A legacy information system represents a massive, long-term business investment. Unfortunately, such systems are often brittle, slow, and nonextensible. Capturing legacy system data in a way that can support organizations into the future is an important but relatively new research area. The authors offer an overview of existing research and present two promising methodologies for legacy information system migration.

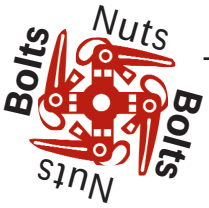
Legacy Information Systems: Issues and Directions

Jesús Bisbal, Deirdre Lawless, Bing Wu, and Jane Grimson,
TRINITY COLLEGE DUBLIN

Legacy information systems are typically the backbone of an organization's information flow and the main vehicle for consolidating business information. They are thus mission critical, and their failure can have a serious impact on business.[1] In fact, a LIS can be defined as "any information system that significantly resists modification and evolution."[2] They can cause host organizations several problems:

- LISs usually run on obsolete hardware that is slow and expensive to maintain.
- Software maintenance can also be expensive, because documentation and understanding of system details is often lacking and tracing faults is costly and time-consuming.
- A lack of clean interfaces makes integrating LISs with other systems difficult.
- LISs are also difficult, if not impossible, to extend.

NUTS & BOLTS: Karl Wiegers and Dave Card, editors • kwiegers@acm.org / card@computer.org



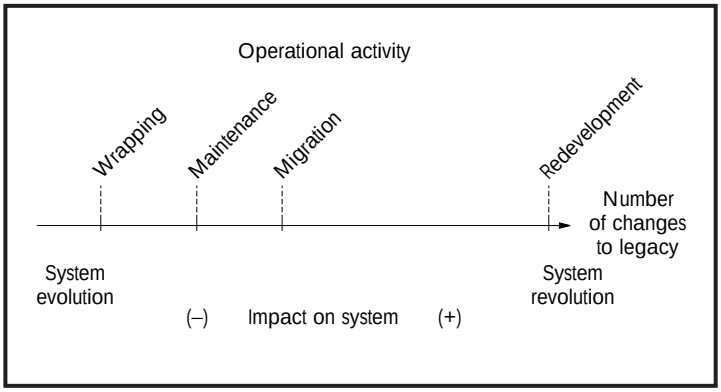


Figure 1. Solutions to LIS problems range from the relatively mild wrapping to redevelopment, which can completely change the system or create an entirely new system.

Several solutions have been proposed to these problems. These solutions fall generally into three categories: *redevelopment*, which rewrites existing applications; *wrapping*, which provides a new interface to a component, making it more easily accessible by other software components; and *migration*, which moves the LIS to a more flexible environment, while retaining the original system's data and functionality.

Given the scale, complexity, and risk of failure in LIS projects, a well defined, easily implemented, and detailed methodology is essential to project success. However, few comprehensive LIS migration methodologies are available, and a general approach has yet to be agreed on. Existing approaches are either too high level or have yet to be applied in practice.[3–5] Although partial solutions such as wrapping are widely adopted, such solutions are short-term and can actually complicate LIS maintenance and management over the long term. On the other hand, redevelopment approaches tend to be too risky for most organizations. We thus advocate migration as a sound strategy for contending with LISs. Here, we offer an overview of existing LIS strategies, then discuss migration issues and present two promising approaches for migrating LIS data.

LIS Coping Strategies

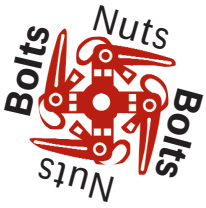
Figure 1 shows different approaches commonly used to cope with LISs. We add maintenance only for completeness, because it is part of every system's life cycle. In fact, if a software system can be maintained within an acceptable budget it is usually not considered a LIS.[6] As the figure shows, each approach varies in terms of changes required and costs and risks involved. Redevelopment leads to the most changes (system revolution) and wrapping the least (system evolution).

However, given a concrete LIS problem, it is not always possible to categorize the solution according to one approach. For example, wrapping can be seen as a maintenance activity that aims to make other components oblivious to changes in the wrapped component; migration can involve wrapping part of the system, maintaining another part, and redeveloping still another, and so on. Thus, although we describe these approaches as being applied at a system level, they are typically applied at a component level.

We divide the various approaches into three general categories. The first, redevelopment, requires the LIS to be shut down either during development or during cut-over to the replacement system. In this category, we include both redevelopment and reengineering, as the sidebar "Classifying Terms" explains. The second approach, wrapping, is less drastic, but often serves only as a short-term solution. The third approach, migration, is a middle ground, offering a long-term solution while mitigating the risk of system shutdown.

Much research and consulting attention has been given to the problem of organizations attempting to reinvent themselves. But what about those whose businesses are successful but need to adapt to meet the challenges of evolving technology and market requirements? Many of these organizations have substantial investments in legacy information systems. Replacing these systems involves not only the direct financial cost of software development but also the expense of rediscovering their accumulated knowledge about business rules and processes. This article describes several practical approaches to leveraging an organization's legacy investments so that it can evolve, rather than having to reinvent itself entirely.

—David N. Card and Karl Wiegers, Nuts & Bolts editors



Redevelopment

Redevelopment, commonly referred to as Big Bang and also known as Cold Turkey,[2] redevelops the LIS from scratch using a new hardware platform and modern architecture, tools, and databases. As an example, the Renaissance research project is aimed at developing a systematic method for system evolution and reengineering.[4] The project defines a set of activities and tasks to support an overall reengineering project and the control flow between the identified activities, which drives the cooperation among tasks. Renaissance identifies generic activities that can be specialized for both the organization and the system it is implementing.

Scott Tilley proposes a high-level framework for LIS reengineering from several perspectives.[3] For each perspective, Tilley divides the reengineering problem into phases, listing pertinent issues for each one. Although this approach does offer some guidance, it is far too high level to be practically applied.

Narsim Ganti and William Brayman[7] propose general guidelines for transforming from a centralized legacy environment to a distributed environment. In this approach, select business processes are reengineered as required and then linked to LISs, which provide valued data and business logic to the new environment. New applications are then developed to fit these processes. This approach recognizes that LIS migration should cause as little disruption as possible in the current business environment. However, it is not clear how cut-over to a separately developed target system would be handled.

In reality, the risk of failure is usually too great for organizations to seriously contemplate a redevelopment approach. Another very real concern stems from the fact that technology and business requirements are constantly changing. Thus, at the end of a long process, an organization might find itself with a redeveloped system based on obsolete technology that no longer meets its business needs.

Wrapping

Given the drawbacks of redevelopment, many organizations are forced to seek alternative ways to cope with their LISs. Most practical solutions focus on wrapping, which surrounds existing data, individual programs, application systems, and interfaces with new interfaces. In essence, this gives old components new operations or a "new and improved" look.[6] The wrapped component acts as a server, performing some function required by an external client that does not need to know how the service is implemented.[8] Wrapping lets organizations reuse well-tested components that they trust and leverage their massive investments in the LIS.

The most widely used implementation of wrapping is *screen scraping*, which replaces a LIS's character-based front end with a client-based graphical user interface.[1] Implementing the GUI cheaply and effectively leverages legacy data and lets users employ common graphical data manipulation tools to

Classifying Terms

Although "reengineering" is often used as a synonym for migration,[1,2] we include it in the redevelopment category as most reengineering efforts propose complete system reimplementation. In our view, to reengineer a system is to examine (understand) and alter a system so as to reconstitute it in a new form.[3] Thus, reengineering ultimately leads to an (almost) complete reimplementation of the LIS.[4–6] The resulting system might or might not run in a different computing environment. Also, although redevelopment involves developing a system from scratch, it requires a thorough understanding of the existing system and thus involves many reengineering activities.

We thus view reengineering as closer to redevelopment than to migration, which aims to avoid the long and costly implementation process. Migration seeks to reuse as much of the LIS as possible, including implementation, design, specification, and requirements. Also, the target system resulting from a migration process runs in a different computing environment, whether it is a different programming language or a completely new architecture and technology.

If most of the LIS must be discarded, the engineer will be facing a redevelopment project, not a migration project. Ultimately, in our view, reengineering is not a solution to the LIS problem per se, but rather a technology to be used in migration or redevelopment projects.

References

1. H.M. Sneed, "Encapsulating Legacy Software for Use in Client/Server Systems," *Proc. Third Working Conf. Reverse Eng.*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 104–119.
2. D. Aebi, "Data Reengineering: A Case Study," C.J. van Rijsbergen, ed., *Proc. Advances in Databases and Information Systems (ADBIS97)*, Springer Verlag, Berlin, 1997.
3. E.J. Chikofsky and J.H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, Jan./Feb. 1990, pp. 13–17.
4. N. Ganti and W. Brayman, *Transition of Legacy Systems to a Distributed Architecture*, John Wiley and Sons, New York, 1995.
5. S.R. Tilley and D.B. Smith, *Perspectives on Legacy System Reengineering*, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1996.
6. "Renaissance Project—Methods and Tools for the Evolution and Reengineering of Legacy Systems," Esprit Project, Lancaster University, Lancaster, UK, 1997; http://www.comp.lancs.ac.uk/computing/research/cseg/projects/renaissance/RenaissanceWeb/ (current, July 1999).

input data and process system output.

Despite screen scraping's commercial success, it is still very much a short-term solution. Implementing a GUI in a LIS does not address many of the serious problems such systems face, including overloading, static functionality, and high maintenance costs. In many cases, screen scraping actually compounds an organization's maintenance problems, as the functionally superfluous screen-scraping software will itself require maintenance.

Migration

When redevelopment is unacceptably risky and wrapping is unsuitable, migrating the LIS to an open environment can be the best alternative. Although it is a much more complex undertaking than wrapping, if successful, migration's long-term benefits are also greater. For example, migration offers more flexibility, better system understanding, easier maintenance, and reduced costs.

Although migrating LISs is a major research and business issue, there are few comprehensive approaches to migration. Given the bewildering array of LISs in operation and the problems they pose, it seems unlikely that a single generic migration method would be suitable for all systems. However, a set of comprehensive guidelines to drive migration is essential.

Before embarking on a migration project, engineers, management, and users should undertake an intensive study to find the most appropriate approach for solving their organization's LIS problems. To the best of our knowledge, the literature contains no successful, practical experience reports from projects using a comprehensive migration approach. The few (successful) migration-like projects reported in the literature describe ad hoc solutions to the problem.[9,10]

LIS migration issues

LIS migration essentially moves an existing, operational system to a new platform, retaining the legacy system's functionality and causing as little disruption to the existing operational and business environment as possible. This is a significant challenge, and it could quite legitimately encompass numerous areas of software engineering, including program and database understanding, system development, and testing.

Figure 2 shows important practical issues in migration, divided roughly according to those related to the LIS and those related to the target system. Some migration issues are common to all software-engineering projects and are widely researched and supported. These include target system development, testing, and database model selection. Other issues are specific to migration and have yet to be extensively researched. These include target system database population and cut-over with mission-critical support.

Because a LIS already meets some of the business and user requirements demanded of the target system, it is important to understand its operations and interactions. Poor LIS understanding can lead to incorrect target-system requirement specifications and ultimately to failed migration projects. Thus, to begin, engineers should have a good understanding of the LIS data, interfaces, and applications that require tool support.[1,11] Although some support is available, engineers may have to develop specialized tools to fit their LIS and target systems.[10] They might also classify their LIS by type and properties, and develop appropriate migration guidelines.[12]

Database population

To populate the target database with LIS data, engineers first map the LIS schema onto the target schema[13] and work out the required transformation. Data must also be mapped at instance level[14] before migration. Data can also be migrated in separate steps, by dividing it into independent fragments. If LIS data is of poor quality, data cleaning might be required.[10] If so, decisions must be made about which method to use[15] and when to use it: before, during, or after migration.

Testing and functionality issues

Up to 80 percent of a migration engineer's time can quite legitimately be spent testing the target system,[1] which is an ongoing process during migration. Given the legacy system's mission-critical nature, target system outputs must be completely consistent with those of the LIS. Thus, it is inadvisable to introduce new functionality to the target system during the migration project.[1,16,17] When functionality is the same, engineers can directly compare outputs to determine the target system's validity. However, on a practical level, migration projects are often expected to add functionality to justify the project's expense and risk. In this case, the LIS should be migrated first. New functionality can be incorporated into the target system after the initial migration.

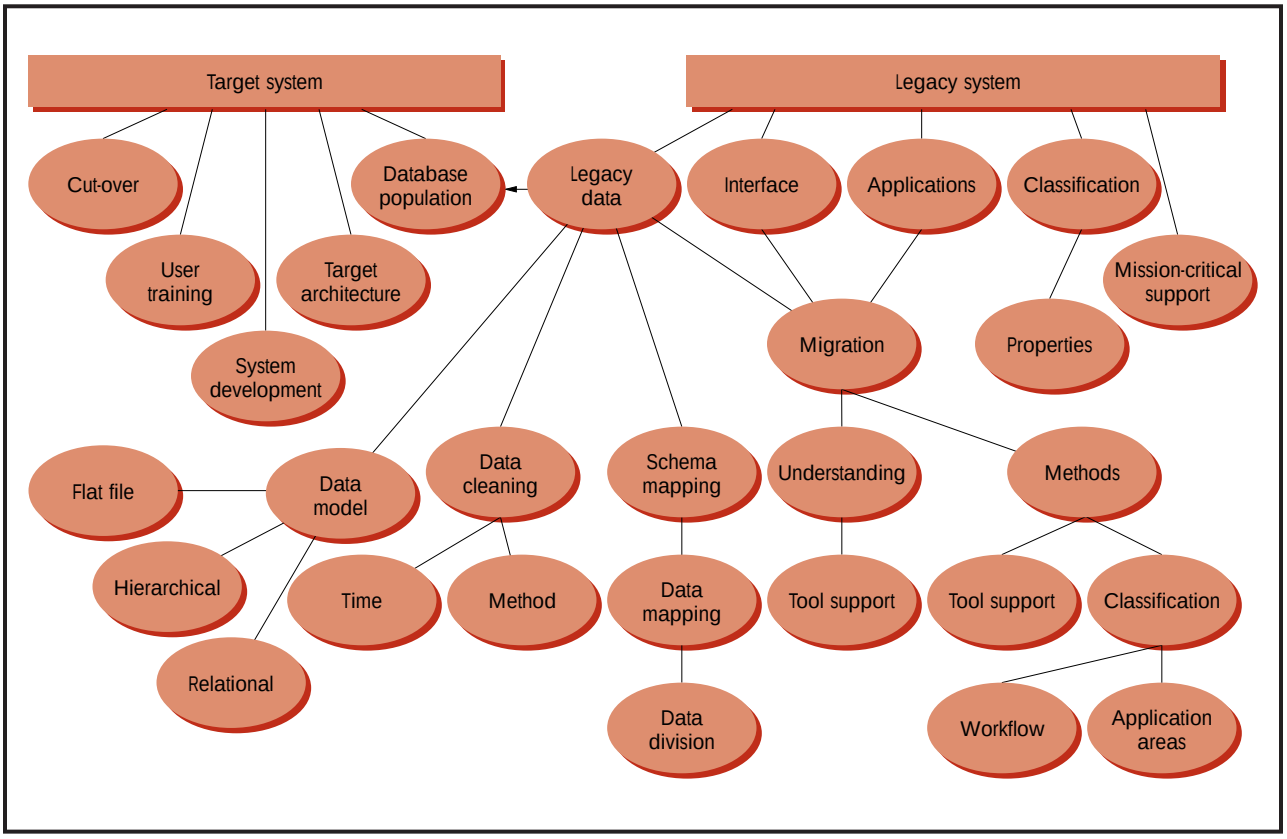


Figure 2. Classification of LIS migration issues. The issues common to software engineering projects, such as testing and database model selection, have been widely researched; those related specifically to migration, such as populating the target system's database, have received less attention.

Cut-over

The last step in the migration project is the cut-over from the LIS to the target system. Three different transition strategies have been proposed:[17]

1. The *cut-and-run* strategy consists of switching off the LIS and turning on a new feature-rich replacement (see Figure 3a).

2. With the *phased interoperability* strategy, the cut-over is performed in small, incremental steps: each step replaces a few LIS components (applications or data) with corresponding target components (see Figure 3b).

3. In the *parallel operations* strategy, LIS and target systems operate simultaneously, with both systems performing all operations. During this period, the target system is continually tested; once it is fully trusted, the LIS is retired.

The cut-and-run strategy is, in many cases, unrealistic because of the risk: cutting over to the target system in a single step puts the organization's whole information flow in an untried and thus untrusted system. On the other hand, phased interoperability is potentially highly complex. To be successful, this method requires the migration engineer to split LIS applications into functionally separate modules or to separate the data into portions that can be independently migrated. The monolithic and unstructured nature of most legacy systems makes such a step-wise approach difficult, if not impossible.

In addition, the management of such a process would be challenging, as it is likely to involve heterogeneous environments and distributed applications and databases.[13,18] Although incremental LIS migration is designed to reduce migration-phase risk, its inherent complexity might actually increase the overall migration risk.

Each of these proposals is an active research area and may not be mature enough for use in a mission-critical environment. At this point, a concrete transition strategy for a particular migration project would probably involve a combination of these ap-

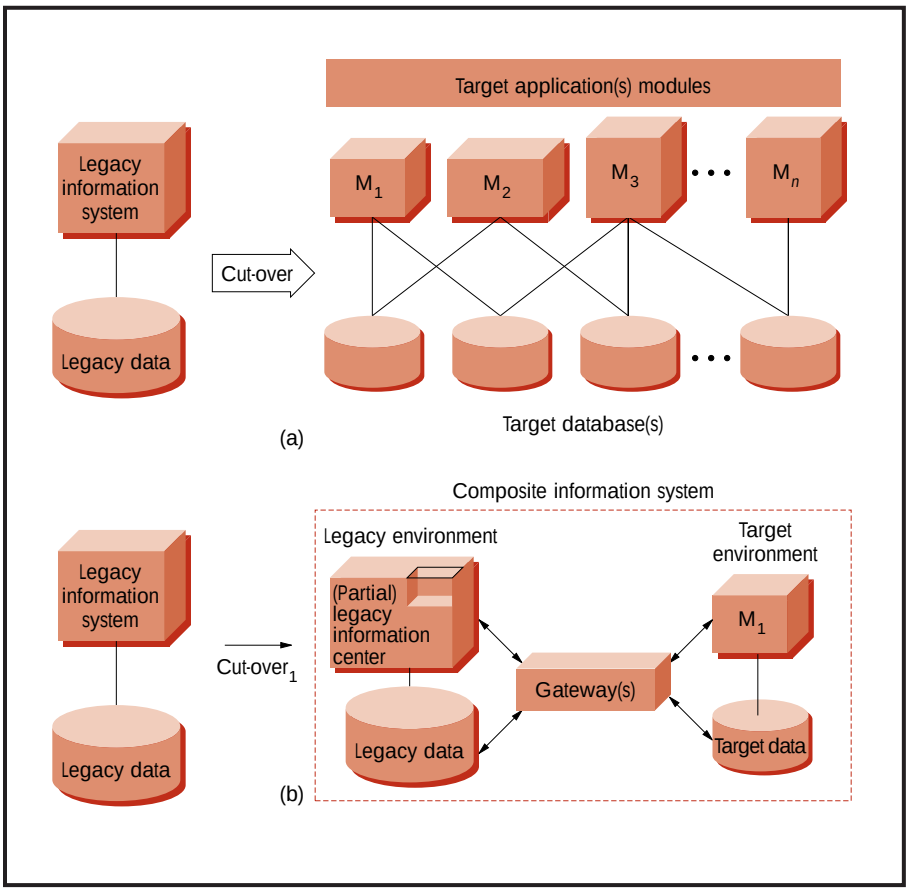


Figure 3. Legacy information system cut-over is typically performed using a combination of the (a) cut-and-run strategy and the (b) phased-interoperability strategy.

proaches, applied to different LIS components. More research is needed to identify the best strategies for the cutting-over phase, which is central to migration project success.

Migration methods

Current approaches to LIS problems tend to offer short-term solutions to long-term problems. They also fail to recognize that the essence of migration is to reuse LIS components in the replacement system, and to operate the legacy system while the replacement system is being developed.

Many migration approaches, including the Chicken Little approach described later, let the legacy and its replacement system interoperate via a gateway. While such approaches can be successful, gateways suffer from serious limitations. For example, such approaches

- offer no support for transaction management and thus no way to ensure data consistency between the legacy and target systems;
- provide no way to homogenize the structural and representational differences between the two database schemas; and
- are difficult to build and operate.

Our recognition of such problems led us to develop the Butterfly Methodology, which addresses many software development topics, including target system development and testing, user training, and other complex tasks unique to migration. Above all, however, we recognized that data is an important business resource and thus focused our efforts on developing a method to support data migration.

The Chicken Little Strategy

Michael Brodie and Michael Stonebraker propose the Chicken Little strategy, which lets LIS and target systems interoperate during migration using a mediating module, generally known as a "gateway."[2] The Chicken Little strategy offers an 11-step plan for cutting over from the LIS to the target system. Each step is incremental.

1. Analyze the LIS.
2. Decompose the LIS structure.
3. Design the target interface.
4. Design the target application.
5. Design the target database.
6. Install the target environment.
7. Create and install necessary gateways.
8. Migrate the legacy databases.
9. Migrate the legacy applications.
10. Migrate the legacy interfaces.
11. Cut over to the target information system.

Chicken Little is a refinement of the Composite Database Approach[2] (see Figure 4), which is also known as stepwise migration.[17] Using this strategy, LIS applications are gradually rebuilt on the target platform using modern tools and technology. The target system is initially quite small, but grows as the

migration progresses. Eventually, the target system performs all the functionality of the LIS, which can then be retired. During migration, the LIS and target systems form a composite information system.

Chicken Little uses a forward gateway to translate and redirect calls to the target database service and to translate the target database's results for use by LIS applications. A reverse gateway maps the target data to the LIS database. This mapping can be complex and slow, thus affecting new applications. Also, many of the complex features found in modern databases (such as integrity, consistency constraints, and triggers) might not exist in the archaic LIS database and hence cannot be exploited by new applications.

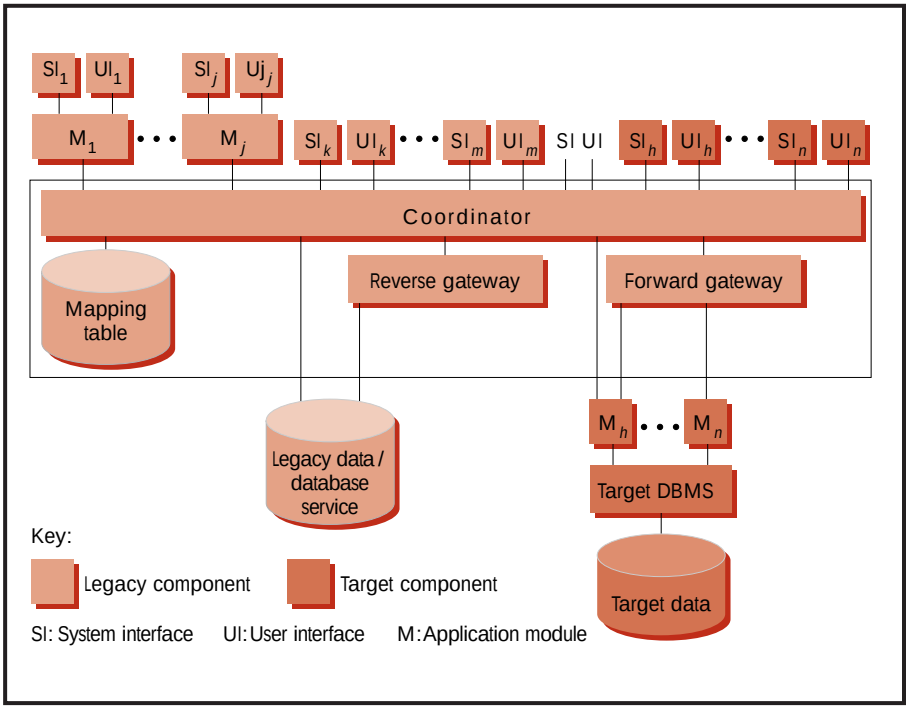


Figure 4. The Chicken Little strategy is a refinement of the composite database approach, which gradually rebuilds LIS applications on the target platform.

The Chicken Little approach can use data duplicated across the LIS and target databases. To maintain data integrity and consistency, a transaction coordinator intercepts all update requests from LIS or target applications and identifies whether they refer to data replicated in both databases. If they do, the update is propagated to both databases using a two-phase commit protocol (as in a distributed database[18]). As Brodie and Stonebraker themselves point out, update consistency across heterogeneous information systems is a complex technical problem and an open challenge for researchers.[2] Thus, although the Chicken Little approach has the advantage of breaking the migration process into a series of well-designed stages, it can involve highly complex strategies to ensure consistency between the target and LIS databases.

The Butterfly methodology

Our Butterfly methodology[5] assumes that although the LIS must remain operable throughout migration, the LIS and target system need not interoperate during the process. This assumption eliminates the need for gateways and their potential complexity.

Successfully migrating the data management service from the LIS to the target system is the key to overcoming many LIS problems.[2,10] The Butterfly methodology thus focuses on LIS data migration, and develops the target system in an entirely separate process.[19]

Once data migration commences, the LIS data store is set to read-only. The data access allocator (DAA) redirects LIS data manipulations; the results are stored in a series of auxiliary data stores, or *TempStores*. Figure 5 shows the migration process using Butterfly methodology. While the legacy data is being migrated, the DAA stores all manipulations to the first TempStore (TS_1). When the legacy applications issue a data request, the DAA will retrieve the correct data from either the legacy data or TS_1. Once LIS data has been migrated, the target database contents must be updated with the data stored in TS_1, which must also be migrated.

The Chrysaliser data transformer migrates LIS and TempStores data to the target system. When Chrysaliser is migrating the legacy data, all manipulations are stored in TS_1; when it is migrating TS_1 data, manipulations are stored in TS_2; and so on. If the Chrysaliser migrates a TempStore faster than the LIS can build the next one, the TempStore size decreases at the next iteration.

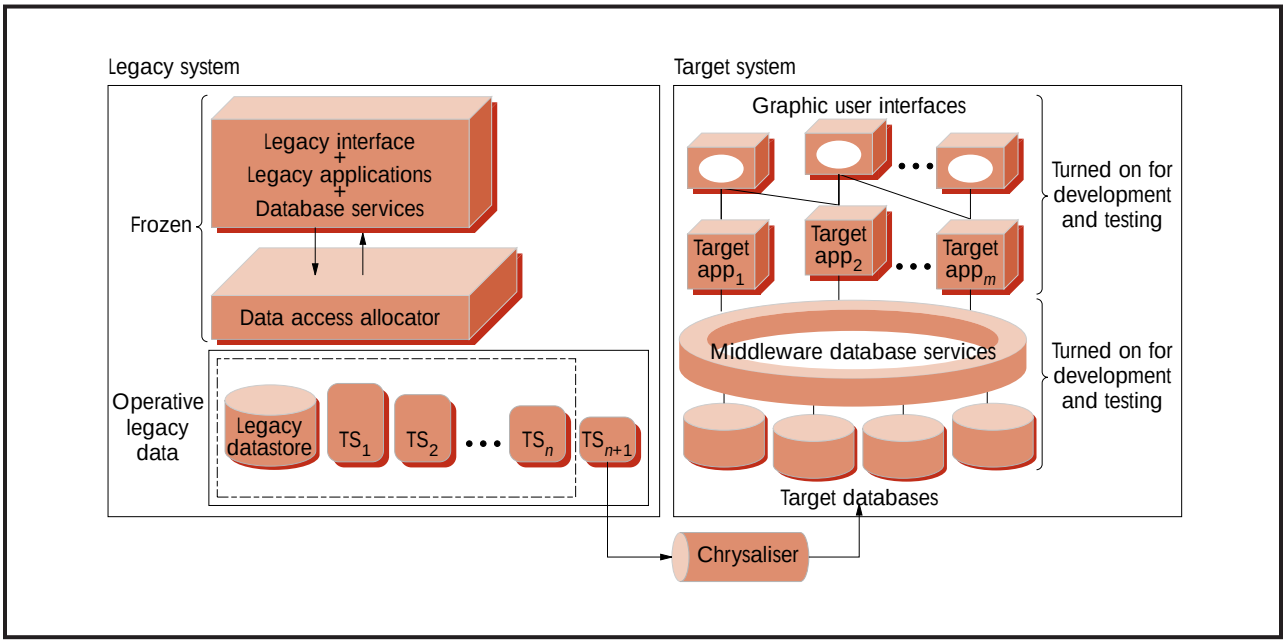


Figure 5. The Butterfly methodology splits target system construction into a separate task, and focuses on migrating data. The Chrysaliser migrates the LIS data, stored in TempStores (TS_1...TS_n), to the target system. The data access allocator populates the TempStores with legacy data.

In the Butterfly methodology's final step, the DAA and Chrysaliser act as a data migration engine for the LIS data migration. Throughout migration, the LIS operates normally until the size of the last TempStore reaches a predetermined threshold value, such that the amount of time necessary to migrate this last TempStore is sufficiently small to let the LIS be brought down without causing serious inconvenience to the core business. When this condition is reached, the legacy system can be switched off (frozen), the last TempStore migrated, and the target system turned on, having reached data consistency with the LIS.

Using the Butterfly methodology, the LIS is accessible for all but a very brief period of time and never needs to interoperate with the target system. Also, at any stage prior to cut-over, the process is reversible and migration can be safely stopped.

Because LIS migration has received scant attention in the research community until relatively recently,[1] all aspects of migration require more study. Methodologies such as Chicken Little and Butterfly address some key migration issues and are a useful starting point for further research.

Another promising research direction aims to identify different types of legacy systems and develop specific migration processes and methodologies for each.[12] A next step is to identify and develop specific tools to support these processes and aid migration engineers in performing time-consuming and error-prone tasks.

Finally, we need project reports to support all areas of LIS migration. Such practical case studies can provide a better understanding of the legacy problem in general and the migration process in particular. These reports would also help identify appropriate classifications for legacy systems, and thus further aid the development of specific processes and tools to address the problems inherent in each. ❖

Acknowledgments

We thank Ray Richardson and Donie O'Sullivan from Broadcom Éireann Research and Vincent Wade from the

Computer Science Department in Trinity College Dublin for their insightful comments on previous versions of this article.

References

1. K. Bennett, "Legacy Systems," *IEEE Software*, Jan. 1995, pp. 19–73.
2. M. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*, Morgan Kaufmann, San Francisco, 1995.
3. S.R. Tilley and D.B. Smith, *Perspectives on Legacy System Reengineering*, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1996.
4. "Renaissance Project—Methods and Tools for the Evolution and Reengineering of Legacy Systems," Esprit Project, Lancaster Univ., Lancaster, UK, 1997; http://www.comp.lancs.ac.uk/computing/research/cseg/projects/renaissance/RenaissanceWeb (current, July 1999).
5. B. Wu et al., "The Butterfly Methodology: A Gateway-Free Approach for Migrating Legacy Information Systems," *Proc. Int'l Conf. Eng. Complex Computer Systems (ICECCS '97)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 200–205.
6. N. Weiderman et al., "Implications of Distributed Object Technology for Reengineering," Tech. Report CMU/SEI-97-TR-005, Carnegie Mellon Univ., Pittsburgh, 1997.
7. N. Ganti and W. Brayman, *Transition of Legacy Systems to a Distributed Architecture*, John Wiley and Sons, New York, 1995.
8. H.M. Sneed, "Encapsulating Legacy Software for Use in Client/Server Systems," *Proc. Third Working Conf. Reverse Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 104–119.
9. A.J. O'Callaghan, ed., *Practical Experiences of Object Technology*, Stanley Thornes Publishers, Cheltenham, UK, 1996.
10. D. Aebi, "Data Reengineering: A Case Study," C.J. van Rijsbergen, ed., *Proc. Advances in Databases and Information Systems (ADBIS97)*, Springer-Verlag, Berlin, 1997.
11. J.L. Hainaut et al., "Database Design Recovery," *Proc. Conf. Advanced Information Systems Eng. (CAiSE96)*, Lecture Notes in Computer Science 1250, Springer-Verlag, Berlin, 1996, pp. 272–300.
12. P. Stevens and R. Pooley, "Software Reengineering Patterns," *Proc. SIGSOFT'98 6th Int'l Symp. Foundations of Software Eng.*, ACM Press, New York, 1998.
13. S.T. March, ed., Special Issue on Heterogeneous Databases, *ACM Computing Surveys*, Vol. 22, No. 3, 1990.
14. S.B. Davidson and A.S. Kosky, "WOL: A Language for Database Transformations and Constraints," *Proc. 13th Int'l Conf. Data Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 55–65.
15. K. Kukich, "Techniques for Automatically Correcting Words in Text," *ACM Computer Surveys*, Vol. 24, No. 4, 1992, pp. 377–439.
16. B. Beizer, *Software Testing Techniques*, second ed., Van Nostrand Reinhold, New York, 1990.
17. A.R. Simon, *Systems Migration—A Complete Reference*, Van Nostrand Reinhold, New York, 1992.
18. D. Bell and J. Grimson, *Distributed Database Systems*, Addison Wesley Longman, Reading, Mass., 1992.
19. J. Bisbal et al., "Building Consistent Sample Databases to Support Information System Evolution and Migration," *Proc. Database and Expert Systems Applications (DEXA98)*, Lecture Notes in Computer Science 1460, Springer Verlag, Berlin, 1998, pp. 196–205.

About the Authors

Jesús Bisbal is a doctoral student with the Department of Computer Science in Trinity College Dublin. He is currently working on the Synex project, a European Union funded project that is developing an open, generic, and secure means for sharing healthcare records and related medical data. Previously, he worked on the Milestone project, which set out to develop a methodology for legacy information systems migration. His research interests include legacy migration, health informatics, database theory, and data mining. Bisbal received his BA and BAI degrees in computer science from Technical University of Catalonia, Spain. Contact him at Jesus.Bisbal@cs.tcd.ie.

Deirdre Lawless is a systems analyst for Coras Iompair Eireann, Ireland's state-owned national transport company. After earning a BSc in computer science from University College Dublin, Ireland, she worked in the software industry for several years. She recently completed her MSc degree in research with the Department of Computer Science in Trinity College Dublin. Contact her at Deirdre.Lawless@cie.ie.

Bing Wu is a lecturer in the Department of Computer Science at Dublin Institute of Technology. His current research interests include legacy system migration, health informatics, modeling and engineering of distributed information systems, and application of active databases. He received his BSc and MSc in computer science from the National University of Defence Technology, China, and his PhD in computation from the University of Manchester Institute of Science and Technology, England. Contact him at bwu@maths1.kst.dit.ie.

Jane Grimson is an associate professor and founder of the Knowledge and Data Engineering Group at the Department of Computer Science, Trinity College, Dublin, where she is also co-chair of the Centre for Health Informatics and the dean of Engineering and System Sciences. Her research focuses on heterogeneous, distributed multimedia databases, knowledge-based systems, and healthcare informatics. She is a fellow of the Irish Computer Society, the British Computer Society, and the Institution of Engineers of Ireland, the latter of which she will serve as president in 1999-2000. She is also a member of the ACM and the IEEE, and a Chartered Engineer. Grimson holds a BA and BAI in engineering from Trinity College Dublin, a MSc in computer science from the University of Toronto, and a PhD in computer science from the University of Edinburgh. Contact her at Jane.Grimson@cs.tcd.ie.

Address questions about this article to Bisbal at the Knowledge and Data Engineering Group, Computer Science Department, Trinity College Dublin, Dublin 2, Ireland; Jesus.Bisbal@cs.tcd.ie.