Submitted to:
2nd Workshop on
PROGRAM COMPREHENSION
July 8-9, 1993
Europa Palace Hotel, Capri, Naples-Italy

*Charon: a tool for code redocumentation and re-engineering

Oreste Signore - Mario Loffredo
CNUCE - Institute of CNR - via S. Maria, 36 - 56126 Pisa

Abstract

The maintenance of applications constitutes the most relevant issue of the overall life cycle activities. CASE tools claim to be effective in producing efficient and error free software, but usually the maintainer doesn't want to produce new application systems, but just to modify the existing ones. Re-engineering appears to be a suitable way of getting the advantages of the automated CASE tools, without facing the costs involved in a complete redevelopment of the existing systems, whose specifications are sometimes obsolete and no more corresponding to the actual version of the software.
In this paper, a totally automatic approach towards the reconstruction of the software documentation and possible code re-engineering is presented.
We move from the source code by using a static code analyser and capture information pertinent to higher level design phases that are subsequently imported into the ADW CASE tool.

Keywords: Software reverse engineering - Maintenance - Software documentation - CASE Tools

* This work has been partially supported by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R.

*Charon: a tool for code redocumentation and re-engineering

Oreste Signore - Mario Loffredo
CNUCE - Institute of CNR - via S. Maria, 36 - 56126 Pisa

Abstract

The maintenance of applications constitutes the most relevant issue of the overall life cycle activities. CASE tools claim to be effective in producing efficient and error free software, but usually the maintainer doesn't want to produce new application systems, but just to modify the existing ones. Re-engineering appears to be a suitable way of getting the advantages of the automated CASE tools, without facing the costs involved in a complete redevelopment of the existing systems, whose specifications are sometimes obsolete and no more corresponding to the actual version of the software.
In this paper, a totally automatic approach towards the reconstruction of the software documentation and possible code re-engineering is presented.
We move from the source code by using a static code analyser and capture information pertinent to higher level design phases that are subsequently imported into the ADW[1] CASE tool.

1. Introduction

It is well known that the most relevant part, up to 95% according to some estimates ([1]), of the EDP departments activities is dedicated to the maintenance of applications.
In particular, we have to outline that in the latest years the maintenance is enhancing in order to delete errors, meet new requirements, improve both design and performance, interface new programs, modify data structures and formats, test new hardware and software techniques.
However, it must be noted that maintenance interventions may alter the features of the original software applications, and may therefore substantially contribute to their degrade. In fact, alterations to the data structures, the documentation out of date, the turn over of the personnel in charge of the project (and it is well known that in many cases only the people involved are aware of the real features of the application), make the application system itself less reliable and maintainable, error prone, difficult and expensive to be modified. Application systems presenting poor or even inexistent documentation cause further problems during the maintenance work because the user has to rebuild the knowledge before making any change. In addition, old technology or poor quality code based systems show other problems (low performance, limited integration with other systems, difficulty to test) which decrease the productivity.
This situation faces the development of new application systems and the adoption of the modern CASE tools, which are claimed to be the most effective way of developing new applications, at lower costs and at a higher quality level ([2], [3], [4], [5]). As a

* This work has been partially supported by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R.

1 ADW is a CASE tool implemented by KnowledgeWare (USA) for the OS/2 environment, selected by IBM as one of the International Alliance products for the AD/Cycle platform.

consequence, there is a tendency towards the implementation of tools that can help in understanding, improving and managing existing systems, even if implemented in non CASE environments, in order to recover existing software portfolio and manage it by using CASE techniques.
In this paper, the authors present an automatic approach towards the reconstruction of the software documentation and the re-engineering of code; we collect data from the source code which are, subsequently, manipulated and imported in a CASE tool, thus rebuilding information pertinent to higher level design phases and giving the possibility of implementing of a new engineering phase.

2. The re-engineering

Up to the end of the last decade, users who intended to enhance the features or the performances of their software, had only two alternatives: leave all the software unchanged, or face the costs of a complete re-development. A new alternative, which is presently object of a relevant research effort, is available since the beginning of this decade: the *re-engineering* ([6], [7]). The re-engineering gives the possibility of updating the technology the application system is based upon, without incurring relevant management costs.
More precisely, the software re-engineering consists in the analysis of an existing software, for the sake of giving a more readable structure to it or, perhaps, in order to represent its structure in an alternative form, thus allowing a new, different implementation. This process is normally accomplished in two successive steps: first of all, we analyse the system, capturing information about its components, afterwards, we operate a restructuring, thus getting a more standardised form. In some cases, it could happen that the user intends to operate a transformation of the existing application, either because a different language or a different data management system are to be used, or because an interface towards a different tool must be implemented. In these cases, we can clearly identify two different phases, as depicted in fig. 1. In the first phase we operate a reverse engineering (RE) where we abstract the characteristics of the product that we have to reconfigure. Subsequently, we operate a forward engineering (FE) process, which is in charge of implementing the new product.

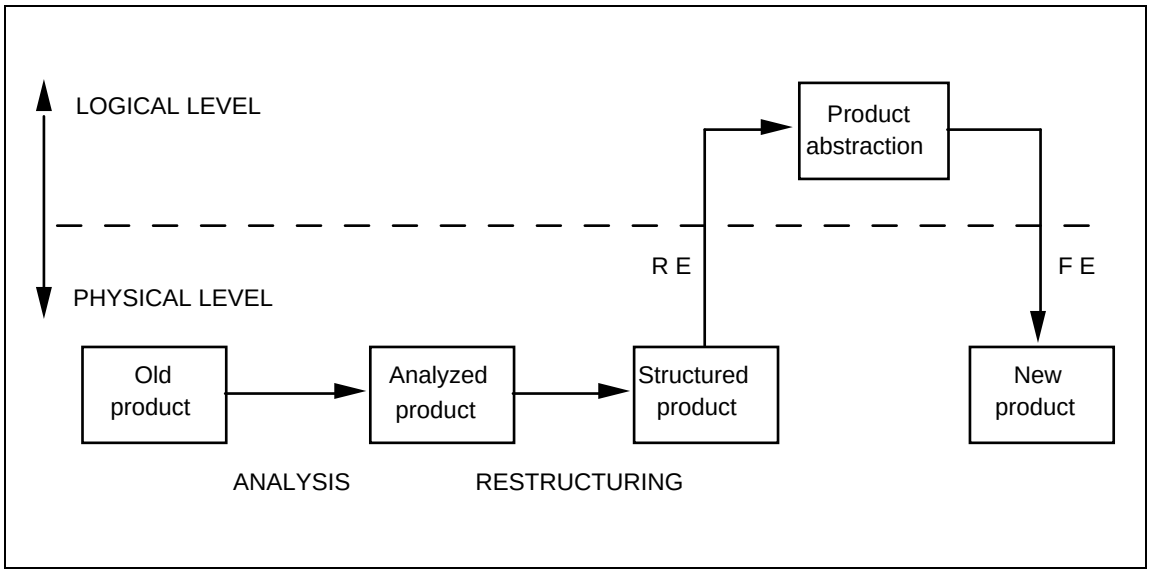


Fig. 1 - The re-engineering process

The reverse engineering phase is the focus of the re-engineering process. Starting from any life cycle level and with the automatic tools' support, such process does not aim at modifying system objects, but only at *improving comprehension* by generating and synthesising independent implementation abstractions.

3. Related work

Recently, the software engineering market has been enriched by the appearance of many reverse engineering and re-engineering tools. These tools allow to solve partially the problem of recovering the design of existing systems by capturing information that are explicitly represented in the source code.
BACHMAN/Data Analyst is part of the BACHMAN Product Set, an integrated collection of CASE tools for maintaining and developing IBM mainframe applications.
This tool permit to reconstruct the E-R model by examining COBOL, IMS, PL/1 data structures and by extracting, via BACHMAN/DBA, information from DB2 or SQL/DS catalog. Thus we can modify the old application at a high level (analysis) and, then, generate a new application by executing a forward engineering step.
In other words, Bachman Product Set enables the maintainer to make an almost completely automatic re-engineering cycle even if it is limited to the system data architecture.
Cscope, for the UNIX environment, is an interactive tool that makes possible to abstract information from a program by cross-referencing various source code entities.
TIBER (Techniques and Instruments to Build Encyclopedias of Redevelopment engineering) is a set of methods, techniques, instruments and know-how through which existing COBOL applications are redocumented, reorganised and restructured inside the ADW repository named “Encyclopedia”.
TIBER analyse COBOL source code, JCL, TP maps (CICS or IMS/DC) and DDL and produce a system documentation composed by relational metamodel, record layouts, structure charts, action diagrams and video maps.

Therefore, such tool can be considered more as a re-systemisation tool rather than a re-engineering one because the number and the quality of the output documents is strictly related to the programming language of the source code analysed.
Furthermore, we want to outline a re-engineering case study towards the IEW[2] CASE tool. This experience consists in an environment which supports the re-engineering of Pascal programs at the design level starting from the information abstracted by the static analyser ATOOL ([8]).

4. Charon[3]: an integration with a CASE tool

Charon is a tool implementing a re-engineering cycle which has a C program, where EXEC SQL statements are embedded, as source and has a different version of the same program, written in COBOL/CICS/DB2 environment, as target.
The reverse engineering step is performed by the static C code analyser C-TOOL[4], the forward engineering one is executed by the ADW COBOL code automatic generator; we are committed to convert the first tool output into the second one proper objects.
C-TOOL evaluates some structural and dimensional metrics and produces the nesting tree and control graph of the program. In addition, the embedded SQL statements are extracted. At the same time, all information about relations accessed by the SQL statements are exported from the catalog and stored in some supporting tables (IBM Query Manager).
Files and tables contents are processed, and information relevant for the design and analysis phases are collected and stored in an appropriate format, compatible with the import/export format of the ADW. Such information are also inserted in the SQL supporting tables, making possible the implementation of the presented approach in other development environments (StP, Bachman).
After the creation of an empty encyclopedia related to the project and the subsequent import phase of the *Charon* output, the user finds the extracted information represented by most of the diagrams of the Analysis and Design workstation.

---

2 IEW (Information Engineering Workbench) is the MS-DOS version of ADW.

3 Charon is a son of Erebus who in Greek myth ferries the souls of the dead over the Styx.

4 C-TOOL is a tool developed by the CRIAI (Consorzio Campano di Ricerca per l' Informatica e l' Automazione Industriale), and has been modified for this worrk in order to intercept the SQL statements. We acknowledge our colleagues from CRIAI for their kind support.

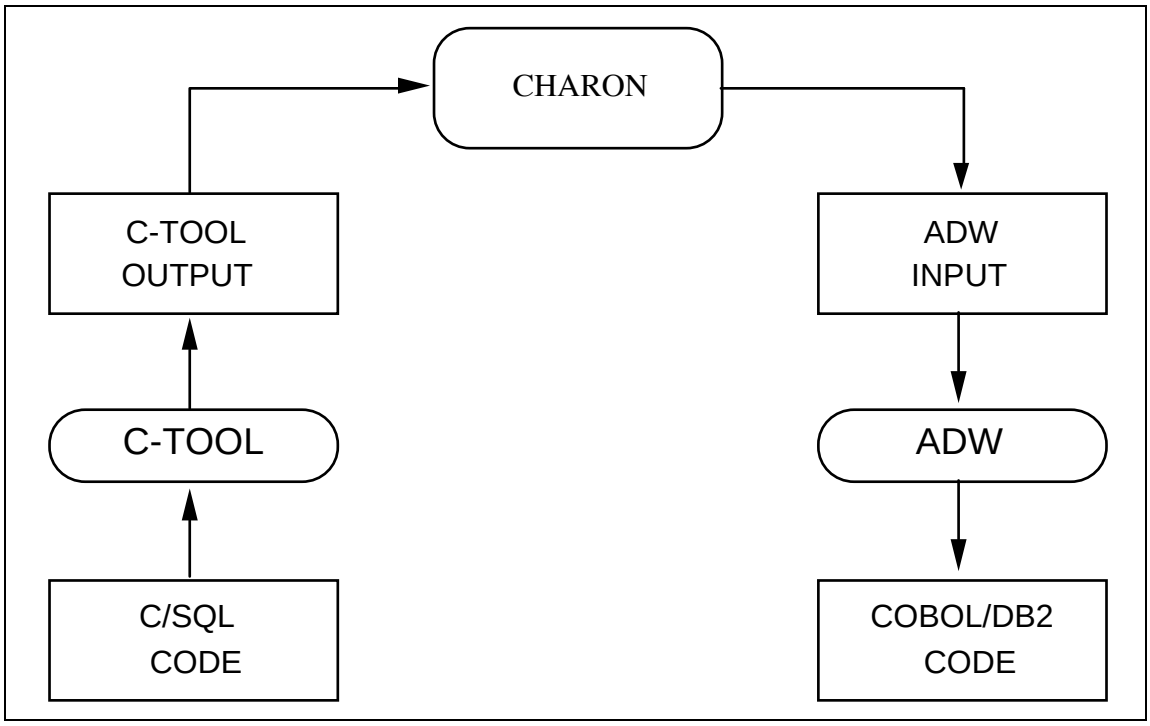


Fig. 2 - The re-engineering cycle implemented by *Charon*

In fact, as far as the data architecture is concerned, the user can access Relational DB Design, Data Structures, Data Types, SQL Statements of the design level, and Entity-Relationship Diagram, Entity Attributes Description of the analysis level.
Concerning the processes architecture, two diagrams of the design level are available: Structure Chart and Action Diagram. They describe respectively the program decomposition in a modules hierarchy and the logic of a module. Therefore the user can modify, add or delete data, so starting a complete redevelopment of the project, with an obvious improvement of the overall software quality.
Starting from the information extracted and imported in the ADW/DWS module, *Charon* also proceed to reconstruct the database conceptual model which belongs to the upper level (AWS).
In Appendix 1 a list of these diagrams and a more detailed description of their functionalities is presented.

5. Reverse engineering through C-TOOL

C-TOOL involves two procedures, subsequently executed, which can analyse a syntactically corrected C program:

- **ANADIM**, which records on the SQL_STAT file all the EXEC SQL commands being in the source C code and their positions;

- **ANASTRU**, which produces the program Call Graph, the Nesting Tree and the Control Flow Graph.

The first graph represents the program decomposition into modules (procedures and functions) also including the main program. An edge exists between two nodes N_1 and N_2 if N_1 calls N_2 in the program.

The Nesting Tree describes the program as a tree in which terminal nodes correspond to the simple instructions (assignment, read, write, call, return) and the intermediate nodes correspond to the control structures: sequence, alternative, repetition. An edge exists between two nodes N_1 and N_2 if N_1 includes N_2 in the program.

The last representation diagram shows the control flow between all the program components. The nodes are of the same type of the Nesting Tree ones. An edge exists between two nodes N_1 and N_2 if N_2 can follow N_1 in a program execution.

These diagrams are presented in a table form in the following files: OUT2, GRAPH and NESTING.

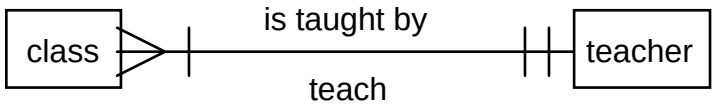
6. The ADW I-CASE workbench

ADW, as most of the CASE tool supporting systems development, is based on the software life cycle Waterfall Model concepts and methodologies. It is modular and includes four workstations, i.e. Planning, Analysis, Design, Construction, which reflect the upper four levels of that model. The user can manage the information, stored in a repository, the "Encyclopedia", by using some structured analysis and design techniques as SADT, DFD, Modular Decomposition, PDL, etc.. In addition, other important characteristics of the ADW information engineering CASE tool consist in the possibility of keeping continuously and automatically the consistency even between a large number of diagrams and of easily navigating through such diagrams.
We'll import in the ADW/Encyclopedia the information included in a set of ASCII files which can be divided into three groups: ".EXP" files, ".ENC" files and "MASC" files.

6.1 - ".EXP" files

They contain records representing all the program information entities as objects (OI.EXP), associations between objects (AI.EXP) and properties of objects and associations (PI.EXP for the short properties and TI.EXP for the long textual properties).
The records of the OI.EXP file have three fields: the first one is for a token (*Instance token*) which uniquely identifies an object in the project encyclopedia, the second one is for the type of the object (*Type code*) and the third one is for the name (*Instance name*) given to the object by the user.
The records of the AI.EXP file have four fields: the first one is for the instance token of the association, the second one is for the type code, the third one is for the instance token of the source object of the association and the last one is for the instance token of the target object.
The properties files include records with the same structure, but the last field in TI.EXP file is longer than the related one in the PI.EXP.
The first field of the records is for the instance token of the object or the association which the property is related to, the second one is for the type code of the property, the

third one is for the row number of the textual properties (*Repetition number*) and the last one is for the value of the property (*Property value*).
In the following, the ADW internal representation of a relationship between two classes is depicted.



Objects
10000000003,10007,"teacher "
10000000004,10007,"class "

Associations
20000000022,20044,10000000004,10000000003

Short properties
20000000022,30034,00000,"is taught by "
20000000022,30037,00000,"teach "
20000000022,30035,00000,"1 "
20000000022,30036,00000,"1 "
20000000022,30035,00000,"M "
20000000022,30036,00000,"M "

Long textual properties
10000000004,30076,00001,"A_group_of_students_which_study_"
10000000004,30076,00002,"the_same_arguments "

6.2 - ".ENC" files

They include the description of the program modules Action Diagrams. Each row in a ".ENC" file contains a record having nine fields:

- the first one identifies the type of diagram simple instruction or control structure,
- the subsequent eight fields concern the diagram layout;
- the last field is for the description of the particular action executed.

In the following, the ADW internal representation for an Action Diagram is showed:

```
$ADTEXT$3.00ENGLISH
T  0000000*SECTION P1-READ-CUSTOMER - WE READ THE CUSTOMER RECORD
   0000100**USING EITHER THE CUSTOMER-NO OR THE CUSTOMER-NAME
B  0000000If CNAME-CUSTOMER-NO NOT = 0
   0000000AND CNAME-CUSTOMER-NO NOT = SPACES
   0000000MOVE CNAME-CUSTOMER-NO TO OPG01-CUSTOMER-NO
D  0000004Customer&oi0003D9%FIND KEY OPG01 USING CUSTOMER-NO
C  0000000Else
   0000000MOVE CNAME-CUSTOMER-NAME TO OPG01-CUSTOMER-NAME
D  0000004Customer&oi0003DA%FIND KEY OPG01 USING CUSTOMER-NAME
E  0000000ENDIF
B  0000000If OPG01-STATUS = SPACES
   0000000MOVE CORRESPONDING OPG01 TO CNAME
```

```
X20000000EXIT
C  0000000Else
   0000000MOVE 'CUSTOMER NOT FOUND' TO CNAME-ERROR-MESSAGE
X20000000EXIT
E  0000000ENDIF
G  0000000
```

6.3 - "MASC" files

These files are used when transferring information from the workbench construction level to the extern, but in our application they hold the masks of the tables SQL DML commands.
These templates are called by the modules through an option named "Using call" and include two groups of records:

- the first group takes into account the general information of the accessed table;
- the second one specifies the SQL command.

In the following, the mask of a SELECT command is presented:

```
67,OPG01CUSTOMER
                    <table creator>      <table creation date>
83,MLOPG01A100000CUSTOMER
83,MLOPG01B100000OPS DB.CUSTOMER
83,MLOPG01C100000DB200TSTOPCUSTSP
83,MLOPG01P100000TV
87,MLOPG01R100000CUSTOMER-NO                          *SELECT
31,MLOPG01R100001B 0000001EXEC SQL
54,MLOPG01R100002B 0000000SELECT            CUSTOMER_NO
53,MLOPG01R100003  0000000                ,     CUSTOMER_NAME
54,MLOPG01R100004  0000000                ,     STREET_ADDRESS
28,MLOPG01R100005E 0000000
61,MLOPG01R100006B 0000000INTO              :OPG01-CUSTOMER_NO
60,MLOPG01R100007  0000000                ,     :OPG01-CUSTOMER_NAME
61,MLOPG01R100008  0000000                ,     :OPG01-STREET_ADDRESS
28,MLOPG01R100009E 0000000
55,MLOPG01R100010  0000000FROM                        OPCUST_TABLE
57,MLOPG01R100011B  0000000WHERE            CUSTOMER_NO =
58,MLOPG01R100012  0000000                    :OPG01-CUSTOMER_NO
28,MLOPG01R100013E 0000000
31,MLOPG01R100014E 0000001END-EXEC
```

6.4 - The import process

The *Charon* output information must be transferred in the ADW environment according to three way of importing (Fig. 3).
The ".EXP" files are imported by running the *Encyclopedia Data Transfer* option of the *File* menu of the *Encyclopedia Services* task in any workstation.

The ".ENC" files must be copied in the ADW subdirectory associated to the empty encyclopedia.
The "MASC" files are transferred by starting the *Transfer* option of the *File* menu of the *Code Generator* task in the Construction workstation (CWS).

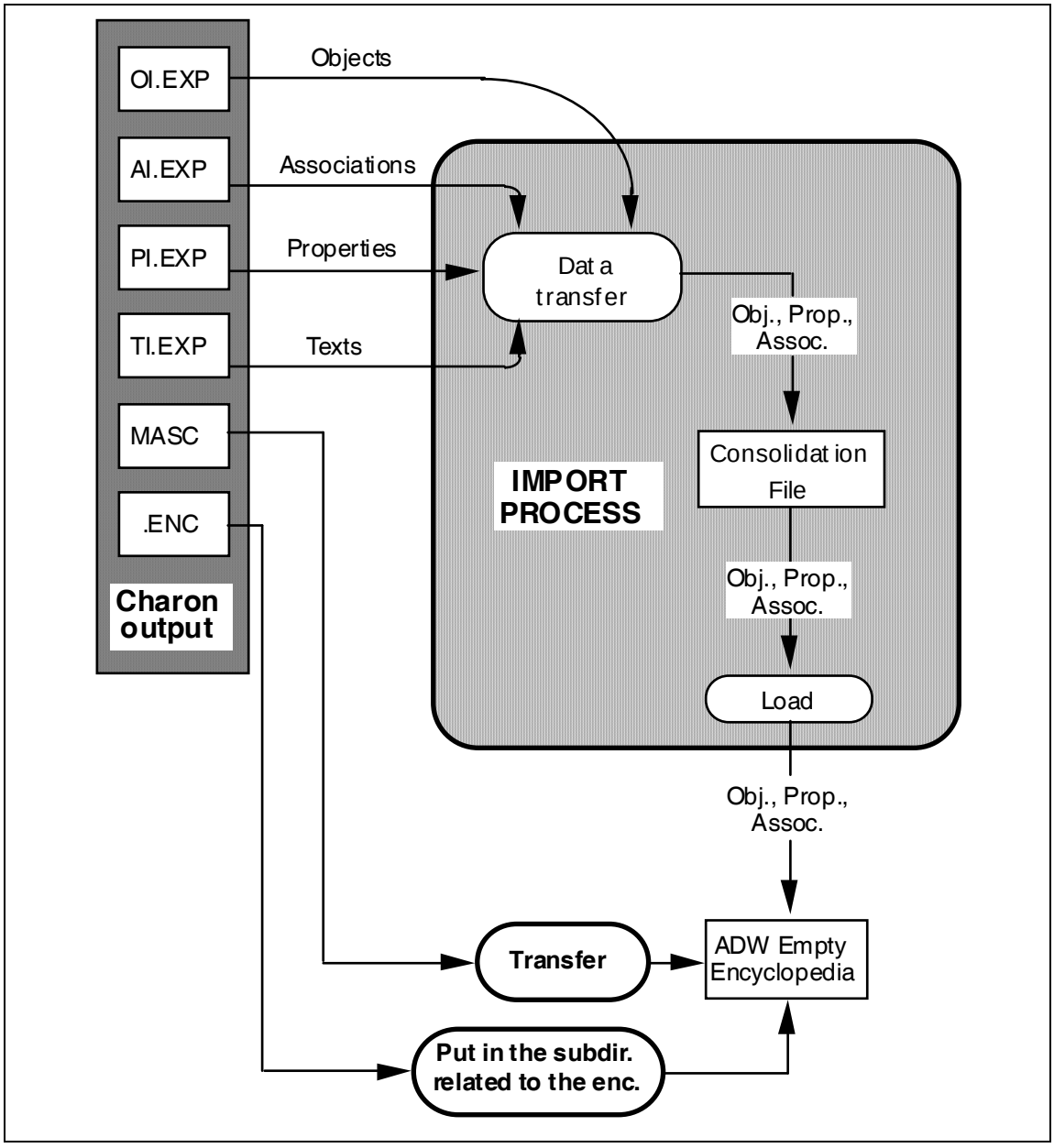


Fig. 3 - The data flow of the Adw empty encyclopedia populating.

7 - The conversion tool (*Charon*)

The conversion from the source program into the ADW tool occurs in several steps, as may be seen from Fig. 4. In more detail, the sequence of the actions is the following:

- *Step 1*

1a) Create the ".EXP" files records in order to represent the logical relational model and the data structure of the relational tables (keys, data types, formats).
1b) Create and open "MASC" files for coding the general features of the tables (physical name, SQL type, space name).

- *Step 2*
  Reconstruct the Structure Chart.

- *Step 3*
  Generate the E-R model by observing the contents of the ".EXP" files and QM tables.

- *Step 4*
  4a) Represent the modules procedural logic, including the accesses to the database and the calls to other modules.
  4b) Transform the EXEC SQL statements into masks to be inserted into "MASC" files. The statements which are not supported by ADW are inserted as comments.

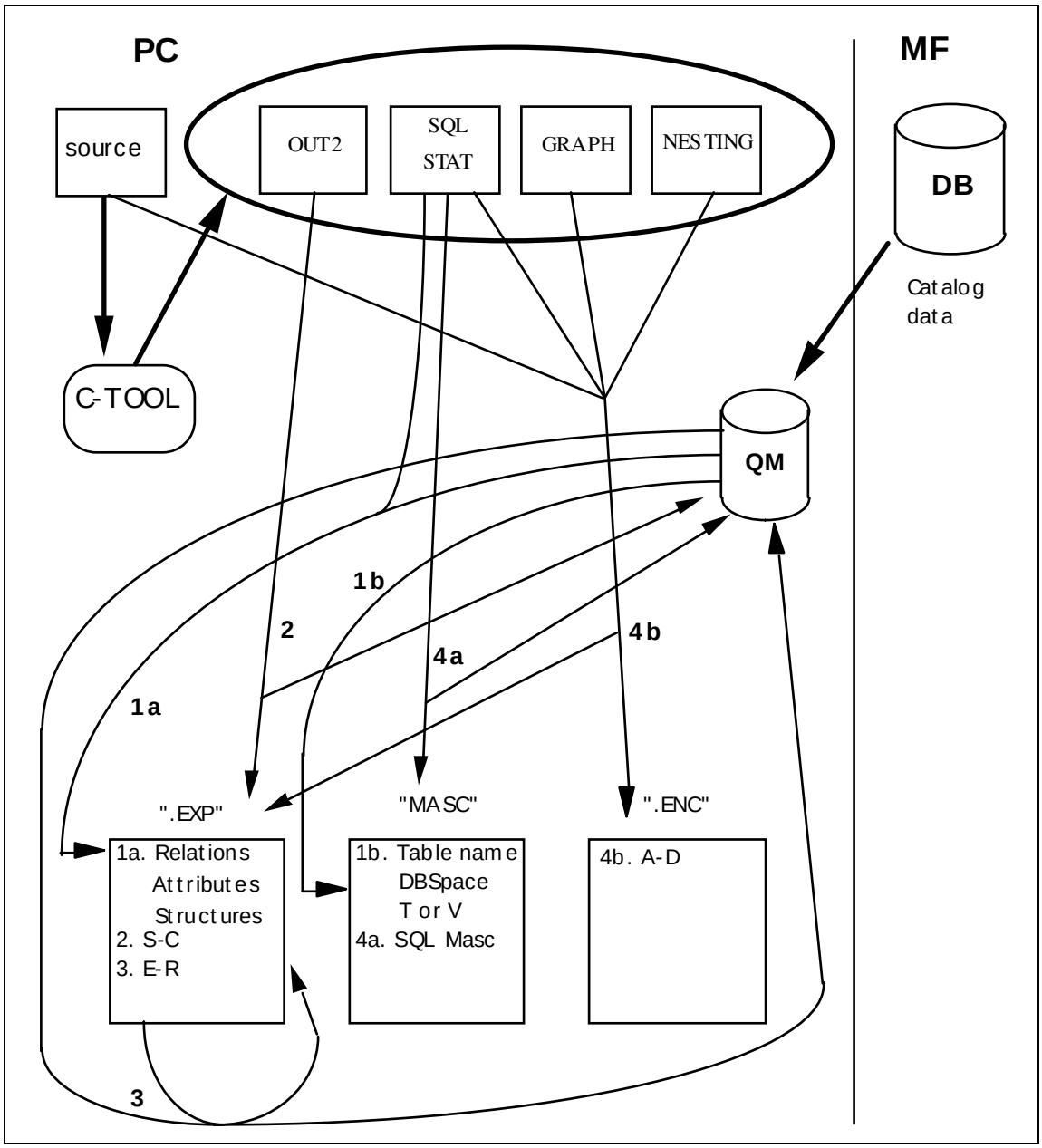


Fig. 4 - The four steps in *Charon*

Conclusions and future developments.

In many cases, software quality improvement may require a redevelopment of the application system. When a software development methodology and CASE tools have been adopted as an enterprise standard, recovering the existing software, and documenting it according to these standards may constitute a consistent improvement. It is obvious that understanding the semantics of the original programs is a key point. As a matter of fact, the re-engineering and the adoption of CASE tools, especially in large scale projects, may produce relevant advantages, namely consistency, easy maintenance and clean documentation ([9]). The consistency with the enterprise standards may in fact assure the complete integration of the various subsystems, and reduce the maintenance effort. In fact, the maintenance personnel will no more be forced to operate maintenance interventions on the source code, with the consequence

of being tied to a particular set of programs whose code they are aware of, but can operate on higher level specifications, leaving to the CASE tool the burden of the generation of the code. Needless to say, this style of work assures that the documentation will be kept up to date.
In this paper, *Charon*, a tool for the redocumentation and re-engineering of code, has been discussed. It converts the information extracted from a C program by the static code analyser C-TOOL in ASCII records and inserts them in three different typed sets of files.
The files are subsequently imported in the ADW CASE tool environment in order to populate a repository related to the project, thus the user can find the extracted information represented by most of the diagrams of the Analysis and Design level, that is:

- for the representation of general information:
  - *Object List*
  - *Object Details Window*

- for the modelling of data architecture:
  - *Database Relational Diagram*
  - *Data Structure Diagram*
  - *Data Type Window*
  - *DB2 General Information Window*
  - *Entity-Relationship Diagram*
  - *Entity Type Description*
  - *SQL Action Diagram*

- for what concern processes:
  - *Structure Chart*
  - *Module Action Diagram*

The main advantage offered by the automatic translation operated by *Charon* is that, after the reverse engineering phase, we can operate directly on the high level specifications of the software, getting all the benefits claimed by the CASE tools.
Furthermore, the user can rely on all of the ADW and supporting database manager report writers in order to produce a textual documentation. The integration of the analysed code with the enterprise wide standards can be considered as a benefit too.
With the rebuilding of E-R model, we create a system knowledge at a as high as possible level so that all the ADW capabilities will be exploited.
In this way, if we wish to make some relevant changes to the database structure, we shall easily act on the E-R schema and, then, obtain a normalised relational form by running the Relational Translator task. At that point, according with a bottom-up approach, the user will be able to generate the remaining documentation of the workbench upper levels.
If, on one side, *Charon* can be considered as a test case explaining and making real software engineering capabilities, on the other one, it can be considered as a reference for those who are going to modify software application packages, perform environment conversions, generate code from prototypes.
Secondly, *Charon* complete the C-TOOL reverse results by providing, in some tables, information about database relations, fields, relationships and module calls of the source code, and, in ASCII files, the procedural logic description of each module.

However, it should be stressed that the approach followed in *Charon* requires that the user will conform to some specific design methodology (e.g. the Yourdon Constantine Structured Analysis).
In addition, some code characteristics can't be recovered (SQL dynamic commands, modules formal parameters and input/output conversations) because their conversion from a C environment to a COBOL one is very difficult to make in a completely automatic way.
Perhaps, it would need a human intervention even if the manual approach, more flexible and friendly, could be too much heavy in the re-engineering of high complexity systems. This considerations suggest that it would be worthwhile to consider the possibility of integrating the positive aspects of the two different approaches.
In order to improve Charon, thus enhancing the set of convertible information and the collaboration with the final user, we have to face the following problems:

- testing the existence of the relationships between relations when examining the SQL commands;
- creating another ADW diagram, Screen Layout, and producing CICS code for the activation of video maps which correspond to the I/O commands in the C code;
- enhancing the Structure Chart diagram by considering the recursivity;
- representing modules formal parameters and converting their C types in COBOL ones in order to allow the generation of the Data Flow Diagram in the Analysis level (AWS);
- implementing an enriched user interface.

References

[1] *Software Re-engineering Symposium*, organised by SYSTECH Systems Technology Institute (Rome, 12-14 February 1990)

[2] Martin J.: *Recommended diagramming standards for analysts and programmers. A basis for automation,* Prentice-Hall Inc., Englewood Cliffs (1987)

[3] Martin F.M.: *Second-Generation CASE Tools: A Challenge to Vendors*, IEEE Software (March 1988)

[4] Martin J.: *CASE & I-CASE*, Informatica 70 n.167 and n.168

[5] Lewis T.G.: *CASE: Computer-Aided Software Engineering*, Informa Tex Press (1989)

[6] Chikofsky E.J., Cross II J.H.: *Reverse Engineering and Design Recovery: A Taxonomy*, IEEE Software (January 1990)

[7] De Carlini U., Cimitile A.: *Il reverse engineering nella analisi, documentazione, manutenzione e validazione del software*, Sistemi Informatici e Calcolo Parallelo: progetto finalizzato CNR: risultati, stato delle ricerche e prospettive, Angeli (1991)

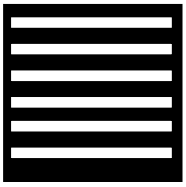
[8] Lanubile F., Maresca P., Visaggio G.: *An Environment for the reengineering of Pascal Programs*, Proceedings of IEEE Conf. on Software Maintenance, Sorrento, August 1991, pp.23-30

[9] Signore O., Celiano F.: *From a "well designed" database to AD/Cycle tools: a reengineering experience*, Proceedings of CASE and Applications Development

in Practice, SHARE Europe (SEAS) Spring Meeting 1991, Lausanne, Switzerland, April 8-12, 1991, pp. 1-8, ISSN 0255-6464

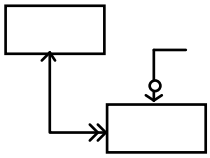
Appendix 1.

A description of the diagrams the user can manipulate after the import process is presented in the following.

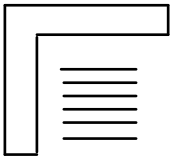


The *Object List* lists all the encyclopedia objects which meet some requirements (name, type, related level in the life cycle model). It combines the report and diagram fashions.

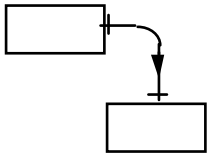
The *Object Details Window* displays general information about an object (name, type, creation date, last update date, definition, comment).



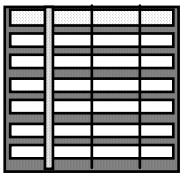
The *Entity-Relationship Diagram* is the well-known Chen diagram including the entities (fundamental, associative or attributive) and the relationships between them.



The *Entity Type Description* describes for each entity the attributes, with their types and cardinality constraints, the relationships the entity is involved in and the keys.

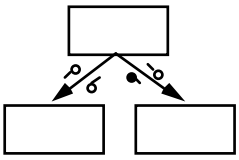


The *Database Relational Diagram* models the database according to the relational logic model. References between relations display cardinalities and show which relation have foreign keys that refer to the primary keys in other relations.



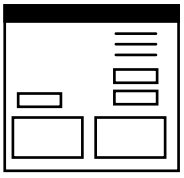
The *Data Structure Diagram* highlights the properties of a stored data. A data structure can be composed of any combination of data elements or data groups. A data element is an atomic unit containing no other structure; on the contrary, data groups can include data elements and/or other data groups. A data structure can describe the structure of a relation, a file record, a segment and screen layout variable. In the case we deal with, data structure describes relations properties like primary and foreign keys, SQL data type, mandatory clause, indexes, etc.

The *Data Type Window* defines the peculiar characteristics of a global or local data type (SQL data type, COBOL format, external and internal length, definition, comment).

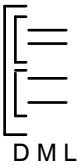


The *Structure Chart* offers a graphic depiction of the external, modular structure of a program. It details the hierarchy and organisation of logic units (modules), the distribution of functionalities among modules, and the data communication between them.

The *Module Action Diagram* details, in a graphic format, the logic and the structure for a program module. It also establishes the encyclopedia relationships between modules and other design object types, such as screen layouts, relations, segments, or other modules.



The *DB2 General Information Window* presents the environmental parameters which are bind to a DB2 physical representation of a relation such as DB2 database name, DB2 table space, table name, DDL subject type (table or view).



The *SQL Action Diagram* enables the generation of a generic template for a SQL DML command. ADW/Construction generates five different sets of generic DML statements, based on the type you select: CURSOR, DELETE, INSERT, SELECT and UPDATE. A sixth choice, EXPERT, allows to write other DML statements.