BRAIN. Broad Research in Artificial Intelligence and Neuroscience,
ISSN 2067-3957, Volume 1, July 2010,
Special Issue on Complexity in Sciences and Artificial Intelligence,
Eds. B. Iantovics, D. Rădoiu, M. Măruşteri and M. Dehmer

ON CONVERTING SOFTWARE SYSTEMS TO OBJECT ORIENTED ARCHITECTURES

Istvan Gergely Czibula and Gabriela Czibula

Abstract. Object-oriented concepts are useful concerning the reuse of existing software. Therefore a transformation of procedural programs to object-oriented architectures becomes an important process to enhance the reuse of procedural programs. Moreover, it would be useful to assist by automatic methods the software developers in transforming procedural code into an equivalent object-oriented one. In this paper we aim at introducing a hierarchical clustering algorithm that can be used for assisting software developers in the process of transforming procedural code into an object-oriented architecture.

Keywords: *software engineering, procedural systems, object-oriented systems, machine learning, clustering*

2000 *Mathematics Subject Classification*: 68N30, 62H30.

1. Introduction

It is well known that software evolution is an inevitable process for software systems. Repeated changes alter the structure of a system, rapidly degrading it and making the system "legacy". Reengineering seems to be a promising approach to upgrade these systems according to the latest technologies [1].

Object-oriented concepts are useful concerning the reuse of existing software. Therefore a transformation of procedural programs to object-oriented architectures becomes an important process to enhance the reuse of procedural programs.

The evolution of some legacy software systems often requires the rewriting of the system into an object-oriented programming language. This activity, especially for large software systems, is difficult and time consuming.

That is why it would be useful to assist by automatic methods the software developers in transforming procedural code into an equivalent object-oriented one.

Unsupervised classification, or clustering, as it is more often referred as, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects [2], being considered the most important *unsupervised learning* problem.

The resulting subsets or groups, distinct and non-empty, are to be built so that the objects within each cluster are more closely related to one another than objects assigned to different clusters. Central to the clustering process is the notion of degree of similarity (or dissimilarity) between the objects.

We have previously introduced in [3] a clustering approach for transforming procedural systems into object-oriented ones. For this purpose, a partitional clustering algorithm, named *kOOS*, was introduced. In this paper we aim at extending the approach from [3] by introducing a *hierarchical agglomerative* clustering algorithm that can be used for re-grouping the entities from an existing software system written in a procedural language. The goal is to obtain a partition of a software system, in which each cluster would correspond to an application class from the equivalent object-oriented system.

The rest of the paper is structured as follows. The clustering approach for assisting developers in the process of transforming software systems written in procedural programming languages into object-oriented systems that we have previously introduced in [3] is described in Section 2. In Section 3 we introduce a hierarchical clustering algorithm for transforming procedural systems into

object-oriented ones. Section 4 outlines some conclusions of the paper and also indicates further research directions.

2. A clustering approach for OO transformation. Background

We have previously introduced in [3] a clustering approach for transforming procedural systems into object-oriented ones. In the following we will briefly describe the proposed approach.

Let $S = \{s_1, s_2, ..., s_n\}$ be a non object-oriented software system, where $s_i, 1 \leq i \leq n$ can be a subprogram (function or procedure), a global variable, a user defined type.

In order to transform S into an object-oriented system, we have proposed an approach consisting of two steps:

1. **Data collection** - The existing software system is analyzed in order to extract from it the relevant entities: subprograms, local and global variables, subprograms parameters, subprograms invocations, data types and modules, source files or other structures used for organizing the procedural code.

2. **Grouping** - The set of entities extracted at the previous step are grouped in clusters. The goal of this step is to obtain clusters corresponding to the application classes of the software system S.

3. A Hierarchical Clustering Algorithm for OO Transformation

In our clustering approach, the objects to be clustered are the entities from the software system S, i.e., $\mathcal{O} = \{s_1, s_2, \ldots, s_n\}$. Our focus is to group similar entities from S in order to obtain groups (clusters) that will represent classes in the equivalent object-oriented version of the software system S.

In order to express the dissimilarity degree between the entities from the software system S, we will use an adapted generic cohesion measure [4]. Consequently, the distance $d(s_i, s_j)$ between two entities s_i and s_j is expressed as in Equation (1).

$$d(s_i, s_j) = \begin{cases} 0 & if \quad i = j \\ 1 - \frac{|prop(s_i) \cap prop(s_j)|}{|prop(s_i)| + |prop(s_j)|} & if \quad prop(s_i) \cap prop(s_j) \neq \emptyset \\ \infty & otherwise \end{cases}, \qquad (1)$$

where, for a given entity $e \in S$, $prop(e)$ defines a set of relevant properties of e, expressed as follows.

- If e is a subprogram (procedure or function) then $prop(e)$ consists of: the subprogram itself, the source file or module where e is defined, the parameters types of e, the return type of e if it is a function and all subprograms that invoke e.
- If e is global variable then $prop(e)$ consists of: the variable itself, the source files or modules where the variable is defined, all subprograms that use e.
- If e is a user defined type then $prop(e)$ consists of: the type itself, all subprograms that use e, all subprograms that have a parameter of type e and all functions that have e as returned type.

Our distance, as defined in Equation (1), highlights the concept of cohesion [5]. It is very likely that entities with low distances will be placed in the same application class, and distant entities will belong to different application classes.

HOOS is based on the idea of hierarchical agglomerative clustering, and uses an heuristic for merging two clusters. We use *average link* as linkage metric, consequently we will consider the distance $dist(k, k')$ between two clusters $k \in \mathcal{K}$ and $k' \in \mathcal{K}$ as given in Equation (2).

$$dist(k, k') = \frac{1}{|k| \cdot |k'|} \cdot \sum_{e \in k, e' \in k'} d(e, e') \tag{2}$$

The heuristic used in *HOOS* is that, at a given step, the most two similar clusters (the pair of clusters that have the smallest distance between them) are merged only if the distance between them is less or equal to a given threshold, $distMin$. This means that the entities from the two clusters are close enough in order to be placed in the same cluster (application class).

The main steps of *HOOS* algorithm are:

- Each entity from the software system is put in its own cluster (singleton).
- The following steps are repeated until the partition of methods remains unchanged (no more clusters can be selected for merging):

- select the two most similar clusters from the current partition, i.e, the pair of clusters that minimize the distance from Equation (2). Let us denote by *dmin* the distance between the most similar clusters K_i and K_j;
- if $dmin \leq distMin$ (the given threshold), then clusters K_i and K_j will be merged, otherwise the partition remains unchanged.

In our approach we have chosen the value 1 for the threshold, because distances greater than 1 are obtained only for unrelated entities (Equation (1)). Each cluster from the resulted partition will represent an application class from the equivalent object-oriented version of the software system S.

In order to experimentally validate the proposed clustering algorithm, we have considered a simple code example written in Pascal and the algorithm successfully provided the equivalent object-oriented code.

4. Conclusions

We have presented in this paper a hierarchical agglomerative clustering algorithm (*HOOS*) that can be used for assisting software developers in transforming procedural software systems into object-oriented ones.

In comparison with existing approaches for transforming procedural systems into object-oriented architectures [6, 1, 7], the approach proposed in this paper offers an automatic method for the transformation process and heuristically determines the number of application classes candidates.

Further work will be done in the following directions:

- To improve the *distance* function used in the clustering process.
- To extend our approach in order to also determine relationships (class hierarchies) between the application classes obtained in the object-oriented system.
- To apply *HOOS* algorithm on large software systems.
- To apply other search based approaches for transforming procedural software systems into object-oriented ones.

Acknowledgement

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 2286/2008.

References

[1] Cobo, H., Mauco, V., Romero, M.d.C., Rodríguez, C.: A tool to reengineer legacy systems to object-oriented systems. In: ER '99: Proceedings of the Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, London, UK, Springer-Verlag (1999) 186–197

[2] Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)

[3] Czibula, I.: A clustering approach for transforming procedural into object-oriented software systems. In: KEPT '09: Proceedings of the Knowledge Engineering: Principles and Techniques Conference. (2009) 185–188

[4] Simon, F., Loffler, S., Lewerentz, C.: Distance based cohesion measuring. In: Proceedings of the 2nd European Software Measurement Conference (FESMA), Technologisch Instituut Amsterdam (1999)

[5] Bieman, J.M., Kang, B.K.: Measuring design-level cohesion. Software Engineering **24** (1998) 111–124

[6] Zou, Y., Kontogiannis, K.: Incremental transformation of procedural systems to object oriented platforms. In: COMPSAC '03: Proceedings of the 27th Annual International Conference on Computer Software and Applications, Washington, DC, USA, IEEE Computer Society (2003) 290

[7] Jacobson, I., Lindström, F.: Reengineering of old systems to an object-oriented architecture. SIGPLAN Not. **26** (1991) 340–350

Istvan Gergely Czibula and Gabriela Czibula
Department of Computer Science, Babeş-Bolyai University
1, M. Kogalniceanu Street, Cluj-Napoca
email:{*istvanc, gabis*}*@cs.ubbcluj.ro*