Automatic Evolution of Legacy Data Objects

Maseud Rahgozar and Farhad Oroumchian

Control and Intelligent Processing Center of Excellence
Department of Electrical and Computer Engineering,
University of Tehran, Tehran , Iran
(rahgozar@ut.ac.ir, foroumchian@acm.org)

Abstract: - The modernization of the Legacy Information Systems (LIS) is a critical issue for many organizations world wide. The successful migration of the legacy data stored in the old data formats is a challenging issue with respect to backward compatibility and future extendibility. By migration, we mean translating legacy data and related programs to native data and programs running on modern platforms such as Unix (or NT). Currently, in most migration projects, converting all legacy data objects (traditionally stored in flat and indexed files) to RDBMS tables is considered to be unrealistic and out of question because of the expected performance problems. While in our experience, those legacy data objects have to be converted to the modern database (RDBMS) tables and fully integrated in the Global Information System schema. Otherwise those data objects will not be ready for the application of novel technology tools and soon will become a new bottle neck in the system. This is, not only to create a unified schema of the new Information System, but also to take full advantage of transaction management and recovery services provided by the database management systems. In order to guarantee performance, efficiency, share ability and ease of future enhancements, the legacy data has to be fully normalized too. This paper examines the issues concerning the migration of legacy data objects to RDMS environment and offers a practical approach.

Key-Words: - Legacy Data Objects, Legacy Information Systems, Evolution, Normalization, Migration.

1 Introduction

Legacy Information Systems (LIS) are aging application systems developed during the last three decades. They constitute a large number of existing systems [7]. These applications need to be evolved to new technology environments. There are many approaches to modernization of Legacy environments [17],[1],[5],[8]. A classification of different approaches is presented in [13].

There have been limited research works on reverse engineering of legacy data files [3],[4],[12],[6],[16], [10],[9]. This is in contradiction with market demands [11], but, it can be explained by the amounts of difficulties expected in such environments. Even so, these works do not deal with migration of the legacy programs together with their corresponding data files. We have not found any work related to the normalization of legacy data in the context of migrating legacy programs.

According to our longtime experiences, for most migration projects, project managers avoid converting the legacy data files to database environments (RDBMS). This conservative approach is explained by the lake of effective tools and solutions for the expected performance problems. That is why thinking about data normalization would be still too far fetched.

From the current literature, one may conclude that moving legacy applications from Indexed-Files environments to modern systems (Unix) and modern database environments (RDBMSs) is still *fairly complex and risky activity, in such a way that it is simply not sought of in most of LIS renovation approaches*. In such a context, it may seem very unrealistic to think about running those applications on RDBMS environments with fully normalized data.

This paper focuses on the issues regarding the migration and normalization of the Legacy Data in conjunction with the corresponding legacy programs. A solution is provided here that deals with the migration of "data + code" as a whole. Here, we discuss a data access mapping interface that separates the code from its underlying data representation, therefore enabling us to place the legacy data objects in the normalized relational tables. This solution does not affect the program's code, logic or performance.

The data normalization theories or the methods applied in data reverse engineering approaches are not discussed here. But, the resulting issues of data normalization process, i.e.

the potential changes that may happen to the data structures, and transparent integration of such changes are discussed.

Section 2 discusses migration issues of legacy data to Unix RDBMS environments, and section 3 presents our data access interface solution. This approach is the result of many years of managing R&D projects related to the renovation and evolution of the LIS systems [13][14][15]. Some of the solutions implemented by this approach are being used on hundreds of sites in Europe. Section 4 offers a comparison of the performance of this solution to the others. Section 5 presents one of our successful experiences as a case study.

2 Migrating Legacy Data

Historically, legacy data objects have played a vital role in keeping permanent data in legacy systems. These data stored in indexed files are at least as important as those stored in database tables. They need particular attention because they need more normalization and they have to be reformatted and moved to relational data bases. They have to be set together with the data in other databases in a unified Information System schema. There are five particularities in this context:

- The legacy data files are going to be evolved to a RDBMS environment,
- Their definitions are to be evolved to a normalized model,
- They are to be integrated again into the legacy programs environment,
- The legacy programs codes (i.e. their structures and data access logics) are to be fully respected,
- The same data is going to be shared with future applications using new technology tools.

The design of legacy data needs significant changes in order to be converted to RDBMS environment and fully normalized. Legacy data normalization is a prerequisite to data sharing and future extention of the unified Information System. This includes many topics to deal with, such as, splitting, atomizing or adding new data items, or splitting, joining and merging of the tables. For most legacy information systems, the conceptual and logical designs of the legacy data files are rather poor [2]. Some approaches to normalization of legacy data files have already been worked out. These approaches are mostly developed in the context of data reverse engineering, and precede the redevelopment of the programs from scratch [11]. The data reverse engineering only takes advantage of the conceptual or logical definition of the old information system and defines a data mapping method from old information system to the new one. That context is different from the migration context where the resulting data structures are re-integrated into the same programs environment. This integration has to be transparent to the related programs. They should be able to access the data as before (i.e., with the same logic, the same structure, and the same performance or better!).

In the rest of the paper the results and consequences of data normalization are discussed and then a solution for re-integrating the new data structures into the migrated programs is presented.

2.1 Data Normalization

Many kinds of changes in data structures can be expected following the normalization of the data design [2],[11]. The changes may concern the logical or the physical structure of data items as well as data tables. Here are some resulting changes that are expected following the normalization of the legacy data design:

- **Changing items format:** the format of items may be changed to support future extensions or better representation of data such as dates, times, currencies, etc.
- **Adding or Suppressing items:** there may be some old items that have no use in the existing programs. They do not need to be created in the new tables. There may also be some new items that will be needed for future programs.
- **Splitting, Atomizing or decoding items:** some compound legacy items and some encoded items may be split into multiple atomic elements for better representation and extended usages in the future programs.
- **Verticalizing arrays and matrix of items:** the single or compound items with multiple values are mostly stored in single records with a maximum places reserved. Such items will be stored in multiple rows for a better representation in the new tables.
- **Merging, Mixing and Redefining items:** to suppress redundancies or to obtain a better representation of data, some items may be merged, mixed or redefined differently in the new tables.
- **Splitting or Merging tables:** legacy data records may also contain redundancies or inconsistent representation of data. They have to be merged, mixed or redefined differently in the new tables.

The migrated programs should interact with the “Legacy View” of the normalized data. That is, they should view the data records as if there have not been any changes. The mapping between the Legacy View and the Normalized

View of the data will be provided by "Data Access Interface". This interface should resolve changes such as suppressed, split, encoded, reformatted and added items or split and added tables. That is to avoid any unpredictable behavior by migrated programs due to unexpected changes in the data layout or any changes in the data access logic.

2.2 Tasks to accomplish

We define four tasks that have to be accomplished for migration of the data design:

a) **Data Objects Unification:** One of the frequent problems encountered with the legacy data files is the lake of a unique and global definition of items. This is due to the flexibility of languages such as COBOL and the absence of controls and services that are normally provided by Database Management Systems (and not by File Management Systems). In most legacy systems, the sources of many programs have to be searched and their file definitions have to be extracted in order to gather a unique and global definition of data items for each data file.

b) **Design Normalization:** Another frequent issue encountered with the legacy data files is the lake of a conceptual and logical definition of the Information System. This may be due to the initial design or the numerous technical extensions, modifications or optimizations being applied to the Information System over the years. So, the normalization task has to be done using one of the approaches referenced in this section. The choice of the normalization approach is not the subject of our study in this paper.

c) **Creating access mapping interfaces:** The first consequence of the preceding tasks is that with the new data design, the migrated programs will not be able to find the data records as they used to see on the legacy environment. On the other hand, changing the thousands of programs' codes to adapt to the new data design is too risky and may result in thousands of bugs and performance problems. Thus, we need to map between the legacy view (i.e. the old data structures) and the new data view. The most effective solution will be to create for each legacy data file the interfacing programs (Access Mapping Functions) that link between the old and the new data structures. Fig. 1 depicts this situation where each legacy read/write function could map to user defined mapping interface that could perform multiple read/write operations in the new data environment. This solution is easy to implement and brings no risk of bugs or performance problems to the existing codes. These interfacing programs can mostly be generated automatically.

d) **Creating conversion interfaces (export/import):** We also need to unload the legacy data and to reload it on the new environment. One solution is to simply dump or unload each data file into a flat file using portable formats and then port it to the new system. That is to create a program that reads the data file sequentially, reformats the data and then writes it into the dump file. Another program will do the reverse action by reloading the data in the new system but this time the data will automatically be normalized. The first program has to be run on the legacy environment. The second program has to be created using the Access Mapping Functions (created in the preceding task) and run on the new environment. The automatic generation of such export/import programs is straight forward.

In the rest of this paper, the focus will be on the implementation of the Access Mapping Functions or Interfaces.

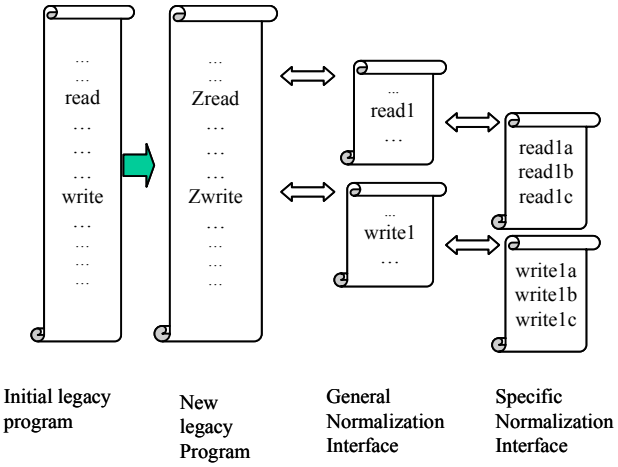

Fig. 1. Dynamic Normalization Procedures

3 Data Access Interface

The implementation or structure of the legacy data is mostly navigational or hierarchical and the logic of the legacy programs has been built around this structure. The simplistic approach of replacing isolated legacy data access statements by equivalent SQL statements will lead to significant and prohibitive performance degradation. An effective transformation of the legacy data access logic to the relational data access logic is not linear. Therefore the legacy data access logic should be considered as a whole and managed through a specialized data access interface. The main objective of creating the "Data Access Mapping

Interfaces" is to avoid any alteration in the legacy data access logic in the programs.

3.1 Legacy Access Logic

The most common way legacy programs access data files is to set a pointer on a desired position in the legacy file with a specified key value (START statement) and then read the file, record by record from that position on (READ NEXT or READ PREVIOUS statements). The records returned by the READ NEXT statement will be sorted based on the key item specified in START statement. This logic is adapted to the physical implementation of the data in the indexed files, so the performance of legacy programs is very good on such file structures. However, when the same data is moved to a relational table, supporting such access logic with acceptable performance (i.e. the same response time) is not straight forward. In other words, if we simply use SQL statements to replace the START and READ NEXT statements, the response time will be hundreds or thousands of times greater than those of the indexed files depending on the number of records in the table. This is because of the difference between SQL access logic and that of the indexed files. In SQL, the SELECT statement with the expression (ITEM >= "value" ORDER BY ITEM) will result in loading all the records satisfying this criteria and then sorting them in the memory work space. With this access logic, the response time of the SQL statements replacing START and READ NEXT statements will be measured by minutes and hours instead of milliseconds as in the indexed file.

The only remedy to above problem is to use the expression (ITEM >= "value1" AND ITEM <= "value2" ORDER BY ITEM) where the "value2" should be chosen so that the number of resulting records stay limited (e.g., between 50 and 100). But finding the "value2" is a challenging issue by itself. To implement such access method, we have to use a caching mechanism for keeping a suitable range of keys in the memory. Then, we can find the best candidate key value ("value2") following "value1" in the cache-key buffers. Managing the cache-key buffers, in the cache memory is also a challenging issue, when the number of records in the table is measured by hundreds of thousands or millions. To solve this problem, we have developed a swapping mechanism that is explained below.

3.2 The key Caching Mechanism

The cache-key buffers have to be maintained dynamically and to be refreshed regularly. They may also need too much memory space and they have to be shared between concurrent programs. Hopefully, they do not need to be updated for every single key value created or deleted within the table. The most effective solution to support these requirements is to implement a *dynamic swapping mechanism* that uses an indexed dump file containing a raw picture of key distribution within the table, e.g. one sample key value for every 100 distinct key values. The key dump file should be created and refreshed dynamically. The expected frequency for refreshing the key dump file is determined automatically by the rate of changes in the table, which can be measured by the rate of valid key values fetched from the file. The key values fetched from the file are said to be valid, if for two neighbor values "value1" and "value2" in the file, the number of key values returned by the SELECT statement with the expression (ITEM <= "value1" AND ITEM > "value2") stay within a reasonable range (e.g., between %50 to %200 of the expecting number of keys)

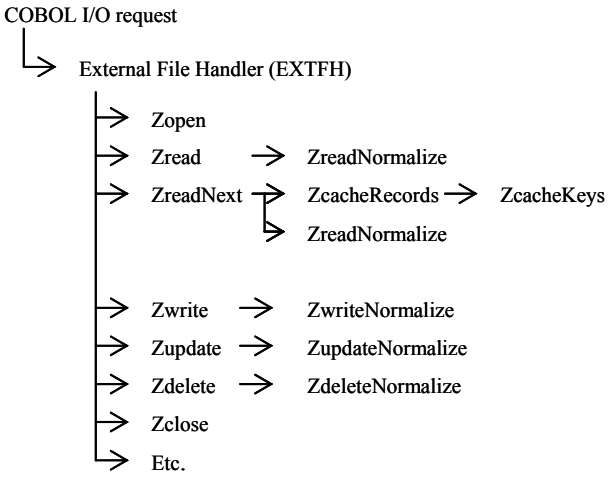

Fig. 2. The structure of calls following the COBOL I/O requests

3.3 Data Access Functions

We have to provide multiple data access mapping interfaces and different sets of system functions to access data files depending on the original legacy environment of the programs. Some examples of legacy data file environments are: KSAM (for HP3000 of HP), UFAS (for DPS6, DPS7 and DPS8 of BULL), ISAM and VSAM (for IBM mainframes), CISAM, etc. Although there are some differences among these access interfaces, they provide very similar functionalities. These functionalities are more standardized in the COBOL language file access environment across platforms. Fig. 2 shows the structure of I/O normalization functions that are implicitly called for COBOL programs.

4 Program Performance

We have implemented the above mentioned techniques and performed several performance benchmarks and comparative studies between legacy applications running on their original platforms, and their migrated versions on UNIX platforms with different configurations and using different RDBMS environments. The results have always confirmed our expectations. Fig. 3, shows the average access time we have observed for the most common data access methods, i.e., READ NEXT statements on different configurations. The horizontal axe represents the total number of records, ranging from 1000 to 1000000, in the legacy files or their corresponding database tables. The vertical axe shows the average access time, ranging from 10 milliseconds to 100000 seconds, per record. The line (a) shows the average access time for the indexed file on the legacy environment. The line (b) shows the average access time for the same indexed file on the UNIX platform. The line (c) shows the average access time observed when the data is moved into a database table and the READ NEXT statements are replaced with SQL statements (the simplistic solution). As expected, in this case the response times are measured in minutes and hours instead of milliseconds. The line (d) shows the equivalent situation but with the proposed approach, i.e., with our data access mapping interface, swapping mechanism and transaction management. This time, the access times are better than those on the original platform while the data is moved to the normalized database tables. As line (b) depicts, UNIX indexed files are faster than the data base approach but with a major draw back. That is, in the UNIX indexed file approach, we do not have the transaction management, data recovery management and many other services that are the integral part of the RDBMS environment.

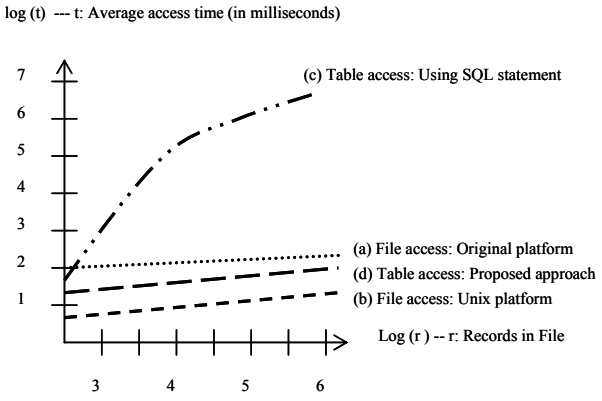

Fig. 3. Average access time per Record access (in READ NEXT)

5 Case Study

One of the successful experiences we accomplished recently is the migration project of a French company's general ledger applications for accounting, payrolls and stock management running on BULL DPS6 platform under GCOS6 operating system. The programs are in COBOL and used to run under TP4 transaction monitor. Fig. 4 illustrates a general view of the system which was built on UFAS data files. Interactions with users were provided through the screen FORMS supplied with TP4 transaction monitor. Fig. 5 shows the renovated environment for this legacy system. Programs run under AIX/ORACLE on a RS6000 platform. The FORMS screen management system is replaced by a Unix Graphical screen management interface. The legacy data (UFAS data files) is fully normalized and migrated to the ORACLE database environment. The target tables are free of legacy-dependant extra data such as "chaining pointers", etc., so they are easily shared with newly written programs using new technology tools (UNIFACE, etc.). The final application performance is much better than the original legacy environment. The main TP4 transaction monitor function that used to control the transaction commitment and the data recovery is now replaced by the ORACLE database functions. Using automated tools, the whole migration project took about 6 man months for 750 programs.

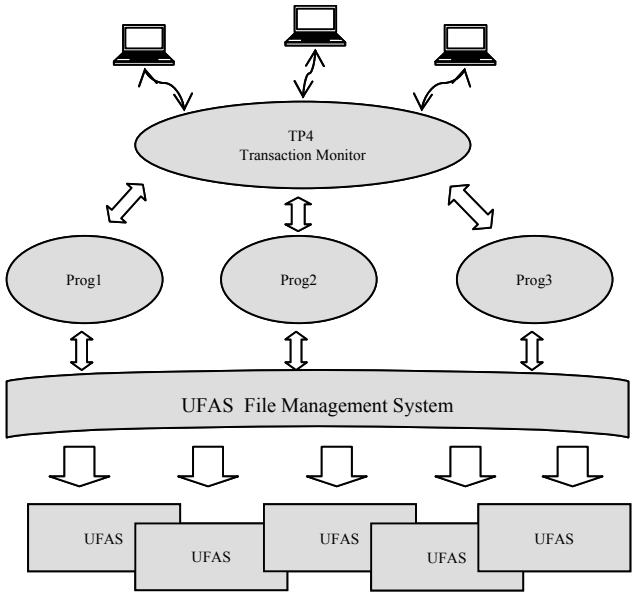

Fig. 4. Sample Legacy System Environment (BULL/GCOS6)

6 Conclusion and future works

Some practical guidelines regarding migration of legacy systems "data+code" is discussed and a technical solution is presented briefly. In this approach we pay attention to

preserving the business logic and making the migrated system free of any constraint for future extensions.

The efficiency of the solution has been tested on multiple migration projects in many different environments. We have also gained very successful experiences regarding migration of the legacy database environments such as CODASYL databases, etc., that we hope to discuss in a future paper.

Technical issues regarding the automation of migration process for specific legacy components such as JCL, different programming languages, and the optimization and implementation of data normalizations/unifications still need to be developed in more details.

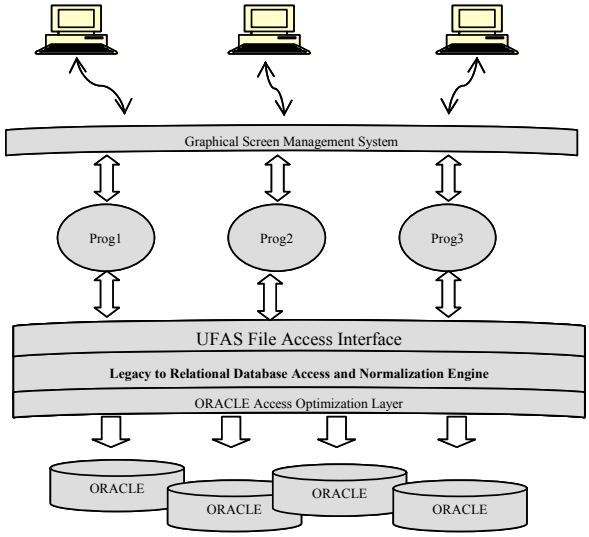

Fig. 5. Sample Legacy System Renovation Environment (UNIX/ORACLE)

References:

[1] Bisbal Jesús, Lawless Deirdre, Wu Bing, and Grimson Jane: Legacy Information Systems: Issues and Directions, *IEEE Software*, September/October 1999.

[2] Blaha, M.R., Premerlani, W., J.: Observed Idiosyncrasies of Relational Database designs, in *Proc. of the 2nd IEEE Working Conference on Reverse Engineering*, Toronto, IEEE Computer Society Press, July 1995.

[3] Casanova, M., Amarel de Sa, J. : Designing Entity Relationship Schemas for Conventional Information Systems, in *Proc. of Entity-Relationship Approach*, pp. 265-278, 1983.

[4] Casanova, M., A., Amaral De Sa.: Mapping uninterpreted Schemes into Entity-Relationship diagrams: two applications to conceptual schema design. *IBM J. Res. & Dev.*, Vol. 28, No 1, 1984.

[5] Comella-Dorda Santiago, Wallnau Kurt, Seacord Robert C., Robert John: *A Survey of Legacy System Modernization Approaches*, Carnegie Mellon University,Tech. Note CMU/SEI-2000-TN-003, 17 August 2000, URL:http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html

[6] Davis, K., H., Arora, A., K.: A Methodology for Translating a Conventional File System into an Entity-Relationship Model. *Proceedings of ERA*, IEEE/North-Holland, 1985.

[7] Deursen Arie van, Klint Paul, Verhoef Chris : Research Issues in Software Renovation. In J.-P. Finance, editor, Proceedings Fundamental Approaches to Software Engineering (FASE99), pages 1-23. *Lecture Notes in Computer Science, Springer-Verlag,* 1999.

[8] Deursen Arie van, Elsinga Ben, Klint Paul, Tolido Ron: *From Legacy to Component: Software Renovation in Three Steps*, CAP Gemini Institute (http://www.cs.vu.nl/~daan/cwicap/) - CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands http://www.cwi.nl/~paulk/publications/CAP00.pdf, 2000.

[9] Edwards H. M., Munro M.: Deriving a Logical Model for a System Using Recast Method, *Proceedings of the 2nd IEEE WC on Reverse Engineering*, Toronto, IEEE Computer Society Press, 1995.

[10] Hainaut Jean-Luc, Chandelon M., Tonneau C., Joris M. : Transformational techniques for database reverse engineering, *Proceedings of the 12th International Conference on ER Approach*, Arlington-Dallas, E/R Institute and Springer-Verlag, LNCS, 1993.

[11] Hainaut Jean-Luc: *Database Reverse Engineering*, University of Namur - Institut d'Informatique rue Grandgagnage, 21 l B-5000 Namur (Belgium), http://www.info.fundp.ac.be/~dbm, 1998.

[12] Nilsson,E., G.: The Translation of COBOL Data Structure Rel-type Conceptual Schema. *Proceedings of ERA Conference*, IEEE/North-Holland, 1985.

[13] Rahgozar Maseud, and Oroumchian Farhad: Classification and guidelines for Legacy Systems' Renovation Issues. The 10th Electrical Conf. of Iran (IEEE), Iran Electrical Engineering Society, University of Tabriz,Tabriz, Iran, May 14/16, 2002.

[14] Rahgozar M., Oroumchian F. 2002. A Practical Approach for Modernization of Legacy Systems., *EuroAsian Conference on Advances in Information and Communication Technology (ICT 2002) - Workshop on Recent progress in Computers and Comunications*, Tehran, Iran, 29-31 Oct.

[15] Rahgozar M., Oroumchian F. 2002. Transformational Approach for Legacy Systems evolution. Submitted for publication in: *2002 WSEAS International Conference on Applied Mathematics and Computer Science (AMCOS'02)*, Copacabana, Rio De Janeiro, October 21-24, 2002.

[16] Sabanis, N., Stevenson, N.: Tools and Techniques for Data Remodelling Cobol Applications. *Proceedings of the 5th International Conference on Software Engineering and Applications,* Toulouse , 7-11 December 1992, pp. 517-529, EC2 Publish.

[17] Weiderman Nelson H., Bergey John K., Smith Dennis B., Tilley Scott R.: *Approaches to Legacy System Evolution* , Carnegie Mellon University, Tech. Note CMU/SEI-97-TR-014, 1997, URL:http://www.sei.cmu.edu/publications/documents/97.reports/97tr014.html