ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

B Novikov, University of St Petersburg, Russia and J W Schmidt, University of Hamburg, Germany. (Eds)

Advances in Databases and Information Systems, Moscow 1996

Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS '96). Moscow, 10-13 September 1996

Migrating Data-Oriented Applications to a Relational Database Management System

K. Menhoudj and M. Ou-Halima

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Migrating Data-Oriented Applications to a Relational Database Management System

K. MENHOUDJ

LISI-Bat. 401, INSA, 20, Avenue A. Einstein
69621 Villeurbanne cedex France

M. OU-HALIMA

LISI-Bat. 502, INSA, 20, Avenue A. Einstein
69621 Villeurbanne cedex France

Abstract

The need to update and reengineer legacy information systems to improve business processes and upgrade technology is becoming increasingly desirable for many organizations in the current economic climate. This involves migrating systems from centralized to distributed client-server architectures. These information systems are often written in COBOL, their data is stored on files and they use text-based user interfaces.

In this paper, a method to migrate such an information system is presented. The target environment in our case is a relational database management system. In our method, the migration is carried out in an incremental way and by small steps. The migration process is driven by the data files. These files are previously arranged and then migrated in a defined order. An algorithm that defines the order of migration of the data files is described and an example is presented to clearly illustrate our method.

Keywords: Migration, reengineering, legacy information system.

1 Introduction

Reengineering is being a significant activity in many software organizations. It has come about mainly by the need to:

- Take maximum advantage of new technology
- Improve business processes and meet new requirements
- Improve maintainability and reduce maintenance cost

Recently several studies addressed different aspects of this problem as:

- Reverse engineering existing applications [7, 8, 17] including redocumentation and design recovery
- Migrating large systems to rightsize them [5, 15]
- Migrating applications to more powerful platforms [1, 13]
- Migrating text-based user interfaces to graphical user interfaces [12]

However, papers that dealt with migrating legacy systems as [5, 13] did not address the order in which legacy system components (i.e. data and programs) should be migrated to the target system. So, in this paper we propose a data file driven migration method that takes into account this problem. Indeed, we present an algorithm that defines an order of migration for the data files of the legacy system.

This paper deals with migrating data-oriented applications written in COBOL to a relational database management system and a graphical user interface. The COBOL data files will be mapped into relational tables but the modules will be entirely rewritten since the target environment provides a graphical user interface.

This paper is organized as follows. Section 2 presents the framework of the study. Section 3 describes the migration method and details its steps. Section 4 addresses the problem of arranging data files to migrate them and finally the conclusion in section 5.

2 Framework

Generally, we identify two main strategies to migrate an existing system:

1. The revolutionary strategy that involves replacing and rebuilding a legacy system from scratch using new technology of the target environment.
2. The evolutionary strategy that involves achieving an incremental transition to the target environment by small steps.

That is, the second strategy is much safer and more feasible [5] than the first one especially when the existing system is too big and complex. That is why we adopt it in our method. Therefore, the revolutionary strategy may be of use in the case of small applications. Thus, we intended to build a database gateway between the legacy system and the target system so that they could communicate and exchange data (Figure 1), i.e., legacy modules accede to the target database and the target modules accede to the legacy COBOL data files too.

Figure 1: The database gateway

But in this manner, the development of the gateway is labor-intensive and time consuming. So, we have chosen to reduce the gateway functionality by defining one-way database gateway (Figure 2) in the following direction: **legacy system** → **target system**. This means that only legacy modules can get data from the target database. The target modules are not authorized to accede to legacy COBOL data files.

3 The migration method

Based on the evolutionary strategy, our method achieves an incremental migration. Moreover, the migration process is driven by the data files of the subject system. This means that the migration of the modules depends on the migration of the data files.

Figure 2: The one-way database gateway

3.1 Principle of the method

The migration is achieved according to this principle: *when a given data file is migrated to the target database, it defines a set of modules that will be migrated (rewritten) before transferring the next candidate data file. These modules are typically those that only accede to the target database since all of the legacy data files they used to accede have already been migrated.*

The example below illustrates this principle (Figure 3). It presents three intermediate states of a migration process where a data file (T1) and the module (M1) that uses it have already been migrated to the target system (Step (a)). Assume now that the candidate file to migrate is F2. It will be replaced by the table T2 as shown in Step (b). Let us see now which module satisfies the condition of migration. There is one module (M2) that accede only to the target tables, so it is the next component to migrate (Step (c)). Then we go on with the remaining data files as in this example.

Figure 3: Intermediate states of a migration process

The previous principle of migration is implemented by the following algorithm:

```
ALGORITHM: Data file Driven Migration (DDM)
BEGIN
  WHILE The list of legacy data files is not empty
    DO Migrate the first file of this list
      WHILE There is a legacy module that does
        not accede to any legacy data file
        DO Migrate this module ENDDO
      ENDDO
    ENDDO
  END.
```

N.B: The list of legacy data files is previously ordered and when a legacy data file or module is migrated it is removed respectively from the set of legacy data files or modules.

Now, if we consider the set of the data files to migrate, a question must be asked: in which order should they migrate? But before this, let us see the importance of arranging these files.

3.2 Why arranging the data files

As we saw, there is a need to arrange the legacy data files and migrate them in a defined order. Indeed, we have noticed during our experiments that according to the order in which the data files are migrated, the migration is simplified or complicated. Hence, we have developed an algorithm that arranges the legacy data files before migrating them. In fact, the order of migration of the data files is defined so that:

- the legacy code is altered as little as possible to minimize risk
- the database gateway that ensures data transfer between the legacy system and the target system is simplified.

Let us see an example to clearly illustrate this problem. Here are two legacy data files that will be migrated to a relational database:

- Customer (Numcus, Namecus, Fstnamecus, Towncus)
- Order (Numord, Numcus, Namecus, dateord)

and a legacy module M that uses the file Order and especially its property *Namecus*.

The normalized target relational tables will be:

- Customer (Numcus, Namecus, Fstnamecus, Towncus)
- Order (Numord, Numcus, dateord)

Notice that the property *Namecus* is removed from the target table Order. Now, assume we migrate the file Order before migrating the file Customer. Two situations can appear:

- the case where the legacy module M has not been migrated yet (Figure 4)
- the case where the legacy module M has been rewritten in the target system (Figure 5)

So, in the first case, the legacy module M cannot get the information *Namecus* from the target database. Hence, we are compelled to alter the code of this module so that it can get this information from the legacy file Customer. But, as we have said above, we try to avoid altering legacy code as far as possible in order to minimize risk. Indeed, in the case of several files, the alteration of legacy code can lead to serious problems and make the migration difficult.

Figure 4: First case

Figure 5: Second case

In the second case, we have to build a database gateway in the direction: **target IS** $\longrightarrow$ **legacy IS** to get the information *Namecus* from the legacy file Customer. This operation complicates the development of the database gateway and then the migration.

That is why the order of migration of the files is very important and directly affects the complexity of the migration process [16]. To determine this order, we use an algorithm that has as inputs both the legacy data files and the target relational tables.

3.3 The migration steps

Step 1: *Preliminary step.*

In this step, we list all of the modules of the legacy system and the data files they use. Then, we perform a mapping of the legacy data files into relational tables. This task is important for the next steps.

Step 2: *Arrange the data files.*

The order of migration of the files is very important and affects directly the complexity of the migration process. To determine this order, we use an algorithm that has as inputs both the legacy data files and the target relational tables. It performs three operations:

- Construction of the matrices of the legacy files and the corresponding target tables.
- Detection of the interdependence of the legacy files.
- Construction and then interpretation of the matrix of precedence.

Section 4 will present in detail this algorithm.

Step 3: *Plan the migration.*

Once the data files are arranged, we use the algorithm DDM (c.f. paragraph 3.1) to get a global view of the migration process as in Figure 3 and to estimate the complexity of the database gateway to develop.

Step 4: *Prepare the migration.*

Migrating legacy data files to the target database involves downloading, converting and uploading large amount of data. A program that can carry out this operation has to be developed. Moreover, the database gateway should be designed to ensure the mediation between the legacy modules and the target database. Concerning the modules, before migrating a legacy module, we have to design the target module and its interface in the target graphical environment.

Step 5: *Carry out the migration.*

At this step, we implement what was specified in the previous step, i.e.:

- implement the target relational schema and migrate the data
- build the database gateway
- rewrite the target modules and interfaces

Step 6: *Run the changes.*

At this step, we run the changes i.e. the migrated components and we go back to the step 4 to prepare the migration of the next component.

Step 7: *Switch from the legacy IS to the target IS.*

When all of the legacy modules and files are migrated, we switch from the legacy IS to the target IS.

4 Algorithm of precedence

This algorithm arranges the legacy data files and defines an order of precedence for them.

4.1 Description

- **Construction of the matrices F (for files) and T (for tables)**

They represent the legacy files and the target tables with their corresponding properties. When a property belongs to a given file, we put 1 in the corresponding case, otherwise we put 0 (Table 1).

	P1	P2	...	Pn
F1	1	0	...	1
F2	1	1	...	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Fm	0	1	...	0

Table 1: Input matrix of files/tables

- **Detection of the interdependence of the legacy files**

Typically, this operation locates the files that lose some of their properties by normalization. In order to do this, we have built up the following formula: $\mathbf{R} = 2 * \mathbf{F} - \mathbf{T}$. The resulting matrix R can be interpreted according to the following table. It shows the possible cases that can appear in the target tables:

F	T	2*F-T	Interpretation
0	0	0	No properties
0	1	-1	Creation of a property
1	0	2	Suppression of a property
1	1	1	Unchanged property

Table 2: Interpretation of the resulting matrix

Given two legacy files F1, F2 with a common property P and their corresponding target tables T1,T2, we say that F1 is dependent on F2, i.e. F1 should migrate after F2, when P is kept in T2 and removed from T1.

Based on the interpretation given above, we can say that in Table 3, Fm is dependent on F2 because P1 will be kept in T2 and removed from Tm.

	P1	P2	...	Pn
F1	0	0	...	2
F2	1	-1	...	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Fm	2	1	...	0

Table 3: An example of a resulting matrix

• Construction and interpretation of the precedence matrix

The precedence matrix is presented in Table 4. It is obtained from the resulting matrix and gives the interdependence of the legacy files in a more readable way.

	F1	F2	...	Fm
F1	0	0	...	1
F2	1	0	...	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Fm	0	0	...	0

Table 4: The precedence matrix

It can be interpreted as follows:

- A file F_i of a given line precedes a file F_j of a given column when the value of their corresponding case is 1.
- When a given line is null, then the corresponding file does not precede any files.
- When a given column is null, then the corresponding file is not preceded by any files.
- When the line and the column of a given file are null, then it does not precede any files and it is not preceded by any files either.

4.2 An example

The following example will illustrate the described algorithm.

• Mapping the files into tables

Files

```
Customer (Numcus, Namecus, Fstnamecus, Strcus,
          Poscodcus, Towncus)
Supplier (Numsup, Namesup, Fstnamesup, Strsup,
          Poscodsup, Townsup)
Product (Numpro, Descpro, Pricepro)
```

Migrating Data-Oriented Applications to a Relational Database Management System

Order (Numord, Numcus, Namecus, Dateord, Numlin,
 Numpro, Descpro, Qty, Amolin)
 Invoice (Numinv, Numord, Dateord)

Tables

Customer (Numcus, Namecus, Fstnamecus, Strcus,
 Poscodcus)
 Supplier (Numsup, Namesup, Fstnamesup, Strsup,
 Poscodsup)
 Town (Poscod, Town)
 Product (Numpro, Descpro, Pricepro, Rem)
 Order (Numord, Numcus, Dateord)
 Invoice (Numinv, Numord)
 Line (Numlin, Numord, Numpro, Qty)

Notice: The properties Poscod and Town of the table Town correspond respectively to the numbers 5, 10 and 6, 11.

To simplify the task, we will number the different properties (Table 5).

N0	Property	N0	Property	N0	Property
1	Numcus	8	Namesup	15	Numord
2	Namecus	9	Strsup	16	Dateord
3	Fstnamecus	10	Poscodsup	17	Numlin
4	Strcus	11	Townsup	18	Qty
5	Poscodcus	12	Numpro	19	Amolin
6	Towncus	13	Descpro	20	Numinv
7	Numsup	14	Pricepro	21	Rem

Table 5: Identification of the properties

• Construction of the matrix F of the files

Files	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Customer	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Supplier	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Product	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
Order	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	0	0
Invoice	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0

Table 6: Matrix of the files

• Construction of the matrix T of the tables

When several tables correspond to one file, we put them in the same line.

• Construction of the resulting matrix $R=2 \times F \cdot T$

Tables	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Customer+Town	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Supplier+Town	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Product	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1
Order+Line	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0
Invoice	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0

Table 7: Matrix of the tables

Tables	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Customer+Town	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Supplier+Town	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Product	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	-1
Order+Line	1	2	0	0	0	0	0	0	0	0	0	1	2	0	1	1	1	1	2	0	0
Invoice	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	1	0

Table 8: Resulting matrix

	Cust.	Sup.	Prod.	Ord.	Inv.
Customer	0	0	0	1	0
Supplier	0	0	0	0	0
Product	0	0	0	1	0
Order	0	0	0	0	1
Invoice	0	0	0	0	0

Table 9: Precedence matrix

- Construction of the precedence matrix
- Order of migration of the files

Once the resulting matrix is interpreted, we can represent the order of migration of the files by an oriented graph as in Figure 6.

Figure 6: Order of migration of the files

The files that are not linked in the graph are independent. So, they can be migrated at any time.

5 Conclusion

This paper dealt with a topical problem in industry. It involves migrating a legacy IS. Thus, we have proposed an incremental method of migration of a centralized IS. In our method, we tried to capitalize the legacy IS components (e.g. data and design) and reuse them in the target IS so that we save money and time. The process of migration is driven by data files that are arranged beforehand. The remaining components of the IS are migrated according to the defined order of the files. Finally, it should be stressed that our method eases the migration by defining a one-way gateway and reducing its size.

A Appendix

ALGORITHM OF PRECEDENCE

```

BEGIN
  READ (F[m,n],T[m,n])
  CALCULATE (F[m,n],T[m,n],R[m,n])
  INTERPRET (R[m,n],P[m,n])
END.

CALCULATE (X[m,n],Y[m,n],Z[m,n])

BEGIN
  FOR i=1,m
    DO FOR j=1,n

```

```
        DO Z[i,j]<-- 2*X[i,j]-Y[i,j] ENDDO
    ENDDO
END.

INTERPRET (X[m,n],Y[m,n])

BEGIN
    FOR i=1,m
        DO FOR j=1,n
            DO IF X[i,j]=2
                THEN FOR k=1,m
                    DO IF X[k,j]=1
                        THEN Y[k,i]<-- 1
                    ENDF
                ENDDO
            ENDF
        ENDDO
    ENDDO
END.
```

Notations:

F[m,n]: Matrix of files,
T[m,n]: Matrix of tables,
R[m,n]: Resulting matrix,
P[m,m]: Precedence matrix.

References

- [1] G. Antoniol, R. Fiutem, E. Merlo, P. Tonella. Application and user interface migration from BASIC to Visual C++. In *Proceedings of the International Conference on Software Maintenance*, Nice, France, October 16-20, 1995.
- [2] R. Arnold. *Software reengineering*. IEEE Computer Society Press, 1993.
- [3] M. Ayache, M. Ou-Halima, K. Menhoudj. La réingénierie : objectifs et techniques. In *Software Engineering and Applications: 2nd ISEI*, Bucharest, May 95.
- [4] T.J. Biggerstaff. Design and recovery for maintenance and reuse. *IEEE Computer*, 22(7), July 1989.
- [5] M.L. Brodie, M. Stonebraker. *DARWIN: On the incremental migration of legacy information systems*. DOM Technical report, TM-0588-10-92-165, GTE Laboratories Incorporated, November 1992.
- [6] M.L. Brodie. The promise of distributed computing and the challenge of legacy information systems. In *Hsiao, D., E. J. Neuhold, and R. Sacks-Davis (eds.), Proc. IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems, Lorne, Australia*, November 1992, Elsevier North Holland, Amsterdam 1993.
- [7] E.J. Chikofsky, J.H. Cross II. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13-17, January 1990.
- [8] C. Desclaux, M. Ribault. *Reverse Engineering Methodology RE-ORDER*. Projet Esprit II-2570 Version 1.0., April 1992.

- [9] S.B. Ornburn, S. Rugaber. Reverse engineering: Resolving conflicts between expected and actual software designs. *In Proceedings of the Conference on Software Maintenance*, pp. 32-40, Orlando, Florida, November 1992.
- [10] J. Martin. *Information engineering, Book III: Design and construction*. Prentice Hall Englewood Cliffs, New Jersey 07632: prentice hall, 1990.
- [11] K. Menhoudj, M. Ou-Halima. *Data file driven migration*. Technical report, LISI-INSA de Lyon, Juin 1995.
- [12] M.M. Moore, S. Rugaber. Issues in user interface migration. *In Proceedings of the Third Software Engineering Forum*, Orlando, Florida, November 1993.
- [13] S. Rugaber, S. Doddapaneni. The transition of application programs from Cobol to a fourth generation language. *In Conference on Software Maintenance*, Montreal, Canada, September 27-30, 1993.
- [14] A.R. Simon, T. Davis. *Applications migration, IBM to VAX*. Van Nostrand Reinhold, NY, 1992.
- [15] H.M. Sneed, E. Nyary. Downsizing large application programs. *In Proceedings of the Conference on Software Maintenance 1993*, pp. 110-119, IEEE Computer Society Press, September 1993.
- [16] L. Sylvestre. *Projet REMEDES: La migration vers les systèmes client/serveur*, Rapport de DEA, LISI, INSA de Lyon, Septembre 95.
- [17] H.J. Zuylen. *The REDO Compodium, Reverse engineering for software maintenance*, A Special Esprit Report, London, John Wiley & Sons, 1993.