A Transformational Approach for Legacy Systems' evolution

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 evolution of the Legacy Information Systems (LIS) is a critical issue for many organizations world wide. A Large number of organizations for their daily activities depend on the business critical applications that have been developed over the last two decades or more. They mostly run on old software and hardware technology tools and environments. They are hard to modify, expensive to maintain and difficult to integrate with new technology tools and programs. They need to be evolved into modern environments. This paper suggests guidelines for an optimal transformation of legacy systems into Unix-RDBMS architectures, based on many years of professional experiences of the authors in the area.

Key-Words: - Legacy Applications, Legacy Databases, Transformation, Migration, Normalization, Evolution.

1 Introduction

A large number of existing applications running on mini and mainframe platforms are programs developed since 1970's. These applications are mostly written in 3GL programming languages such as: COBOL, RPG, PL1, FORTRAN, BASIC, PASCAL, C, etc. [1][19]. Others are written in 4GL programming languages like: PACBAS, DELTA, etc.. Some of these languages again generate the mentioned 3GL programs before compilation. All these applications, called Legacy Information Systems (LIS), are using either the simple data implementations such as indexed file systems or the old database technologies such as IDMS, CODASYL, NETWORK, etc., [11][9][8][12].

These invaluable assets of encoded "business logic" represent many years of coding, developments, real-life experiences, enhancements, modifications, debugging, etc. [12]. Unfortunately, they represent also many years of bad documentation or no documentation at all. They are well known by their dominant characteristics of "resisting modification and evolution", and "running on obsolete hardware that is slow and expensive to maintain" [2]. Even if, the applications' documentation was perfectly up to date, redeveloping these systems would still be estimated unaffordable in terms of time, costs, and needed human resources. [19]. Since, they are vitally important for enterprise business continuation, they need to be evolved in to new technology environments and run on modern platforms. [2].

The following section presents the related works and reviews their shortcomings. Section 3 makes suggestions and provides guidelines for an effective approach. 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. The implementation issues of such solutions will be described in section 4.

2 Related works

Issues regarding LIS evolution (i.e., modernization, renovation or migration) have been the research and development topics for a while. Many approaches to LIS problems have been worked out. The list of related works would be too long. However, the state of the art may be found in some recent publications such as: [19][12][2][6][5][7]. Classification of the existing methods can be found in [13]. There, we presented wrapping as the first category of current approaches to LIS issues. With wrapping we mean surrounding the old LIS components (data, code or user interface) with new interfaces or programs such that any access to the old system goes through these interfaces. These methods try to keep the existing system as they are on the original platform. The second and third categories are based on changing the platform by either redeveloping the system or migrating the old system to a new environment. The redevelopment approaches can start from scratch or can be combined with the reverse engineering of databases and/or programs' codes. The migration approaches can be with or without new added values on the old LIS components (data, code or user interface).

2.1 What is missing with current approaches?

In spite of the diversity among the different solutions [3][4][10][16][17][18], the current situation in LIS renovation approaches can be summarized as below:

- Database reverse engineering is sufficiently mature to be applied in practice. [12]
- The reverse engineering of procedural components of a large application is still unsolved. [12][7]
- Wrapping solutions are short-term solutions that can complicate LIS maintenance and management over long time. [2]
- Redevelopment approaches are considered risky for most organizations. [2][19]
- Existing approaches to migration are too high level and there is a lack of literature on successful practices. [2]
- Most of the database reverse engineer literature examines solutions for migration of relational databases. While, the market is more concerned with migration of: indexed files, IMS, COBOL and CODASYL data. [12]

Most widely adopted approaches to LIS problems tend to offer short term solutions to long term problems. They mostly fail to recognize that the base of an optimal solution to LIS problems should be to **reuse** LIS components as much as possible and to **recover** those invaluable assets of "data + business logic" as a whole.

Many consider moving legacy applications from IMS, CODASYL and Indexed-Files environments to modern systems (UNIX) and modern database environments (RDBMS) *as a considerably complex and risky activity* [12]. In our experience, this move is not only possible but it is the most viable option. We believe the move to an open system environment is the long-term solution to LIS problems. In the next section we offer guidelines for such an approach.

3 General Guidelines for an effective approach

In [2] authors state that there are few comprehensive approaches to migration and the current literature contains no successful, practical experience report from projects using a comprehensive migration approach. They proclaim that a set of comprehensive guidelines to drive migration would be essential, and a promising research direction would aim to identify different types of legacy systems and develop specific migration process and methodologies for each.

Below, we enumerate the guidelines we have achieved through many years of experiences on LIS renovation projects. We believe following these guidelines will lead to effective solutions in LIS renovation projects.

a) **Avoid short term solutions:** Any short-term solution that leads to maintain the obsolete legacy platform and to add any new complex interface upon the existing legacy environment should be avoided. Solutions such as wrapping may have an appeasing effect at the beginning, but they do not address the real problems of LIS systems such as maintenance costs, system rigidity and aging technology. They add to the complexity and rigidity of the system and prevent searching for real solutions. We believe that any long term solution will need at least: the change of platform to a native Open System, the use of a native compiler on the target platform, the use of native Graphical User Interfaces tools and the transfer of the data to the target native database system with the normalization of the data. These issues will be addressed in the rest of this section.

b) **Avoid partial solutions:** Any partial solutions that leads to the recovery of only one of the LIS components: "user interface", “data” or "programs' codes" (i.e., “business logic”), should be avoided. We believe that the "data" and the "programs' codes” are the two major components of the LIS. The solutions that do not recover both of these components faithfully, will lead to the loss of invaluable parts of the legacy systems. Solutions such as reverse engineering are not advanced enough to recover fully both the "data" and "business logic" as a whole.

c) **Do not alter the code logic:** Keep in mind that the invaluable assets of "business logic" are mostly encoded in programs. So, the programs’ codes should be recovered without any alteration in their logic. It is unrealistic to consider any automatic tool to extract the “business logic” encoded in programs, in all situations. However, the syntax of the code may go through some changes to conform to the target platform's native system functions (e.g., TP-Monitor, Database, Operating System, etc.) or the syntax of the target native compiler. Meanwhile one should be cautious to avoid undesired structural and functional changes in the programs' codes that could alter the embedded “business logic”.

d) **Avoid creating new bugs in the code:** Any unsafe actions, such as manual modifications, on the programs’ codes should be avoided. They would lead to creation of new bugs in the programs that had finally been cleaned up along the years of business practices. The changes in the programs' codes should be done through the automated translation programs initially and then double checked by engineers as far as possible. Manual modifications in the large legacy codes are subject to inconsistencies, human errors, undocumented changes and ad-hoc solutions. Automatic translations are more systematic and predictable thus easier to undo. In case of double checking by engineers, translation errors due to exceptional situations would be easily detected and corrected. The preferred method of correction is by correcting the translator to handle the exception correctly.

e) **Avoid environment emulation solutions:** Any "emulation" of legacy systems' environments should be avoided. The emulation solution refers to executing a legacy program's binary code on a target machine by a run time interpreter. Moving the legacy programs' binary codes (as they are) to the "virtual legacy machines" on the new platforms (such as UNIX) brings no new value to the old situation other than switching the hardware. They lead again to a complex and limited programming environment. Although, such a solution is unavoidable in some situations where the old hardware is no longer supported, we do not recommend it as a final and long term solution.

f) **Avoid emulation of data formats:** Any "emulation" of legacy data formats should be avoided. Such data cannot be easily used within new technology tools. That will create serious limitations and constraints for sharing legacy data with future programs. For example there are 36 bit long integers in some legacy platforms that are not available on UNIX. Emulating such an integer type in UNIX will only complicate the matter for future applications. However, converting them to native UNIX integer types once for all will solve such a problem.

g) **Normalize the data implementation design:** For most legacy systems, the designs and implementations of the legacy data need significant changes to become fully normalized. That's a must for sharing legacy data with future programs. It should also be prepared for future extensions of the unified Information System. This means dealing with issues such as: splitting or atomizing of legacy data items, adding new data items or splitting, joining and merging of records into tables. The legacy data normalization addresses many important design and performance issues [14].

h) **Pay particular attention to the indexed-files:** For many legacy systems, the data stored in indexed files are at least as important as those stored in the database tables. They need particular attention because they need extensive normalization and reformatting before being moved to the relational data bases. They have to be set together with other database data in a unified Information System schema. Their normalizations, new designs and implementations raise many challenges regarding system performance and optimization [15].

i) **Do not alter data access logic inside the code:** Avoid any alteration in database and file access statements inside the programs' codes. The structure and logic of legacy programs is strongly tied to the legacy data access logic. 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 (if not fatal) performance degradation. An effective transformation of the legacy data access logic to relational data access logic is not linear. The situation becomes worse if one tries to normalize and change the new relational definition of data. This would require significant changes in the structure of the legacy programs' codes. However, sometimes this kind of normalization is unavoidable. The effective solution to this situation is to create a specialized data access interface. For this, the legacy data access logic should be considered as a whole and only managed through this interface. This data access interface should be external to the legacy program and should handle both normalization and data access transformation [15].

j) **Do not create unnecessary data:** Avoid creating any unnecessary legacy data such as "record pointers" in the new Information System schema. The physical notations like Areas, Sets, Chaining pointers etc are normally used in legacy systems. The replication of those notations in the new transformed system will bring unnecessary physical dependencies to the old logic. This will create serious limitations and unacceptable constraints for the future programs accessing legacy data. This also leads to performance degradation of the new system.

4 Implementation

4.1 LIS Components

Our focus is on a subset of legacy systems that is the transaction oriented applications. The methods discussed here may or may not be applicable to the real time or the embedded systems. Before considering the implementation issues, we will look at the different constituents of a legacy system. Then, we study the actions to be performed on each component either in isolation or in conjunction with the other components. The set of components bellow covers both classes of interactive and batch programs:

a) **Program source codes:** Legacy programs are mostly written in 3GL languages such as COBOL, RPG, FORTRAN, BASIC, PASCAL, C, etc. There are also programs in some 4GL languages such as PACBAS, DELTA, MANTIS, etc. that are mostly the COBOL derived languages with supper macro verbs. Since, most of these 4GL languages generate COBOL programs before compilation; they can be treated with the same solutions as COBOL programs.

b) **Transaction Processing (TP) Monitors:** Interactive legacy programs are mostly written to work with TP Monitors, i.e., they are structured to use the services such as: Inter Process Communication, Terminal I/O, Transaction Commitment, etc. In such cases, programs are structured differently compared to the legacy batch programs, i.e., the structure of their code is "loop-structured" and not linear as in the batch programs. Each loop contains two steps: one of which is the interaction with the TP Monitor and the other one is the related processing part of the program code corresponding to the current state of the interactions.

c) **System functions:** Some System services are provided through the calls to system functions (or intrinsics). They cover multiple areas such as: File System Operations, Database Operations, Job Control and Operating System Commands, etc.

d) **Screen Management System:** Most of the interactive legacy programs use simple text screens to dialog with the users. These non graphical screens are either managed by the TP Monitors, or by separate Screen Management Tools and Libraries. In most cases, their definitions are integrated in the syntax of the programming languages (e.g., Forms Sections in COBOL).

e) **Indexed Data Files:** For the elderly legacy programs, the Indexed Files are either the only means of storing the legacy data, or the major elements of data manipulations. The structures of such programs are totally different from those of programs working on Database tables. We will discuss this issue later in this paper.

f) **Database tables:** For the newer legacy programs, the main supports for storing data are the Database tables that are created in the legacy database environments such as: CODASYL, NETWORK, HIERARCHICAL, etc. These programs are structured totally differently compared to their Relational Database counterparts. We will discuss this issue later in this paper.

g) **Job Control files (JCL) and O/S Command scripts:** For batch legacy programs, some part of the code for data file manipulations and process scheduling are written in the Job Control Language (JCL) provided by the Operating System. The major operations performed in JCL files and O/S Command scripts are those for creating, indexing, copying, merging and sorting of data files.

4.2 Implementation

We present the implementation methodology that we have successfully applied on some LIS renovation projects. The methodology is based on an "intelligent transformation" of all components of the legacy system while the "business logic" encoded in legacy programs is preserved. In this method the legacy components are adapted to work on the native environment of an Open System (UNIX or NT) and to take advantage of new technology tools (GUI, RDBMS, etc.). This implementation is broken-down into the following actions to take on the different components of LIS:

a) **Program sources codes:** The legacy codes are to be translated into the new system's codes. It is important to note that this step should be automated as much as possible. If the legacy system language is supported in the new environment, the same language must be used as target language. However, if the legacy language is not supported in the new system, the best target language would be C. To do this, we will need some translation tools. If the tools are already available, they can be used. Otherwise, we have to write a translator. It is worth noting that there are some automation tools already available in the market. The existing tools mostly have limitations and shortcomings and it is very important to pay a careful attention in selecting the proper translation tools. We have developed a set of tools to automate this step for a few languages. The performance and reliability of these tools have met the expectations. The legacy code translation should respect the guidelines (c), (d) and (e) explained in the previous section. As we will explain bellow, for the legacy data moved to RDBMS tables, the data access statements within the programs' codes should be replaced with the calls to the equivalent functions. These library functions provide the same access logic, but on the new RDBMS environment. The replacement of data access statements can also be done by the same translation tool. But, for better readability of the programs' code, we recommend to keep these statements as they are, and to do the replacement just before each compilation. This action can be done easily and automatically by a specific pre-compiler.

b) **Transaction Processing (TP) Monitors:** Most of the functions supported by TP Monitors are easily translated to UNIX equivalent functions. Some specific functions of TP Monitors for distributed systems can also be created or replaced with the equivalent functions provided by UNIX TP-tools (such as Tuxedo). It is important to provide the legacy programs with a transparent "system" and "TP" functions and to avoid unnecessary modification of programs' codes. Two of the main functions provided by TP Monitors are the “Concurrency Control” and “Data Recovery Control”. In the new environments these functions are easily translated to their equivalent RDBMS functions. Some other TP Monitor functions such as inter-programs communications are supported in UNIX without the need to any Transaction Monitors.

c) **System functions:** Similar to the functions supported by TP Monitors, the system functions providing the operations on the File Systems, the Databases, the Job Controls, the Operating System Commands, etc., are easily created or translated to UNIX equivalent functions.

d) **Screen Management System:** A Graphical Legacy Logic User Interface (GLLUI) should be created to provide legacy programs with a transparent highly featured user interface. The structure of the programs' codes should be maintained unchanged as far as possible. Usually, a new look is the minimum expectation of users when a legacy system is moved to new environment. Changing the UI of a legacy system carries no risk in terms of business logic performance, it rather adds to the usability of the programs by providing them with the new functionalities available in today’s GUI tools

e) **Indexed Data Files:** Since, Indexed files are widely used in the legacy systems, they need particular attention. They have to be reformatted, normalized and moved to the relational database tables. They have to be set together with the other legacy data in a unified Information System schema. We should also create a Legacy indexed File Access Logic Interface (LFALI) to deal with these legacy data effectively within the programs. LFALI provides the legacy programs with a transparent high performance file access management on the new RDBMS environment. The transparent facet of LFALI provides a uniform file access interface regardless of where the data is stored (i.e., in a database table or in an ordinary indexed file). Harmonizing all legacy data is required for translating them to the new RDBMS tables. This effort leads to taking advantage of the advanced data management facilities provided in RDBMS environments, such as: concurrent transactions management, data access security, data recovery, etc. Successful implementation of this step also requires respecting the same guidelines as those given bellow for the database tables.

f) **Database tables:** The design of the legacy data stored in the legacy databases has to be normalized and prepared for future needs and future extensions. That is to preserve a unique definition of Information System including legacy data and future data. It is important to avoid creating “record pointers” (first / last, father / son, next / previous) in the new implementation of the legacy data. That is to avoid any constraints on future programs accessing legacy data. We should create a Legacy Data Access Logic Interface (LDALI) that supports the access logic of the legacy system in the new RDBMS environment. The LDALI is composed of tools and library functions and provides the legacy programs with a transparent high performance data access on new RDBMS environment. A successful implementation of this step requires following the guidelines (f), (g), (i) and (j) explained in the previous section. The legacy data formats should be translated to native UNIX/RDBMS formats, using the modern data manipulation tools provided with the RDBMS environments. The migrated data must be fully shareable using new technology tools such that the future programs can access them without any constraints.

g) **Job Control files (JCL) and O/S Command scripts:** The JCL files and the O/S Command scripts are almost easily translated to native UNIX shell files and UNIX command scripts. Once they are in native UNIX commands, they give full access to the new technology tools and the facilities available on the new environment, such as: debugging tools, maintenance tools, system and database tools, etc. It is recommended to keep their structure close to the original ones. This may require creating some "Legacy-Like" UNIX shell commands and tools for creating, indexing, copying, merging and sorting data files with the similar arguments to the original platform. It is important to note that all the operations on indexed files performed through JCL commands has to be replace by their equivalent operations in the RDBMS environment.

The normalization of the legacy data, their new design and implementation in RDBMS environments raise many challenges regarding system performance that need to be discussed further in a separate paper. All the translation tasks have to be done using automated tools. Details on the implementation and development stages of these tools are out of the scope of this paper.

5 - Conclusion

Many medium and large organizations are in the process of renovation or planning for renovation of their legacy systems in near future. The approach such organizations are mostly forced to take is the redevelopment of their legacy systems. These organizations have to make huge investments in programming their business logic while this already exists within their legacy code. These investments could be best spent on extending their business logic rather than recoding it again.

This paper, we have presented a classification of current approaches to LIS renovation. The main problem with most of these approaches is that they do not offer an effective and long term solution that preserve the data and business logic. Here, we have suggested general guidelines and a methodology that avoids the cited problem. These guidelines are based on our many years of experiences in LIS domain. This methodology has been proved successfully on multiple legacy environments. Many aspects of this methodology already is and could be more automated. The details of these aspects are the subjects of separate papers.

References:

[1] Bennett K. 1995, Legacy Systems, *IEEE Software*, Jan., pp. 19-73.

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

[3] Brodie M., Stonebraker M. 1995, *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*, Morgan Kaufmann, San Francisco.

[4] Cimitile A., De Lucia A., Di Lucca A., Fasolino A.R. 1997. Identifying Objects in Legacy Systems, *Proceedings of the 5th Workshop on Program Comprehension (WPC97)*.

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

[6] Deursen Arie van, Klint Paul, Verhoef Chris 1999. 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*.

[7] Deursen Arie van, Elsinga Ben, Klint Paul, Tolido Ron 2000. *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

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

[9] Fong J. Ho M. 1994. Knowledge-based Approach for Abstracting Hierar-chical and Network Schema Semantics, *Proceedings of the 12th Int. Conference on ER Approach*, Arlington-Dallas, Springer-Verlag

[10] Haft T. M., Vessey I. 1995. The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension, *Information Systems Research*, 6, pp 286-299.

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

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

[13] Rahgozar M., Oroumchian F. 2002. Classification and guidelines for Legacy Systems' Renovation Issues. *The 10th Iranian Conference on Electrical Engineering*, Iran Electrical Engineering Society (IEE) and IEEE, University of Tabriz,Tabriz, Iran, May 14/16.

[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. Automatic Evolution of of Legacy Data Objects. 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] Stets Robert J., Hunt Galen C., Scott Michael L. 1999. Component-Based APIs for Versioning and Distributed Applications, *IEEE Computer*, 54-61, July.

[17] Seacord Robert C., Wallnau Kurt, Robert John, Comella-Dorda Santiago, Hissam Scott A. 1999. Custom vs. Off-the-Shelf Architecture, *Proceedings of 3rd International Enterprise Distributed Object Computing Conference*, University of Mannheim, Germany, September 27-30.

[18] Von Mayrhauser A., Vans A.M. 1994. Comprehension Processes During Large Scale Maintenance, *Proceedings of the International Conference of Software Engineering ICSE*. Sorrento, Italy, p.39-48. May 16.

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