Forces summarizing boids' behaviors (the red arrow represents the force of each behavior)



(a) initial configuration **(b) an intermediate step** **(c) final configuration**

Figure 2. A sample of application of our distributed behavioral model

III. THE APPROACH

Our approach is composed of the following steps:

- *Preprocessing*. Code comment is extracted from source code and preprocessed;
- *Computing Relevance*. A vector of features for each entity is computed;
- *Visualizing and Grouping*. Software entities move in the space according to our customization of the Reynolds's model to form groups.

A. Preprocessing

For each class the comment is extracted and then normalized deleting non-textual tokens (i.e., operators, special symbols, numbers, etc.), splitting terms composed of two or more words (e.g., *last_name* is turned into *last* and *name*) and eliminating all the terms within a stop-word list and with a length less than three characters. The stop-word list also includes the keywords of the Java programming language. The rationale is that code fragments can be present in the comment to explain the usage of a class/method. In this scenario keywords of the language are removed since they should not provide any useful lexical information. A Porter stemmer [12] is also applied. It reduces the inflected (or sometimes derived) terms to their stems (e.g., the words *designing* and *designer* lead to the common radix *design*). This kind of preprocessing is almost common in lexical based clustering approaches (e.g., [5]).

B. Computing Relevance

A term-by-document matrix A is obtained from the normalized source code comments. A generic entry $a_{i,j}$ of this matrix denotes the occurrence of the i^{th} term in the j^{th} document. The used document model is called *bag-of-words* [12] because each document (the comment of a class) is represented as a multi-set of words disregarding all the information about their order or syntactic structure.

For the weight associated to each pair *(term, document),* we consider *term frequency–inverse document frequency*, also known as *tf-idf*, which is defined as follows for any term *t* and document *d*:

$$tf - idf(t_i, d_i) = tf(t_i, d_i) * log \frac{|D|}{df(t_i)}$$

where $tf(t_i, d_j)$ is the number of occurrences of the term i in the document j, while $|D|$ is the total number of documents and $df(t_i)$ is the number of documents containing the term t_i. In case the term t_i appears either in a document or in all the documents (the presence of the term in the document is semantically irrelevant) *tf-idf* is equal to zero. The *tf-idf* indicator assigns higher values to the terms with a high number of occurrences in the document or to the terms that appear in a small number of documents.

C. -Visualizing and Grouping

We associate to each document d_i the features vector c_i, which is obtained by selecting the *i-th* column of the term-by-document matrix computed in the Computing Relevance phase. In this step, source code classes move in the space according to the proposed distributed behavioral. The features vector of all the classes are used for the cluster-cohesion and cluster-alignment behaviors we added to the Reynolds's distributed model.

As default parameter configuration, we used 360° as field of view and 7 as number of the closest neighbors in the clustering [3]. Different values for these parameters are the subject of future work, so understanding whether and how to adapt our approach to the software system under-study. The effect of the different configuration values is immediately observable looking at the motion of the boids. This is what makes our approach different from the greater part of software clustering approaches available in literature.

IV. PRELIMINARY EMPIRICAL EVALUATION

A. Context and Data Set

We implemented a prototype of a supporting system to group classes implemented in Java. This prototype has been implemented in C++ and uses the open-source library OpenSteer [15]. It provides some features to implement artificial intelligence steering behaviors for virtual characters for real-time 3D applications.

Table 1 shows some descriptive of the systems used in our empirical evaluation. The first column shows the software system and the analyzed version. The number of packages is shown in the second, while the number of source files is reported the third columns. The last column reports comment density (i.e., (CLOC / (LOCs + CLOC)) %)

Table 1. Descriptive Statistics

Software System	# pack.	#Files	Density
EasyMock	3	12	30.4%
JabRef	50	533	23%
JavaGroups	18	346	17.8%
Jedit	25	934	28.1%
JUnit	19	103	29.3%
JVlt	19	345	3.3%

B. Results and Discussion

According to the preliminary nature of the study, we studied the clusters identified from a qualitatively perspective. In particular, we observed the following two points that need further future investigations:

- The approach seems conservative. This means that the correctness of the clusters is preferred to the completeness;
- We expected on software systems with a low comment density worse clustering results. This was not actually true. In fact, we observed that on JavaGroups and JVlt (the systems with the lower comment density) the identified clusters contained classes that were similar.

While executing the approach on the selected software systems, we observed that the boids explore the surrounding environment to look for similar boids. We also observed that 2000 iterations suffice to reach a stable state, thus indicating that the algorithm very quickly becomes stable. Finally, to identify the clusters of the largest software system only few seconds were needed.

C. Threats to Validity

Due to the preliminary nature of the investigation presented here, some threats may be present. For example, we did not consider objective criteria to assess the clusters. Therefore, we plan to employ experimental strategies commonly used in literature (e.g., [4], [5], [17], [20]). Otherwise, measures from the information retrieval theory (e.g., precision, recall, and F-measure) may be adopted.

Another issue concerns the size of the used software systems. Larger software system should be employed to increase our awareness on the effectiveness of the clustering algorithm. The use of open source software systems could also threaten the validity of the achieved results.

Finally, we did not perform a comparison between our approach and other clustering based approaches. This can be considered acceptable due to the kind of our contribution: New Ideas and Emerging Results.

V. Final Remarks

Clustering based approaches play an important role in software maintenance and evolution [8], [10], [13], [19], [20]. In this context, we proposed a new approach to study object-oriented. This approach is unsupervised and aims to group software entities that are similar from the lexical perspective. To assess our approach, we have implemented a prototype of a supporting system implemented in C++. This prototype and our underlying approach have been preliminarily assessed on six open source software systems implemented in Java.

One of the most remarkable advantages resulting from the application of our proposal is that the equilibrium of the software entities can be forced if needed interacting with the prototype. For example, this could be useful to understand the strength of the equilibrium of the entities within each cluster. Another advantage is that the number of clusters to recovery is not a priori specified. Finally, our distributed behavioral model can be easily implemented on GPU (Graphical Processor Unit) [21] to improve its scalability and to reduce energy consumption (e.g., [18]).

To increase our confidence in the obtained results, we plan to assess our proposal on commercial software systems. Further on, we plan to verify whether the approach may be adapted to manage software entities at different granularity level (i.e., at the method level rather than at the class level). The extension of our solutions to analyze software implemented in object oriented programming languages different from Java (e.g. C++ and C#) is another possible direction for our work.

References

[1] P. Andritsos and V. Tzerpos, "Information-Theoretic Software Clustering". In *IEEE Trans. Softw. Eng*. vol. 31, no. 2 pp. 150-165.

[2] N. Anquetil and T.C. Lethbridge "Experiments with Clustering as a Software Remodularization Method", In *Proc of Working Conference on Reverse Engineering*, IEEE CS Press, 1999, pp. 235-255.

[3] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic, "Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study," In *Proc. of the National Academy of Sciences*, vol. 105, no. 4, 2008, pp. 1232-1237.

[4] R. A. Bittencourt and D. D. S. Guerrero "Comparison of Graph Clustering Algorithms for Recovering Software Architecture Module Views", In *Proc of European Conference on Software Maintenance and Reengineering*, IEEE CS Press, 2009, pp. 251-254.

[5] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello " Investigating the Use of Lexical Information for Software System Clustering". In *Proc of European Conference on Software Maintenance and Reengineering*, IEEE CS Press, 2011, pp.35-44.

[6] A. De Lucia, M. Risi, G. Scanniello, and G. Tortora "An Investigation of Clustering Algorithms in the Comprehension of Legacy Web Applications" *In Journal of Web Engineering*, vol. 8, no. 4, Rinton Press, 2009, pp. 346-370.

[7] C. Hartman, B. Benes. "Autonomous boids". In *Computer Animation and Virtual Worlds*, vol.17, no.3-4, 2006, pp. 199–206.

[8] D. Doval, S. Mancoridis, and B.S. Mitchell, "Automatic Clustering of Software Systems using a Genetic Algorithm". In *Proc of the Software Technology and Engineering Practice*. IEEECS Press, 1999, pp. 73-82.

[9] P.J. Flynn, A. K. Jain, and M. N. Murty, "Data Clustering: A Review", In *ACM Computing Surveys*, vol. 31, no. 3, 1999, pp. 264-323.

[10] A. Kuhn, S. Ducasse, and T. Girba, "Semantic clustering: Identifying topics in source code", In *Information and Software Technology*, vol. 49, no. 3, 2007, pp.230-243.

[11] J.I. Maletic, and A. Marcus, "Supporting Program Comprehension Using Semantic and Structural Information". In *Proc of International Conference on Software Engineering*, IEEE CS Press, 2001, pp. 103-112.

[12] C.D. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[13] O. Maqbool and H. A. Babri, "Hierarchical Clustering for Software Architecture Recovery". *Transaction on Software Engineering*, vol. 33, no. 11, 2007, pp. 759-780.

[14] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model". In *Proc. of Annual Conference on Computer Graphics and interactive Techniques*, Ed. SIGGRAPH '87. ACM, New York, NY, pp. 25-34.

[15] C. Reynolds. Opensteer - steering behaviors for autonomous characters, 2004. http://opensteer.sourceforge.net/

[16] M. Risi, G. Scanniello, and G. Tortora, "Using fold-in and fold-out in the architecture recovery of software systems". In *Formal Asp. Comput*. 24(3), 2012, pp. 307-330.

[17] G. Scanniello, A. D'Amico, C. D'Amico, T. D'Amico "Using the Kleinberg algorithm and Vector Space Model for software system clustering". In *Proc. of International Conference on Program Comprehension*, Washington, DC, USA, 2010. IEEE Computer Society, pp. 180-189.

[18] G. Scanniello, U. Erra, G. Caggianese, and C. Gravino "Using the GPU to Green an Intensive and Massive Computation System" In Proc. of European Conference on Software Maintenance and Reengineering, 2013, IEEE Computer Society Press, pp. 384-387.

[19] T.A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization". In *Proc of the Working Conference on Reverse Engineering*, 1997, pp.33-43.

[20] J. Wu, A. E. Hassan, and R. C. Holt, "Comparison of Clustering Algorithms in the Context of Software Evolution" In *Proc of the International Conference on Software Maintenance*, IEEE CS Press, 2005, pp.525-535.

[21] U. Erra, B. Frola, and V. Scarano "BehaveRT: a GPU-based library for autonomous characters" In *Proc. of the Third international conference on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, 2010, LNCS 6459, pp.194-205.