[*]RE-ENGINEERING TOWARDS OBJECT-ORIENTED ENVIRONMENTS: THE TROOP PROJECT

Oreste Signore - Mario Loffredo

SEAL (Software Engineering and Application Laboratory)
CNUCE - Institute of CNR - via S. Maria, 36 - 56126 Pisa (Italy)
Phone: +39 (50) 593201 - FAX: +39 (50) 904052
E.mail: oreste@vm.cnuce.cnr.it

Abstract - Software re-engineering and object orientation are two areas of growing interest in the last years. However, while many researchers have focused their interest in the object-oriented design methodologies, a little attention has been paid to the re-engineering towards an object-oriented environment. In this paper we examine the motivations towards object-oriented re-engineering (extendibility, robustness and reusability of the code) and the related problems, due to the difficulty of moving from a process-based to an object-oriented perspective. Finally, we describe the general architecture of a tool, named TROOP, that implements the object-oriented re-engineering process.

1.INTRODUCTION

The "maintenance iceberg" and the investments made in the development of the existing DP applications make the rescue of the software patrimony and the reuse of both the design issues and the software two fundamental areas.
The activities concerned with the software reusability, whose aim is to give a solution to the twofold requirement of implementing more extensible and maintainable software systems and rescue existing software patrimony are based on processes of analysis and abstraction.
Moreover, the growing popularity of object-oriented methodologies seems to offer a completely new area of research for software engineering; they promise software systems that are more easily maintainable and better documented than systems developed using more traditional methodologies. For this reason, we have identified Object-Oriented Re-Engineering as an important line of research, whose aim is to develop theories and models supporting the re-engineering of existing applications towards programming environments like C++ or Eiffel.
As a conclusion, we can identify three different kinds of re-engineering, depending on their target ([11]):

- *reverse re-engineering*: the target is the system itself, that will be re-documented or re-designed, possibly producing a final version implemented in a different imperative language;
- *reuse re-engineering*: the target is a new system, re-designed reusing knowledge and design elements taken from the previous products, but maintaining the top-down design style;
- *object-oriented re-engineering*: the target is the same application system, however designed according to the object oriented methodology.

2.RE-ENGINEERING TOWARDS AN O-O ENVIRONMENT

[*] This work has been partially supported by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R. (National Research Council), Italy.

2.1 - Motivation

As it has already been pointed out in [8], the object-oriented style makes possible to design software that fulfils several desirable code requirements:

- *Modularity*, by offering a natural support to the decomposition of the entire system into modules (*classes*).
- *Extendibility*, because the inheritance relationship makes easier the reuse of existing definitions and facilitates the development of new ones, and the type polymorphism enables to add new specialisation, without forcing modifications of the entire application.
- *Integrability*, because by means of the encapsulation mechanism the classes can integrate themselves with each other. All the interactions take place only via a well-defined interface, and hide all the implementation details to the rest of the system.
- *Robustness*, as the operators can access and modify uniquely the data pertaining to their class and interfaces act as a filter of the interactions between the classes. As a consequence, we have a reduced number of connections between the various classes.
- *Reusability*, because whenever we declare an instance of a class, we reuse data structures and operators acting on them. In addition, the inheritance supplies at high level the modelling of generalisation and specialisation relationships, at low level the reuse of an existing class as a basis for the definition of a new one.

2.2 - Related work

In spite of its novelty, some relevant work has been yet done by several authors, and we can distinguish two main approaches in the area of the object-oriented re-engineering.
In their paper, Jacobson and Lindström ([5]) suggest to adopt a classical re-engineering process to convert an old system in an object-oriented fashion. After a reverse engineering phase, which allows to identify how the components of the system relate to each other and then create a description of the system at the analysis level, a forward engineering phase takes place, by migrating from a top-down design environment to an object-oriented one. This implies a hybrid Software Life Cycle model ([3]), where we can map the Data Flow analysis models into Object-Oriented Design techniques ([1]). In the whole process, the informal documentation (manuals, requirements' specifications, etc.) is taken into account in order to reconstruct the knowledge about the system functionality.
Liu and Wilde ([7]) concentrate their attention on the methodologies to aid in the design recovery of object-like features of a program written in a non object oriented language. Two complementary methods are proposed, based on an analysis of global data or of data types.
As far as the approach proposed by Jacobson and Lindström is concerned, we notice that, even if in principle the informal documentation may be of relevant importance, in practice it might happen that it is lacking or incomplete or inconsistent and misleading. Therefore, information kept from the informal documentation should be carefully examined and validated.
Liu and Wilde themselves in their paper raise the question if their approach may produce "too big" objects. This is due to the intrinsic characteristics of the proposed methods. In fact they consider as strongly connected procedures and data structures if they are used together, and in this case they identify the set of the data structures as an object and the procedures as methods of this object.
Because of this, we completely agree with the authors about the fact that "a further stage of refinement will be necessary in which human intervention or heuristically guided search procedures improve the candidate objects".

3 - THE TROOP TOOL

TROOP (**T**ool for **R**e-engineering towards **O**bject-**O**riented **P**aradigm) is a tool that addresses the task of re-engineering classical programs towards an object-oriented environment.
In the following, we will describe the general architecture of the tool, the structure of the repository the tool is based upon, the logic of the re-engineering process.

3.1 - Generalities

In our approach, we put much emphasis on the *data*, taking the identification of the relevant data structures as a first step. The motivation is twofold: on one hand, it is easier to perform a reverse engineering on data than on procedures, on the other hand, we may think that if the objects exist, they must be reflected in some data structure. This last point is evident when we are concerned with database applications.
A fundamental aspect is to *capture the semantics* of the existing programs, and this process cannot be accomplished in a completely automatic way, because the access to informal documentation may be necessary, or "tricky" code can make obscure the underlying design issues. As a consequence, a stage of refinement will be necessary in which *human intervention* or heuristically guided search procedures improve the candidate objects.
The general architecture of TROOP is shown in figure 1, which clearly identify the importance of a central repository where the knowledge deduced from the analysis of the existing software and other information sources can be stored.

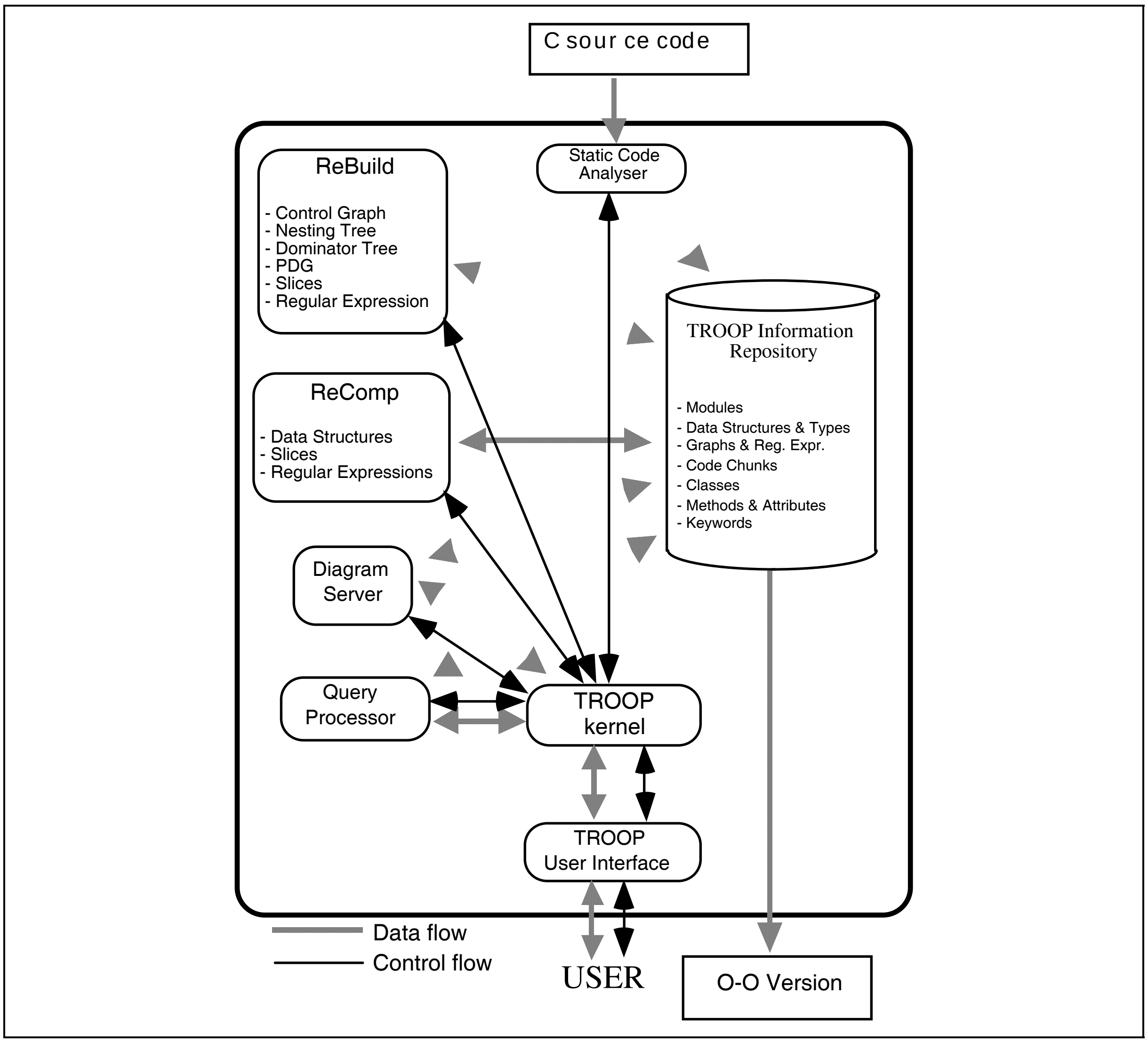

Fig. 1 - The general architecture of TROOP.

The content of the repository will be described in the paragraph 3.3.
The Static Code Analyser is currently under development in the Software Engineering and Applications Laboratory at CNUCE-CNR.
Diagram Server[1] is in charge of displaying graphs described by a formal description, and permits their interactive manipulation by the user. The representations of the code chunks and modules will be generated by the ReBuild (Representation Builder) module which makes use of the algorithms reported in the literature, as described in [12].
The ReComp (Representation Comparator) module basics will be briefly sketched in the next paragraph.
As far as the Information Retrieval aspect is concerned, we will adopt a document vector space model, where each "document" is identified by a set of weighted keywords, selected from a classification scheme ([10]). The classification scheme will be displayed to the user, that will be allowed to navigate through it and choose the right terms, in a way similar to that described in [13].
The user interface will be developed in a windowed environment and will offer some hypertextual capabilities.

[1] Diagram Server is a tool developed in the context of the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo by Dipartimento di Informatica e Sistemistica - Università di Roma "La Sapienza" ([2]).

3.2 - Description of the re-engineering process

The re-engineering process is concerned both with data and procedures. The whole process starts from the static analysis of the data the programs are managing. Subsequently, the procedures are analysed on the basis of the data they are accessing and manipulating. This constitutes a first criterion for the identification of "tentative" methods. In the following we describe in more detail the various steps.

Objects and fields
The very first step consists in the identification of the data structures used by different modules of the existing programs. In this step, we identify the global variables, the record description structures and the data structures that are most used as actual parameters.
When analysing database applications, it is quite easy to identify objects that have been mapped onto database structures. However, the constraints imposed by the relational DBMS force the implementation of repeating attributes as separate tables, and the mapping of complex relationships onto ad hoc link tables. This can lead to the identification of "spurious" objects. The ambiguities must be solved by human intervention.

Methods

1. The modules can be arranged taking into account:
   - their size, expressed as Lines Of Code (LOC);
   - their depth in the calling tree, i.e. their level in the Structure Chart;
   - reuse frequency, i.e. the number of times the module is invoked by other modules, in respect to the total number of calls.

2. On the basis of the variables identified in the object identification phase, we proceed to the *slicing* of the modules ([15], [6]), starting from the modules of limited size, low level in the Structure Chart, and mostly frequently invoked.
   - **2a**. For each connected graph resulting from the slicing process, we consider the possibility of identifying the corresponding code as a *method*. Therefore we assign to it a *name*, a set of *keywords*, the name of the *objects* it is operating on. The keywords are extracted from a faceted classification a browser can graphically display and navigate through. The class is the potential class identified in the object identification phase.
     In addition, we identify the *constraints*, representing them in a semi-formal way (e.g. as a set of pre and post-conditions, assertions, etc.). We are presently considering the pros and cons of the different alternatives.
   - **2b.** Subsequently we proceed to the rebuilding of the program, expressing it by means of the identified components. In this step, a restructuring can take place.
   - **2c.** When considering modules at higher levels in the Structure Chart, attention is paid to the identification of possible cases of generalisation.

Classes and inheritance
In the last step, we consider the similarities among the potential objects and among the potential methods, in order to define classes, def/use relationships and inheritance hierarchies among them.
The similarities among the potential objects can be established on the basis of a type classification based on characteristics like access, scanning and storage methods ([9]).
The similarities among the potential methods will be deduced by comparison of both the slices ([4]) and the regular expressions ([14]) where particular substrings are identified by a single label that gives information about its functionalities. Therefore the different methods

are compared on the basis of both their syntactic structure and their semantics. In this process, we will also make use of the techniques developed in the context of the Information Retrieval area.
The human intervention, that must take place in all the steps, is crucial in this last one.

3.3 - T.I.R.: the TROOP Information Repository

Even if every software engineering tool possesses his own repository, and some large repositories are presently claimed to be available (i.e. IBM Repository or Digital CDD/Repository), it must be noted that they are either tightly coupled with specific tools, or under constant evolution. Therefore, we preferred to implement by ourselves a repository that could be able to accept and manage all the information we need in the re-engineering process. In this sense, our repository may act as an intermediate level repository.
At present stage, we make use of a standard relational DBMS (Sybase for Sun/OS) even if we are also considering the possibility of migration to an O-O or deductive DBMS (ObjectStore, ConceptBase), which should give a better and more coherent support to the management of the items stored in the repository.
In the following we will conform to the terminology adopted by Eiffel ([8]).
A rough graphical representation of the conceptual schema of T.I.R., whose structure is currently in an evolutionary stage, is in figure 2. It is evident that we can identify two main sets of entities, those that describe the classical environment (Programs, Modules, Data structures, Code chunks, etc.) and those pertaining to the object oriented perspective (Classes, Attributes, Methods). The Keywords can be used to characterise both the Code chunks and the Methods.
Some of the entities and relationships are obvious, and will not be described in detail.

The description of some of the most relevant entities and relationships follows.

- *Code chunks*
  Are pieces of code resulting from the slicing process on the modules.
- *Representations*
  Are the representations of the structure of a Code chunk or of a Module, both as a graph (Program Dependence Graph, Nesting tree, Control graph, Dominators tree) and as regular expression.
- *Classes* and *Attributes*
  Are the O-O classes and attributes that have a representation in terms of Data structures in the conventional programs. The two unary relationships involving Classes model the *inheritance* and *use* relationships in the O-O programming environment.
- *Types*
  Model the conventional types (int, char, struct, array, etc.). The unary relationship involving Types model the subtype relationship.

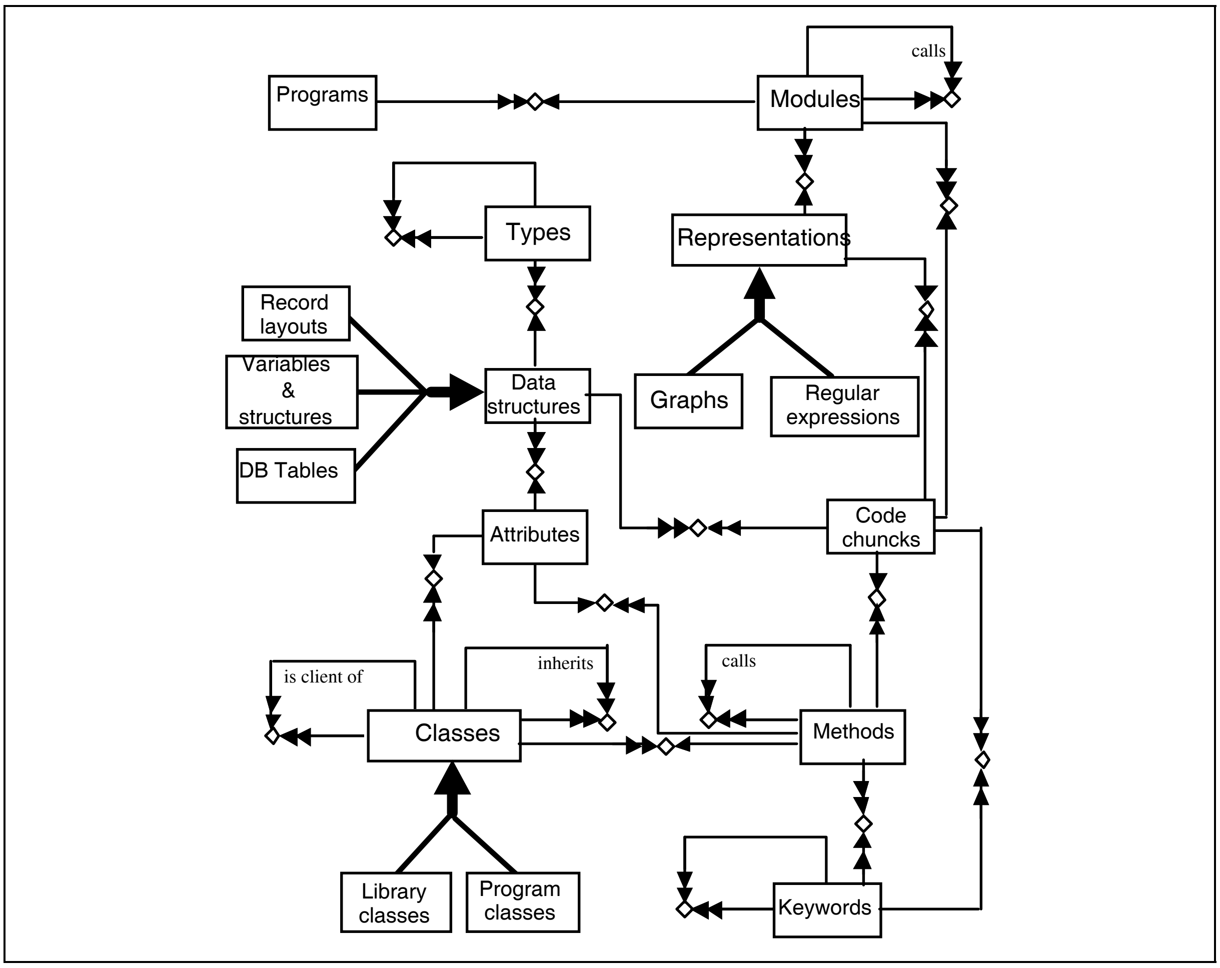

Fig. 2 - The conceptual schema of T.I.R.

4 - Conclusions

Object orientation is claimed to be the most suitable method that can be used to produce software systems that are robust, reliable and reusable.
On the other hand, the existing software patrimony and the related investments are so relevant that it goes without doubt that we have to recover as much as possible of the effort put in the development of the old software systems.
As a consequence, object oriented re-engineering appears a promising research area.
The main difficulty we are faced with when re-engineering old software towards an object oriented environment, is to understand the semantics of the existing code, so that it will be possible to identify objects and methods and, subsequently, classes and inheritance hierarchies among classes.
In this paper, we have presented the architecture of a tool, named TROOP, that implements an object-oriented re-engineering process.
The TROOP main characteristics are the identification of "candidate objects" from the most frequently referred data structures and of the methods via program slicing and clustering by the data that are manipulated. In addition, we define and use in the whole process similarity's criteria among objects and among methods to help the user in the identification of the classes and inheritance hierarchies. TROOP relies on a central repository, that contains information pertaining to both the traditional environment, like modules and data structures, as well as to the object oriented target environment, like classes, attributes and methods. The main steps and the areas where a human intervention is required to solve ambiguous or non automatically decidible cases have been described.

REFERENCES

1. Alabiso M.: “Transformation of Data Flow Analysis Models to Object Oriented Design”, *Proceedings of OOPSLA’ 88*, September 25-30, 1988
2. Di Battista G.: “A Client-Server Architecture for Constructing Diagrammatic Interfaces”, *Proc. of Workshop on Reverse Engineering*, Portici, Naples, Italy, December 11, 1991
3. Edwards J.M., Henderson-Sellers B.: “The Object-Oriented Systems Life Cycle”, *ACM Communications*, Vol. 33, N. 9 (September 1990)
4. Horwitz S., Reps T.: “Efficient comparison of program slices”, *Acta Informatica*, N. 28, pp. 713-732 (1991)
5. Jacobson I., Lindström F.: “Re-engineering of old systems to an object-oriented architecture”, *Proceedings of OOPSLA’ 91*
6. Jiang J., Zhou X., Robson D.J.: “Program Slicing For C - The Problems In Implementation”, *Proceedings of IEEE Conf. on Software Maintenance*, Sorrento (Italy) 1991
7. Liu S-S., Wilde N.: “Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery”, *IEEE Proceedings of IEEE Conference on Software Maintenance*, San Diego, November 26-29, 1990
8. Meyer B.: *Object Oriented Software Construction,* Prentice Hall International (1988)
9. Meyer B.: “Lessons from the Design of the Eiffel Libraries”, *ACM Communications*, Vol. 33, N. 9 (September 1990)
10. Prieto-Diaz R.: “Implementing Faceted Classification for Software Reuse”, A*CM Communications*, Vol. 34, N. 5 (May 1991)
11. Signore O., Loffredo M.: “Reverse, re-engineering e riuso: tre discipline connesse alla manutenzione”, *Technical Report, Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo*, N. 8/34 (October 1992)
12. Signore O., Loffredo M.:”Some issues on Object Oriented Re-Engineering”, *Proceedings of ERCIM Workshop on Methods and Tools for Software Reuse*, Heraklion, Crete, Greece, October 29-30, 1992, pp. 243-265
13. Signore O., Garibaldi A.M., Greco M.: “Proteus: a concept browsing interface towards conventional Information Retrieval Systems”, *DEXA’92 - Database and Expert Systems Application, Proceedings of the International Conference in Valencia, Spain, 2-4 September 1992*, (Tjoa A.M., Ramos I., Eds.) Springer Verlag Wien New York, ISBN 3-211-82400-6, pp. 149-154
14. Wegman M.: “Summarizing Graphs by Regular Expressions”, *Proc. of 10th Annual ACM Symposium of Programming Languages*, Austin Texas, Jan. 24-26, 1983 pp.203-216
15. Weiser M.D.: “Program Slicing”, *IEEE Transactions on Software Engineering*, Vol. SE-10, N. 4 (July 1984)