Extracting Classes from Routine Calls in Legacy Software*

Nicolas Anquetil Timothy Lethbridge
School of Information Technology and Engineering
150 Louis Pasteur, University of Ottawa
Ottawa, Canada, K1N 6N5
(1) (613) 562-5800 x6688 (1) (613) 562-5800 x6685
anquetil@csi.uottawa.ca tcl@site.uottawa.ca

January 12, 1998

Abstract

Extracting object-oriented design from procedural code is an important issue in software maintenance. Existing research in this direction puts a heavy burden on the experts of the system being studied. To try to automate the process, we propose a new method to cluster together routines that are semantically related. The method is based on routine-call analysis. Some experiments on a subset of the system we are studying (23 KLOC) are discussed. They give very promising results.

1 Introduction

While maintaining legacy applications, a large portion of the software engineers' effort is spent in trying to understand the program and data [11]. To help the software engineers in this task, we have built a tool to let them easily browse through the code and find what they are looking for. A component of this browsing tool is an "object-oriented browser": a browser which will present the (procedural) code as a hierarchy of classes.

A number of researchers have tried to migrate procedural code into object oriented code (see for example [4, 9, 10]). This is usually done by choosing some data structure and clustering around it the routines that access it. The result is said to represent a class.

This approach has been reported to be successful in various papers, however our project imposes a number of constraints which set us apart. One of the main difficulties we face is the size of the system (2 MLOC) and the impossibility of relying upon experts of the system to help sort out the true extracted classes.

*This work is supported by NSERC and Mitel Corporation and sponsored by the Consortium for Software Engineering Research (CSER).

In this paper, we propose a new method to cluster semantically related routines together. The method is based on routine calls rather than uses of data structures.

We first present our project, stating the constraints it imposes on us. We then discuss the current practice in program comprehension and contrast it to our project's constraints. Thirdly, we describe a new method of clustering routines and the assumptions on which it is based. We then present and discuss some promising experimental results. We finish with a discussion of related and future work.

2 Our Project

"Does the method scale up?" is a recurring question in program comprehension. A key aspect of our project is to deal with an actual software system in a real world company, to ensure that scaling up is not a problem.

Our goal is to help software engineers to maintain a legacy telecommunication system. The browsing tool we have built contains a variety of different facilities that allow them to navigate the code, search for various things in it and view abstractions of it. All of these facilities, including our "object-oriented browser" are designed to operate rapidly even though the system being studied contains millions of lines of code.

Migrating procedural code to object code is a popular research domain; however the conditions associated with our project create a set of particular constraints:

- Since the software engineers are busy people, we cannot afford to disturb them too often.
- Since we are dealing with an actual system, we must cope with the usual difficulties associated with real world problems (e.g. size, noise in the data).
- Since we aim at building a browser, we need a fully automatic method which could be run periodically to cope with the continuously evolving software.
- On the other hand, because we aim at building a browser, the precision of the generated classes is not as critical as for a real migration project where a key requirement is precise preservation of semantics.

These constraints make the "traditional" class extraction approach ill-suited to our project. We will now present this traditional approach and the problems it raises in our case.

3 Current Practice in Class Extraction

Extracting classes from procedural code has been discussed in several papers (see for example [4, 5, 7, 8, 9, 10]). The common practice consists of choosing a *seed*, which is a data structure, and then clustering "around it" the routines that use this seed. The cluster is said to form a class where the routines are the class's methods. This process is repeated as long there are seeds available. We call this the *seed method*.

The seed may be a structured type (`struct` in C, `record` in Pascal) or a global variable. In the first case (structured type), the routines which use the seed are those with a formal parameter of this type. In the second case (global variable), the routines which use the seed are those that read the variable or write into it. In this case, one usually clusters together more than one global variable and all the routines that access them. Each variable represents a separate attribute of the class.

All the experiments we are aware of which use this method require an expert who understands the system to evaluate the quality of the extracted classes. One of our requirements was that the method be fully automatic. Therefore we require a more "intelligent" method.

Another problem concerns the size of the data. The largest experiments we are aware of that use the seed method deal with 40 KLOC of C [6] or 200 KLOC of Cobol [5] – we are concerned with systems that are substantially larger.

To allow a more sophisticated analysis of the code and to keep computation time low, we propose to split our system into subsystems and to apply the method inside each subsystem. The size of the subsystems we have experimented with only range from 5 KLOC to 23 KLOC which should be small enough to meet our requirements.

However this solution brings another difficulty: since the subsystems are procedural in nature, a "class" is often scattered across different subsystems; the seed data structure is defined in only one subsystem, possibly with the main routines that access it; but other routines that also access it are found in another subsystem.

Applying the seed method purely to this second subsystem will not allow the routines to be clustered around their seed, since the seed is not present in the second subsystem. Hence the seed method will never find the common semantic link among the routines.

We would like this clustering to be possible even in the absence of the seed. To solve this problem and also to try to help in automating the seed method, we will now propose a new method for extracting groups of routines which are semantically related.

4 Clustering Routines on Routine Calls

We are looking for a method to cluster together routines that are semantically related. Unlike the traditional approach to class extraction, this method cannot be based on data uses. A possible answer comes from earlier research we did on class hierarchy redesign [1, 2]. We will now present this solution and the assumption on which it is based. The following section will discuss some experimental results that seem to validate these assumptions.

While doing research on class hierarchy redesign, we discovered that routines calls often carry a semantic meaning. Clustering routines on routine calls would often return semantically coherent clusters. We will now explain how we propose to use routine calls as a clustering base.

A routine calls another one to delegate a part of its job. The caller and the called share a purpose. But this semantic link may be sometimes very weak. For example, a bug-reporting method may call a string-formating function to create a message to be displayed. The link between bug-reporting and string-formating may well be judged not important enough to cluster the two routines together.

We propose a transitive extension of this approach: two routines called by a third one should also share a semantic link. Obviously, this new link would likely be even weaker; however if the same two routines have two, three or more common calling routines, then we can sum the links to create a stronger one.

We therefore propose to say that if two routines are both called by a number of other routines they should be clustered together.

Less obviously, the transitivity should work in the other direction: if two routines both call a third one then they likely share a (very thin) semantic link. And if they both call a number of similar routines, this semantic link could be judged strong enough to cluster the calling routines together.

We have performed some experiments to ascertain the validity of these assumptions.

5 Empirical results

In this section we present and discuss some experiments we have run to discover semantic links between routines. The clustering is based on routine calls, as described in the last section; it would be used to find related routines that would not be detected by the seed method or to validate the results of the seed method without asking an expert.

The first step of the experiment consists of splitting the software into subsystems. In [3] we propose an automatic way to do so, but for these experiments we preferred to use some known subsystems that were identified by the software engineers who maintain the software.

The size of the subsystems range from 5 to 23 KLOC of Pascal. We give here the results for the largest one (45 files and 23 KLOC).

# of children	# rtn pairs	# pairs with seed	% pairs with seed
2	1014	94	9%
3	534	84	16%
4	41	18	44%
5	17	12	71%
6	7	5	71%
7	3	2	67%
8	3	2	67%
9	2	1	50%

# of parents	# rtn pairs	# pairs with seed	% pairs with seed
2	70	7	10%
3	20	3	15%
4	9	0	0%
5	5	0	0%
6	3	0	0%
7	1	0	0%
8	1	0	0%
9	1	0	0%

Table 1: Number of routine pairs that share more than one child (left) or parent (right) in the call graph. The last two columns in each table give the number and percentage of these pairs which are also recognized by the seed method.

For the sake of clarity, calling routines will be named parents (in a call graph) and called routines will be named children. Each experiment includes two parts: we first consider the pairs of routines called by more than one common parent; then we consider the pairs calling more than one common child. We will sometimes refer to these parts as, respectively, the parent experiment and the child experiment.

We first present some statistics on the number of routine pairs that share more than one child (table 1, left) or parent (table 1, right) in the call graph.

In each table, the left most column gives the minimum number of common children or parents that the pairs of routines share. The second column gives the number of pairs having at least that many common children or parents. The third and fourth columns give the number and percentage of pairs that are also clustered by the seed method.

The left table (child experiment) shows that our method compares well with the seed method. Our hypothesis was that the more children two routines share in a call graph, the stronger the semantic link that binds them. This seems to be confirmed by the fact that when the number of common children grows, the percentage of pairs sharing a seed grows too. Assuming that the seed method is a well-founded way to evaluate the effectiveness of our routine-call based method, this means that our method clusters more and more pairs of routines that belong to the same class.

One must not pay too much attention to the last 4 percentages. The number of pairs is so low for these that they loose any statistical significance.

We did not discuss the exact threshold where the semantic link starts to be significant. Here we believe it lies between 2 and 4 common children, but more experiments have to be done to validate this conjecture.

The right table presents the results for the symmetric (parent) experiment. This time, the number of extracted pairs as well as the number of pairs also recognized by the seed method are much lower. To our surprise the results for the child experiment seem better than with the parent experiment.

The first 2 percentages are similar in the child and parent experiments, but then this percentage quickly drops to 0% in the latter case. However, this does not allow one to deduce that the extracted pairs are not valid. For example, the one pair which has more than 9 common parents, actually has thirty! It is very unlikely that 2 routines which are called together by thirty other ones do not share any semantic link, even if they were not clustered by the seed method. We believe that one simply cannot draw any conclusion from the parent experiment.

# of children	# rtn pairs	# pairs with seed	% pairs with seed
2	443	60	14%
3	371	60	16%
4	10	5	50%
5	5	4	80%
6	1	1	100%
7	1	1	100%
8	1	1	100%

# of parents	# rtn pairs	# pairs with seed	% pairs with seed
2	39	3	8%
3	10	1	10%
4	3	0	0%
5	1	0	0%
6	1	0	0%
7	1	0	0%
8	1	0	0%

Table 2: Number of routine pairs that share more than one child (left table) or more than one parent (right table) and do not have any "personal" child or parent.

The next experiment is slightly different in that it not only constrains the pairs to share

more than one child (table 2, left) or more than one parent (table 2, right), but it also forbids them from having any other child or parent. The rationale is that if two routines are always called together by their parents, the semantic link that binds them must be stronger than if they are sometimes used independently. The same reasons apply to routines calling exactly the same children.

This seems to be confirmed in the child experiment (left table). The percentage of pairs clustered by the seed method is higher than in table 1. Once again, one must not pay too much attention to the last results (however good they are) as they are not statistically significant.

The parent experiment does not allow to draw any more conclusions.

Routine pairs		Evaluation
call_reference_is_valid	call_reference_is_valid_for_redirect	ext
execute_del_text_msg	execute_get_text_msg	seed
execute_del_text_msg	execute_put_text_msg	seed
execute_del_text_msg	execute_set_msg_pwd_for_text_msgs	seed
execute_get_text_msg	execute_put_text_msg	seed
execute_get_text_msg	execute_set_msg_pwd_for_text_msgs	seed
execute_put_text_msg	execute_set_msg_pwd_for_text_msgs	seed
execute_initiate_call	q2000cp_send_afc_msg	seed
execute_initiate_call	release_mtce_busy_for_ons	ext
execute_snapshot	q2000_format_and_send_call_status	seed
q2000_display_hash_table	q2000_display_monitors	ext
q2000_feature_interr_monitor	q2000_feature_start_monitor	seed
q2000_feature_interr_monitor	q2000_feature_stop_monitor	seed
q2000_feature_start_monitor	q2000_feature_stop_monitor	seed
q2000_generate_call_status	q2000_generate_key_group_cs_msgs	seed
q2000_send_busy_circuit_req	q2000_send_rts_circuit_req	ext
q2000_send_invoke_reply	q2000_send_pbx_invoke_msg	ext

Table 3: Evaluation of the routine pairs that share more than 5 children in the call graph. In the evaluation column: 'seed' means the pair share a seed in the subsystem; 'ext' means they share a seed external to the subsystem.

Finally, we present more subjective results. We manually evaluated each pair of routines extracted. We are fully aware that to carry any significance this evaluation should be done by an expert who did not take any part in this research – this evaluation will be done later. Before making any appointment with the software engineers however, we wish to have a better understanding of this new method. The experiment would also require that we design a small tool with a proper interface to make the evaluation process as short and easy as possible.

Table 3 presents all the routine pairs that share more than 5 children and table 4 presents all the routine pairs that share more than 3 parents. These thresholds have been chosen to show a small yet meaningful number of pairs (about twenty pairs). However, to allow a

better comparison between tables 3 and 4, the routine pairs that share more than 5 parents in the second one are in italic.

In table 3, most of the pairs of routines would have been clustered by the seed method because they both access a common seed declared in the subsystem considered. All the other pairs access a common seed which is external to the subsystem and thus would not have been clustered by the seed method inside this subsystem. This is one of the primary reasons why we proposed this new clustering method.

Routine pairs		Evaluation
build_feature_access_string	call_reference_is_valid	?
build_feature_access_string	icf_initiate_call	ext
call_reference_is_valid	dispatch_q2000	?
call_reference_is_valid	*format_icf_msg*	*?*
call_reference_is_valid	*icf_generic_invoke_function*	*ext*
call_reference_is_valid	*icf_initiate_call*	*ext*
call_reference_is_valid	q2000_cti_link	ext
call_reference_is_valid	send_recall_message	ext
dispatch_q2000	*format_icf_msg*	*ext*
icf_generic_invoke_function	send_recall_message	ext
display_call_status_details	q2000_get_cr_feature_bits... ..._and_party_swids	seed
display_call_status_details	q2000_get_swid_feature_bits... ..._and_party_swids	seed
q2000_get_cr_feature_bits... ..._and_party_swids	q2000_get_swid_feature_bits... ..._and_party_swids	seed
deactivate_monitor	q2000_hash_swid	ok
find_swid_and_destination_in_chain	q2000_hash_swid	ok
find_target_and_dest_in_chain	notify_database_of_table_changes	?
find_target_and_dest_in_chain	target_monitored	ok
notify_database_of_table_changes	target_monitored	?
q2000_receive_invoke_msg	q2000_send_invoke_reply	ok
q2000_receive_invoke_reply	*q2000_send_invoke_msg*	*ok*

Table 4: Evaluation of the routine pairs that share more than 3 parents in the call graph. The pairs in italic share more than 5 parents. In the evaluation column: 'seed' means the pair share a seed in the subsystem; 'ext' means they share a seed external to the subsystem; 'ok' means the cluster is probably correct, and '?' means we are not sure whether this is correct or not.

Once again, the results seem not as good in the parent experiment (table 4). Many pairs (10 out of 20) do share a seed either internal or external to the subsystem. But for the others, we did not find any seed. This does not mean that the routines do not belong to the same cluster. By studying the comments of the routines, we were sometimes (5 cases) able to see a common purpose. For other pairs we will have to go to an expert.

6 Related Work

Various researchers are concerned with extracting classes from procedural code. However, we are not trying to re-engineer the system, but rather to build an object-oriented browser on it. This sets us apart from most of these researchers:

- Our object model will have to be recomputed often to keep up with the continuously evolving system. This implies that we use a fully automated method.
- On the other hand, we do not need the extreme precision that re-engineering projects require.

Few researchers have tried to cope with as large a system as ours. In [6], Girard also proposed to split the system to reduce its size. He proposes to cluster around a seed only those routines that are declared in the same file as the seed. This approach is similar to ours, however we chose to split the system at the level of subsystems instead of files. Based on our experiments, Girard's heuristic seems valid: a vast majority of the routine clusters we find are located inside a single file.

Possibly for the same reasons as ours, Girard also proposes to cluster routines based on routine calls. He states that two routines called by a third one "should belong to the same module as long as they are semantically related". In his case, the checking is done by an expert.

Since we are seeking an automatic solution, we could not afford this. Instead we decided to rely on a repetition heuristic: two routines will be considered semantically related if they are called by more than one common parent routine. We have not yet ascertained the number of common parents above which this assumption is valid.

Girard also proposes that a routine that has only one caller be clustered with this caller. We agree with him although we did not consider this heuristic in our current research.

Moreover, following this idea, we also propose to cluster together pairs of routines that share more than one parent and are never called independently. Presumably, this would allow to lower the number of common parents required to consider two routines semantically related.

A third heuristic used by Girard consists of clustering together sets of mutually-recursive routines. We did not consider it because there is no recursion in the subsystems we studied.

Finally, we introduce a new idea which is to cluster routines that share some common children. To our surprise, this heuristic seems more promising than the parent-oriented heuristic.

7 Future Work

This research is still in an early stage. Although the results are very promising, there are many questions to answer. We will now summarize some of them.

Routines sharing more than one parent or more than one child in the call graph do seem to be semantically related. However, the precise number of parents or children above which we may confidently cluster them is not yet clear. Moreover, the thresholds seem to

be different for the number of parents and the number of children. Experiments have to be done to try to ascertain these numbers.

We also wish to understand what differences exist between the parent and child heuristics. Given our experiments, one seems to work better than the other, but this may simply indicate a difference in the semantic link elicited. For example, some clusters generated by these methods did not share any seed (structured type or global variable). Nevertheless their comments did show a connection. Can these routines be said to belong to a "class", or does their cluster have another meaning ?

Some other "complementary" heuristics have been proposed: cluster sets of mutually recursive routines, cluster routines which do not have "independent" parents or children. We would like to establish their actual impact on the main heuristics.

Although this was not shown in the discussion in section 5, our method gives dissimilar results for the various subsystems we studied. This may be related to the size of the subsystems, but we need to find the reasons for this difference and establish their importance.

8 Conclusion

We are trying to design an object-oriented browser to help software engineers to maintain legacy software. Trying to extract an object-oriented design from procedural code is a research issue which receives significant interest. The traditional method of doing it (what we call the seed method) consists of choosing a data structure (either a structured type or some global variables) and to use it as a seed around which to cluster the routines which use this seed.

A major problem of this method, for us, is that it requires the intervention of a human expert. Because we will have to recompute our object model often, our project requires that the method be automatic.

We first proposed to use the classical seed method on subsystems instead of the full system. However, this brings in new difficulties as a seed may be declared in a subsystem and the routines accessing it in another one. Therefore these routines cannot be clustered around their seed.

We then proposed to rely on routine calls. The idea is that if two routines are both called by a number of common parents, then they must share some semantic link and we would be justified to cluster them together. The same analysis holds for pairs of routines calling a number of common children.

We performed experiments which tend to prove the two hypotheses are valid. To our surprise, common children seem to be a better sign of semantic link than common parents. However, these are early results, more experiments need to be done to understand more precisely the effects of our method.

Thanks

The authors would like to thank Steve Lyon from Mitel who initiated this research and proved always available to share his experience with us. We are also indebted to all the

members of the KBRE research group for fruitful discussions we have had with them.

The URL for the project is `http://www.csi.uottawa.ca/~tcl/kbre`.

References

[1] N. Anquetil. *Contribution à l'amélioration des modélisations à objets.* PhD thesis, Université de Montréal, 1996.

[2] N. Anquetil and J. Vaucher. Automatic Redesign of Class Hierarchies Breaking Classes into Pieces. Technical Report 1032, DIRO, Université de Montréal, C.P. 6128, succ. Centre Ville, Montréal, PQ, Canada H3C 3J7, 1996.

[3] Nicolas Anquetil and Timothy Lethbridge. Extracting concepts from file names; a new file clustering criterion. In *International conference on Software Engineering, ICSE'98.* ACM SIGSoft, ACM Press, 1998. Accepted for publication.

[4] G. Canfora and A. Cimitile. An Improved Algorithm for Identifying Objects in Code. *Software: Practice and Experience*, 26(1):25–48, jan 1996.

[5] A. Cimitile, A. De Lucia, G.A. Di Lucca, and A.R. Fasolino. Identifying objects in legacy systems. In *5th International Workshop on Program Comprehension, IWPC'97*, pages 138–47. IEEE, IEEE Comp. Soc. Press, 1997.

[6] Jean-François Girard and Rainer Koschke. Finding components in a hierarchy of modules: a step towards architectural understanding. In Mary Jean Harrold and Guiseppe Visaggio, editors, *International Conference on Software Maintenance, ICSM'97*, pages 58–65. IEEE, IEEE comp. soc. press, oct 1997.

[7] Sying-Syang Liu and Norman Wilde. Identifying Objects in a Conventional Procedural Language : An Example of Data Design Recovery. In *Conference on Software Maintenance*, pages 266–71. IEEE Comp. Soc. Press, 1990.

[8] A. De Lucia, G.A. Di Lucca, A.R. Fasolino, P. Guerra, and S. Petruzzelli. Migrating legacy systems towards object-oriented platforms. In Mary Jean Harrold and Guieppe Visaggio, editors, *International Conference on Software Maintenance, ICSM'97*, pages 122–29. IEEE, IEEE Comp. Soc. Press, oct 1997.

[9] Harry M.Sneed. Migration of Procedurally Oriented COBOL Programs in an Object-Oriented Architecture. In *Conference on Software Maintenance*, pages 105–116. IEEE Comp. Soc. Press, 1992.

[10] Hee Beng Kuan Tan and Tok Wang Ling. Recovery of Object-Oriented Design from Existing Data-Intensive Business Programs. *Information and Software Technology*, 37(2):67–77, feb 1995.

[11] S.S. Yau and J.S. Collofello. Some stability measures for software maintenance. *IEEE Transactions on Software Engineering*, 6(11):545–552, nov. 1980.