ROMEO: Reverse Engineering from 00 Source Code to OMT Design

Lantzos Theodoros [1], Helen M. Edwards*, Anthony Bryant' and Neil Willis'

1. Leeds Metropolitan University, UK
{t.lantzos, a.bryant, n.willis} @lmu.ac.uk
2. Sunderland University, UK
h.edwards@sunderland.ac.uk

Abstract.
The Reverse Engineering Method of Object Oriented systems (ROMEO) takes the source code for an existing 00 system and derives a no-loss representation of the system documented in Object-Oriented Modeling Technique (OMT) format. This representation of the system is derived through the use of a series of transformations. This paper describes in detail all the transformation steps needed for the extraction of the Object Design, discusses the experience gained from the application of the method to a case study and outlines the tools that can support the ROMEO methodology.

1. Introduction.

Software maintenance as the most expensive task in software engineering can be assisted by Reverse Engineering (R.E.) technologies, thus reducing the cost of maintenance. Clearly the benefits offered by R.E. technologies attract more practitioners as the problem with software maintenance becomes more substantial,

A way of decreasing the cost of software maintenance is through a practical design recovery method. Design recovery as a major part of R.E. process supplies the system maintainer with a higher representation of the system, an accurate documentation, a well knowledge of what and how the system does [7]. The recovered and documented system design would then provide a path into some systems development method thus assisting the development of a modified or replacement system.

ROMEO method provides a set of transformation rules to allow the full recovery of a system design from its existing source code. ROMEO method has been developed within a specific environment :

- The method takes as its input an existing system implemented in an 00 language
- The deliverable of the method is a no-loss representation of the system design expressed in the Object Modeling Technique (OMT) notation [11].
- The output documentation is in a form that is appropriate for use in a CASE environment.

This paper discusses the ROMEO method [5]. The remainder of the paper is organised as follows: Section 2 describes the rationale for ROMEO method. Section 3 describes the specific environment within which ROMEO has been developed. Section 4 explains the ROMEO method and discusses how the transformations are used to recover the design and convert it to OMT documentation. In section 5 an outline of the use of ROMEO on a case study is presented and section 6 gives an overview of related work.

2. Rationale for ROMEO.

The aim of the project is to develop a method to provide a viable reverse engineering path from existing 00 source code into an OMT environment, where CASE tools can then be used to develop a replacement system. A primary concern of the method is the ability to be applied manually. The following four events or hypothesis provide the rationale for this project:

A. A need for design extraction in 00 systems.

In recent years, the benefits offered by 00 technology has made it one of the leading technologies employed in the software community and a prime

0-8186-8967-6198 $10.00 © 1998 IEEE

candidate for transforming old legacy systems [6, 8, 9, 10, 12, 13]. The subject of implementing old legacy system in 00 technology has been a major concern in R.E. community.

Latest research efforts in the 00 include Design patterns, Frameworks, Components, Agents [14]. The adoption of these technologies raises major concerns about maintaining the first generation of 00 systems. First generation 00 legacy systems already exists and were first mentioned by researchers currently working on the FAMOOS project [9, 10, 15, 16].

Maintaining 00 systems is not an easy task. The essential characteristics - abstraction, inheritance and polymorphism of 00 have offered substantial benefit in terms of reuse, understandability, modeling but at the same time these benefits are not cost-free. The fundamental 00 features add complexity to the process of program comprehension, understanding and - critically - to system maintenance. These issues have been the main concern of many researchers since the early days of 00 [17, 18, 19, 20, 21, 22]. High level problems associated with first generation 00 legacy systems include - absence of documentation, lack of modularity, duplicated functionality, improper layering and low levels problems such as misuse of inheritance, missing inheritance and encapsulation, misplaced encapsulation [10].

Design recovery methods promise to offer a substantial help in the process of re-engineering 00 systems and especially for the capture of the existing system model. Issues like how well a system is documented, what it does and how it works can be addressed by applying design extraction methods. A representation of the system in a higher form makes the existing system more maintainable and more capable for re-engineering technologies.

B. A need for a methodological way for extracting system design manually.

Much research efforts have been devoted for automating the R.E. process. In this endeavour, many data structures and approaches have been identified from which CASE technologies can be developed for automating the process of R.E., design recovery, and assisting program understanding and comprehension. Here, most efforts are expended on studying graphs. Despite this work progress in R.E. there is little or no work done on the methodological implementation of R.E. and especially without the use of CASE tools.

Although, the use of CASE tools are claimed to increase effectiveness in terms of productivity and quality, research results on CASE adoption show paradoxical events. Kemerer [23] reports that one year after introduction, 70% on the CASE tools are never used, 25% are used by only one group and 5% are widely used. Kusters and Wijers [25] found in their analysis of 262 Dutch organizations that in 37% of the organizations, 25% or less of the analysts used CASE tools on a regular basis. Huff [26] gives an explanation that CASE tools are relatively expensive and associated training costs may exceed the original price of the tool. Juhani [24] gives the reasons why CASE tools are not used as participation, management support, training, complexity, compatibility etc.

Culling from these research studies we can see that for one reason or the other, most people partially or do not use CASE tools, thus there is the need for a manual approach to reverse engineering. A practical method for extracting a system design that can be implemented manually could be a very promising help in the area.

C. A dual approach to design recovery (CASE - MANUAL).

A step by step method that provides a set of intermediate forms which at the end will lead to a full design recovery. These forms can adopted CASE technologies where the design extraction process could be automated. The need for a consistent way of work, which independent of the approach adopted (CASE or Manual) should facilitate the adoption of the other i.e. the notion of retracing one's steps.

D. To fill the gap between Comprehension models and Design recovery methods.

Program comprehension theories focus on how people comprehend code but not how they extract the underlying design. Obviously, extracting a system design manually is a an iterative process of comprehending pieces of code. A step by step method for design extraction will subsume the concepts of programming comprehension.

3. The ROMEO development environment.

The project aim is to develop a method to provide a viable reverse engineering path from existing 00 systems into an OMT environment where CASE tools can then be used to develop a replacement system.

3.1: The input to ROMEO.

ROMEO as method for extracting system design of an existing 00 system is not tightly bound to any particular 00 language. To explicate the ROMEO, method we apply it to case studies written in C++. C++ [27] as an 00 programming language is not a truly 00 language but it is widely used language because it extends the C language. Details are provided further in this article about the case study. It should be noted that as a method it isn't restricted to a any particular version of C++.

3.2. The ROMEO deliverable

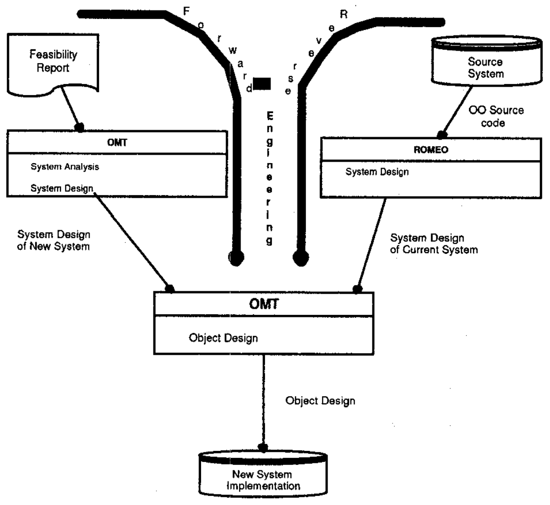
The output from the ROMEO method has been defined as the set of inputs for the Object Design phase of OMT (Object Modeling Technique). OMT [1 1] methodology consists of four stages: Analysis, System Design, Object Design and Implementation. OMT methodology uses three kinds of models to describe a system *object model* describing the objects in the system and their relationships; the *dynamic model,* describing the interactions among objects in the system; and the *functional model,* describing the data transformations of the system. Each model is applicable during all stages of development and acquires implementation detail as development progresses.

3.3. Why the OMT method?

Object Modeling Technique (OMT) was chosen as an output for the ROMEO method because OMT is the main Modeling tool for many 00 development projects. OMT was regarded as the most widely used OOAD method among the rest of 00 methodologies. Furthermore, another reason for this choice is that Unified Modeling Language (UML) which is the latest Modeling language has become a standard in 00 area; and it inherits most of the diagrammatic notations from OMT. Consequently, a system representing OMT design could be transferred to UML specification with little or no effort.

3.4. The relationship between ROMEO and OMT.

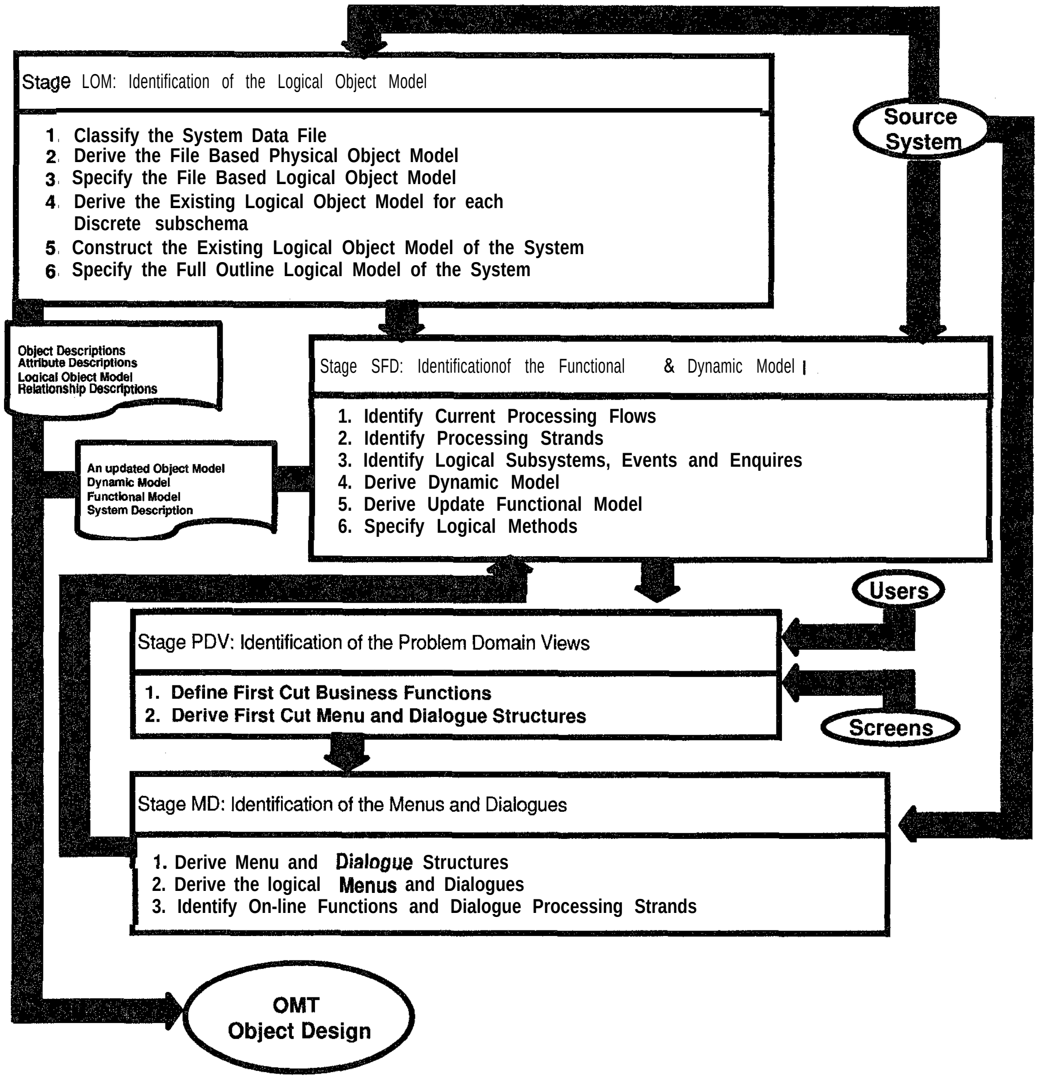
The ROMEO method supplies the software engineer at the Physical Design module of OMT with the same information that would normally be supplied using OMT in a forward engineering environment. The relationship between ROMEO and OMT is illustrated in Figure 1.



"Figure 1. OMT - ROMEO"

4. The ROMEO method.

ROMEO method consist of four discrete stages where each stage includes a number of steps. A full outline of the ROMEO method is illustrated in the Figure 2.


Stage LOM: Identification of the Logical Object Model
1. Classify the System Data File
2. Derive the File Based Physical Object Model
3. Specify the File Based Logical Object Model
4. Derive the Existing Logical Object Model for each Discrete subschema
5. Construct the Existing Logical Object Model of the System
6. Specify the Full Outline Logical Model of the System
Source System
Object Descriptions
Attribute Descriptions
Logical Object Model
Relationship Descriptions
Stage SFD: Identificationof the Functional & Dynamic Model
1. Identify Current Processing Flows
2. Identify Processing Strands
3. Identify Logical Subsystems, Events and Enquires
4. Derive Dynamic Model
5. Derive Update Functional Model
6. Specify Logical Methods
An updated Object Model
Dynamic Model
Functional Model
System Description
Users
Stage PDV: Identification of the Problem Domain Views
1. Define First Cut Business Functions
2. Derive First Cut Menu and Dialogue Structures
Screens
Stage MD: Identification of the Menus and Dialogues
1. Derive Menu and Dialogue Structures
2. Derive the logical Menus and Dialogues
3. Identify On-line Functions and Dialogue Processing Strands
OMT
Object Design

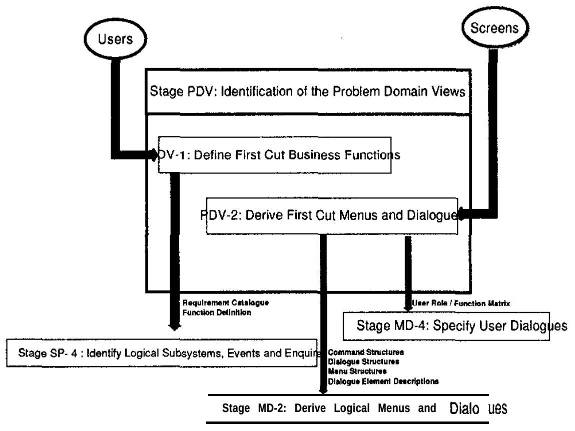
“Figure 2. ROMEO method”

4.1. Stage PDV: Identification of the Problem domain Views:

The first step of ROMEO method studies the problem domain and records information that later used through the application of the method. Figure 3 outlines the processing involved in this stage.

In this step the problem domain area is investigated in order to obtain a detailed explanation about the system through Users specifications and system executions. In the first step of this stage the users are interviewed and details about the user’s job title, activities carry out, input in the system, information returned to the user, frequency of the activities are recorded in the Users Catalogue and Requirements Catalogue. Also, roles of the users, first cut business functions, events and enquiries are identified through the identification of the problem domain views and documented. This information is heavily used in the processing step SFD.

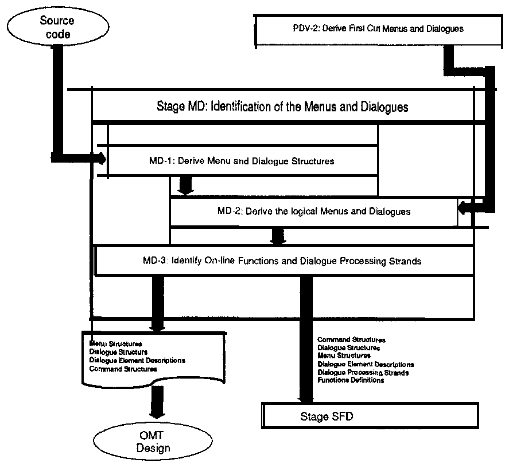
In the second step of this stage program screens and menus are analysed at the execution time and recorded as they appear to the business users. The step is carried out and documented without considering the source code. This step produces a menu and dialogue structure, identifies the menu, dialogues elements - descriptions, and command structures. This information is later used by the stage MD-2. Also, User Role / function Matrix is created that later is forward to stage MD-4.


Users
Screens
Stage PDV: Identification of the Problem Domain Views
DV-1: Define First Cut Business Functions
PDV-2: Derive First Cut Menus and Dialogues
Requirement Catalogue
Function Definition
User Role / Function Matrix
Stage MD-4: Specify User Dialogues
Stage SP- 4 : Identify Logical Subsystems, Events and Enquir
Command Structures
Dialogue Structures
Menu Structures
Dialogue Element Descriptions
Stage MD-2: Derive Logical Menus and Dialo ues

“Figure 3. Stage PDV.”

4.2. Stage MD: Identification of the Menus and Dialogues

This stage is only carried out for those systems that have some elements of on-line processing. The specified output of this stage (Menu structures, Dialogue structures, Dialogue element descriptions and Command structures) are recorded and is forward the step SFD for assisting the process of extraction of Dynamic and Functional model. The structure of this step is illustrated in figure 4.


Source code
PDV-2: Derive First Cut Menus and Dialogues
Stage MD: Identification of the Menus and Dialogues
MD-1: Derive Menu and Dialogue Structures
MD-2: Derive the logical Menus and Dialogues
MD-3: Identify On-line Functions and Dialogue Processing Strands
Menu Structures
Dialogue Structurs
Dialogue Element Descriptions
Command Structures
Command Structures
Dialogue Structures
Menu Structures
Dialogue Element Descriptions
Dialogue Processing Strands
Functions Definitions
Stage SFD
OMT Design

“Figure 4. Stage MD.”

4.2.1. Derive Menu and Dialogue Structures.

In this step, the source code from all application modules is analysed to identify the screen, menu and dialogue definitions. Menu navigation paths are identified by tracing screen references through the system. Program slicing techniques are used to identify input output messages through the system. This step delivers Menu-Dialogue structures, dialogue element descriptions, command structures, dialogue table.

4.2.2. Derive the complete Set of Menus and Dialogues.

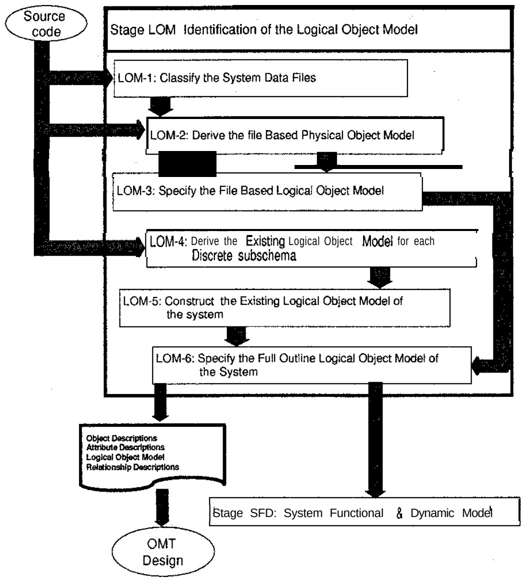
Information derived from steps PDV-2 and MD-l is merged to give a more complete view of the system menus and dialogues.

4.2.3. Identify On-line Functions and Dialogue Processing Strands.

The objective of this step is to identify Dialogue Processing strands. By using program slicing techniques for the processing of each dialogue, this step leads to the identification of dialogue processing strands. Most of the DPS correspond to a business function and to either Enquiry - Update process model.

4.3. Stage LOM: Identification of the Logical Object Model

In this stage two sets of rules for the derivation of an object model are given. One set of rules considers the abstraction of the object model contained within the physical data files of a system; the other set explicitly considers the derivation of the object Model contained within an OODBMS database implementation. Systems accessing both files and database(s), from which the derived object model of each type is merged together by the last step of this stage produces a full object model. An outline of the stage is given in the figure 5.


Source code
Stage LOM Identification of the Logical Object Model
LOM-1: Classify the System Data Files
LOM-2: Derive the file Based Physical Object Model
LOM-3: Specify the File Based Logical Object Model
LOM-4: Derive the Existing Logical Object Model for each Discrete subschema
LOM-5: Construct the Existing Logical Object Model of the system
LOM-6: Specify the Full Outline Logical Object Model of the System
Object Descriptions
Attribute Descriptions
Logical Object Model
Relationship Descriptions
Stage SFD: System Functional & Dynamic Model
OMT Design

“Figure 5. Stage LOM.”

4.3.1: Classify the System Data Files

The data files of the system are classified in order files that contain objects of the system can be identified. Physical data files are classified as either :
A permanent file : files that are created and updated during the execution of the system. Permanent files consist main part of the object model.
A permanent reference file: files that are queried but not updated during the execution the system
A report file : files that are produced as output from the system and never used from the system
A transitory file : files that are generated and used by the system modules but never go beyond the system run.

4.3.2: Derive the File Based Physical Object Model

Based on the classification of files in the previous step, the practitioner here extracts the file based object model from the permanent and permanent reference files. The model is constructed by considering the record contents of these files, which are described in the source code. They key(s) of each object are also required.

4.3.3: Specify the File Based Logical Object Model

The object model from the previous step is further refined in order to produce the Logical Object Model. The refinement process of the file based model is applied in terms of inheritance, generalization and association relationships. The result of this refinement is an 00 logical Object Model.

4.3.4: Derive the Existing Logical Object Model for each Discrete subschema
4.3.5: Construct the Existing Logical Object Model for the Combined Subschemas.

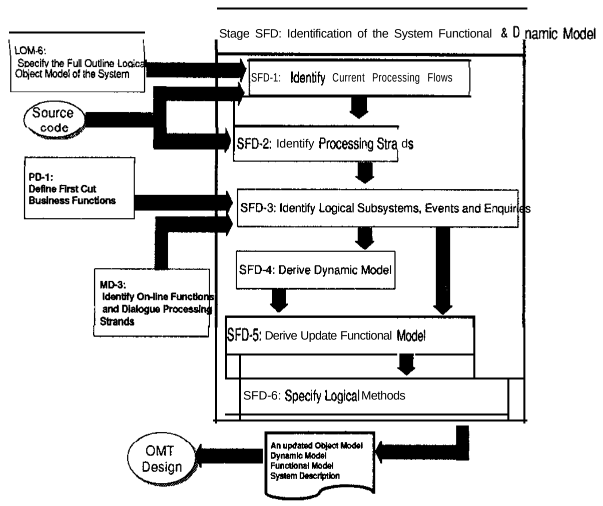
This two steps are dedicated to extracting the object model from the OODBMS database. Each module that accesses an OODBMS database the subschema and schema used are identified. Each object of the subschema becomes an object in the Object model.

4.3.6: Specify the full Outline Logical Object Model of the system

This step brings together all the descriptions of the Object Model into one full, consistent model.

4.4. Stage SFD: Identification of the System Functional & Dynamic Model

The SFD is the most iterative step in ROMEO method. The output of this stage is an updated Object model, a Dynamic Model and a Functional Model. All the steps in this stage are iterated until a final result is achieved. This stage is illustrated in figure 6.


Stage SFD: Identification of the System Functional & D namic Model
LOM-6: Specify the Full Outline Logical Object Model of the System
Source code
PD-1: Define First Cut Business Functions
MD-3: Identify On-line Functions and Dialogue Processing Strands
SFD-1: Identify Current Processing Flows
SFD-2: Identify Processing Stra ds
SFD-3: Identify Logical Subsystems, Events and Enquiries
SFD-4: Derive Dynamic Model
SFD-5: Derive Update Functional Model
SFD-6: Specify Logical Methods
An updated Object Model
Dynamic Model
Functional Model
System Description
OMT Design

"Figure 6. Stage SFD"

4.4.1. Identify current Processing Flows

In this step the source code of the system is analysed in order to identify system modules. System modules are sythesised to compose subsystems. Dependencies among module and subsystems are identified. Output elements from the stage MD are used in this step to identify module dependencies. Inheritance structure form and Candidate Method binding table is created through the source code analysis. By using the Candidate Method binding table for each overloaded method the real bind methods are identified.

4.4.2. Identify Processing Strands

The detailed processing within the subsystems is abstracted using program slicing techniques. Slicing technique such as those described in [497, 460, 236, 244] are adopted. Following program slicing the processing for each class is followed through the system. This leads to the identification of the objects processing strands with each associated operations. Method binding table is then updated to be accurate.

4.4.3. Identify Logical Subsystems, Events and Enquiries.

The business functions defined by the user in the step PDV-1 are compared with the physical subsystems derived in the step SFD-1 and the logical subsystems are isolated. The logical events and enquires in the system are identified from the user input.

4.4.4. Derive Dynamic Model

Information derived from SFD-2 and SFD-3 is merged together to build the state diagrams. During this process more information is discovered to merge the full dynamic model. Here, the OMT representation for state diagrams is used.

4.4.5. Derive Up-date Functional Model.

Functional model gives a full outline of the processes taking place in the system under study. Based on the output of the system and the inputs in the events we identify the set of intermediate processes taking place. Data flow diagrams are used for the representing the system functions.

4.4.6. Specify the Logical Functions.

A full description of the logical functions is created in order to create a mapping from the business function to implementing functions.

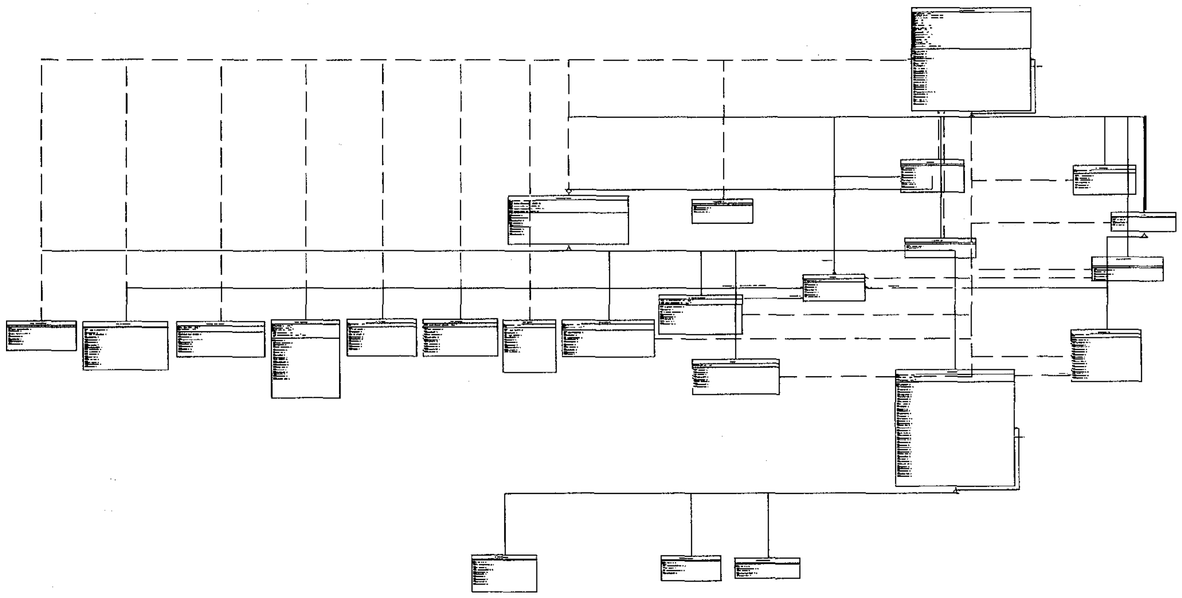
5. CASE Studies

5.1. The file on-line processing system

The ROMEO method has been applied to one case study. In this case study the source system is written in GNU C++ and the application can be compiled from either the GNU compiler or the Sun C++. The system consists of 66000 lines -64K size with 166 classes and 2350 methods. The system accesses one file and it is more of an on-line processing application.

For this case study there was no access to the system users and so no expert knowledge was available. Therefore, the stage PDV-1 could not be applied and the information could only be derived from the system execution and source code itself.

Figure 7 shows the generalisation relationship and uses relationship from a subsystem of the CASE study presented in OMT notation.



"Figure 7. Subsystem of The case study"

Two ways exist to validate the ROMEO method: a) to validate the output based on the initial design of the case study and b) to validate the output based on a market reverse engineering tool.

For this particular case study the access to the initial design was impossible and thus the second approach for validating the method was chosen. The tool that was used for the validation was the Rational Rose CASE tool, which extracted the system design from the case study.

Comparing both outputs, only few things were omitted by applying the method and also it has to be noted that the initial system was not developed to be reversed engineered. Thus the CASE tool couldn't extract the state transition diagrams. So, it was impossible to validate the dynamic aspect of the system.

5.2. Further experiments

This research is still in progress and at the moment further experiments are taking place with ROMEO method. The most important test is the application of the method on another CASE study. In this case the ROMEO method is being applied to a small software system 4k source code written in C++. The same approach for the validation of the first case study will be employed. Furthermore the ROMEO will be tested by other practitioners ▪ applying it independently. Also, further testing is needed for the steps LOM-4 and LOM-5 where OODBMS implementation is used.

5.3. Experience learned

The following experiences can be reported :

- program comprehension elements were very useful for the application of ROMEO method
- Program slicing techniques were heavily used for isolating the relative elements and found to be very promising.
- Studies on the user interface helped a lot for approaching the problem domain and to obtain a useful information about the system structure and furthermore to create a plan of action before deciphering the source code.
- Many difficulties were found with the dynamic aspects of the system. The only way of solving the problem of dynamic binding was through the use of Candidate Method binding table and the test case execution for identifying the real method bind.

5.4. CASE tools requirements

Though the ROMEO method has been designed as a method to be applied manually, it could be very easily be adopted for wide use in bigger applications assisted by CASE tools. First of all, form and structures used by ROMEO could be adopted by a METACASE environment with the ROMEO method becoming an automatic method for extracting systems design where the intermediate levels of representation can be easily monitored. It should be noted that case tools like debuggers, class browser and hierarchy browser can be used as a first aid tool to assist the method.

6. Related work.

In developing the ROMEO methodology we were inspired by the RECAST method [29, 30]. Elements of the ROMEO method regarding stages Problem Domain Views (PDV) and Menu and Dialogues (MD) were adopted and modified from RECAST. ROMEO and RECAST can both be applied manually but RECAST is more dedicated for extracting SSADM notation from COBOL systems. The ROMEO method is generic and thus can be applied to any 00 language. Although there still remains a great deal of work to be done in the area methodological design extraction, much has been done developing CASE tools for 00 systems.

The work done in R.E. of 00 systems is dominated by the Visualisation approach. In this approach the system source code is analysed and program dependencies are stored in a database or the system is represented in an intermediate form as one of the following: Symbol table, Abstract syntax tree, Wide Spectrum Language. Queries in the database or access to the intermediate forms create different views of the program displayed in a Graphical User Interface in order to satisfy the maintainer needs. Examples of program views are dependence graph, call hierarchy, inheritance graph, and so on. Systems using these approaches are Sniff+ [16, 9, 10] , CIA++ [22], OMEGA [28], XREF/XREFDB [17, 18], OO!CARE [1]. Current research efforts focus on the 3D visualisation of existing systems. These systems deal with the static aspect of the 00 systems. On the other hand research efforts in [2, 28] deals with capturing the dynamic aspects of the system during its execution.

7. Conclusions

Applying R.E. manually without the support of CASE tools is a painful and time-consuming task. However this is a reality in many software environments. Methodological design extraction is an approved approach for assisting the manual implementation of R.E.. Comprehension theories and Program slicing techniques do not sufficiently aid in extracting a full system design. Extracting a system design manually is a an iterative process of comprehending pieces of code. Maintainers need a complete step by step method for extracting the system design. The ROMEO method addresses these concerns. However, the ROMEO method cannot be applied blindly, this means that all the stages with their sub-steps are iterative until a final OMT object design is derived.

In this paper we presented the rationale for creating the ROMEO method, the outline of the method and the experiences gained from applying it to a CASE study. The ROMEO method is called to play a substantial role in the area of 00 design extraction and can adopt CASE technology. Besides the methodological design extraction, the ROMEO method could be easily used as an approach to test the outcomes of current R.E. CASE tools.

8. Acknowledgements

This research is supported by a HEFCE research fund. The authors wish to thank Anthony Kesterton from Rational Rose, Natwest insurance company, Dr. Margie Pierce and Gilbert Owusu for their contribution to this work.

9. References

[1] Linos, P.K., and Courtois, V., A tool for understanding Object-Oriented Dependencies, Proceedings of the 3rd IEEE workshop on Program comprehension, 1994, pp.: 20-27

[2] Koskimies K. and Mossenbock H., Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs, International Conference on Software Engineering 1996, pp. 366-375.

[3] Larsen L. and Harrold M.J., Slicing Object-Oriented Software, International Conference on Software Engineering 1996, pp. 495-505

[4] Weiser, M. (2), Program Slicing,. 5th International Conference on Software Engineering, San Diego 1981, March 9-12, pp. 439-449

[5] Lantzos T., A Reverse Engineering Methodology for Object Oriented Systems, OOPSLA '97 Doctoral Symposium, October 1997

[6] Lantzos T., Bryant A. and Edwards H.M., Methodological Reverse Engineering and its contribution to software Reuse, Proceedings of BITWorld, Feb 25-27 1998 Delphi India

[7] Chikofsky, E.J., and Cross, J.H. II, Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software January 1990, pp. 13 - 17

[8] Booch G., Object-Oriented Development, IEEE Transactions on Software Engineering, vol. 12, No 2, February 1986, p211-221

[9] FAMOOS, ESPRIT Project 21975, Nol, February 1997

[10] FAMOOS (2), ESPRIT Project 21975, No2, July 1997

[1 1] Rambaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W., Object-Oriented Modeling and Design, Prentice-Hall International, 1991

[12] Jacobson I., Griss M., Jonsson P., SOFTWARE REUSE Architecture, Process and Organization for Business Success, ACM Press, 1997

[13] Ian Graham, Migrating to object technology, Addison-Wesley Pub., 1995

[14] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns. Elements of Reusable Object-Oriented Software, ADDISON-WESLEY Publishing, 1995

[15] Taivalsaari A., Trauter R., Casais E., Object-Oriented Legacy systems and Software Evolution, OOPSLA '95 workshop, Addendum to the proceedings, OOPS Messenger Volume 6, Number 4, pp. 180-l 85, October 1995

[16] Casais E., Taivalsaari A., and Trauter R., Object Oriented Software Evolution and Re-engineering Workshop report, OOPSLA '96, October 7, 1996

[17] Lejter M., Meyers S. and Reiss S.P., Support for Maintaining Object-Oriented Programs, IEEE Transactions on Software Engineering, Vol. 18, No 12, Dec. 1992, pp.1045-1052

[18] Lejter, M., Meyers, S., and Reiss, S.P., Support for Maintaining Object-Oriented Programs, IEEE Conference proceedings on Software Maintenance 1991 pp. 171 - 178.

[19] Wilde, N., and Huitt, R., Maintenance Support for Object Oriented Programs, IEEE Conference Proceedings on Software Maintenance 1991 pp. 162 - 170.

[20] Wilde N. and Huitt R., Maintenance Support for Object-Oriented Programs, IEEE Transactions on Software Engineering, Vol. 18, No 12, Dec. 1992, p. 1038-1044

[21] Bischofberger W.R., Kofler T. and Schaffer, Object-Oriented Programming Environments: Requirements and Approaches, Software-Concepts and Tools, Vol. 1.5, No 2, Springer Verlag 1994

[22] Grass, J.E. and Yih-Farn, C., The C++ Information Abstractor, Proceedings of USENIX C++ 1990, Pages 265 - 277

[23] Kemeber C. F., How the Learning Curve affects CASE Tool adoption, IEEE Software, May 1992, p.23-28

[24] Juhani I., Why CASE Tools are not Used?, Communication of the ACM, October 1996, Vol. 39, No 10, p.94-103

[25] Kusters, R.J. ja Wijers, G.M., On the practical use of CASE-tools: results of a survey, IEEE Proceedings of the 6th International Workshop on CASE, Singapore 1993, pp. 2-10

[26] Huff C.C., Elements of a realistic CASE tool adoption budget, ACM Commun. 35, 4 (1992), p.45-54

[27] Stroustrup Bjarne, The C++ programming language, ISBN o-201-88954-2, 1997

[28] Chen X., Tsai W.-T. and Huang H., Omega- An integrated Environment for C++ program Maintenance, International conference on Software Maintenance, 1996, p. 114

[29] Edwards, H.M., and Munro, M., Recast: reverse engineering from COBOL to SSADM, IEEE 1993 Proceedings of Working Conference on Reverse Engineering p44-53.

[30] Edwards, H.M. and Munro, M., Deriving a Logical Data Model for a System Using the RECAST Method, IEEE Proceedings of the 2nd working conference on Reverse Engineering, 1995 p. 126 - 135.