SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2008; **38**:1333–1364
Published online 26 February 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.870

Developing legacy system migration methods and tools for technology transfer

Andrea De Lucia[1,*,†], Rita Francese[1], Giuseppe Scanniello[2] and Genoveffa Tortora[1]

[1]*Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA), Italy*
[2]*Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo, Macchia Romana, 85100 Potenza, Italy*

SUMMARY

This paper presents the research results of an ongoing technology transfer project carried out in cooperation between the University of Salerno and a small software company. The project is aimed at developing and transferring migration technology to the industrial partner. The partner should be enabled to migrate monolithic multi-user COBOL legacy systems to a multi-tier Web-based architecture. The assessment of the legacy systems of the partner company revealed that these systems had a very low level of decomposability with spaghetti-like code and embedded control flow and database accesses within the user interface descriptions. For this reason, it was decided to adopt an incremental migration strategy based on the reengineering of the user interface using Web technology, on the transformation of interactive legacy programs into batch programs, and the wrapping of the legacy programs. A middleware framework links the new Web-based user interface with the Wrapped Legacy System. An Eclipse plug-in, named MELIS (migration environment for legacy information systems), was also developed to support the migration process. Both the migration strategy and the tool have been applied to two essential subsystems of the most business critical legacy system of the partner company. Copyright © 2008 John Wiley & Sons, Ltd.

Received 23 March 2007; Revised 12 October 2007; Accepted 3 December 2007

KEY WORDS: legacy systems; migration to the Web; reengineering; technology transfer

*Correspondence to: Andrea De Lucia, Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA), Italy.
†E-mail: adelucia@unisa.it

Contract/grant sponsor: MiUR (Ministero dell'Università e della Ricerca); contract/grant number: PRIN-2006-2006098097
Contract/grant sponsor: TOCAI (Tecnologie Orientate alla Conoscenza per Aggregazioni di Impresa in internet); contract/grant number: FIRB 2005

Copyright © 2008 John Wiley & Sons, Ltd.

1. INTRODUCTION

Many legacy systems are business critical and can operate up to 24 h a day. They are written in some legacy language such as COBOL and are often dependent on other applications in their environment. Several options are available for replacing such legacy systems. Typical solutions include discarding the legacy system and building a replacement system, freezing the system and using it as a component of a new larger system, and modifying the system to give it a new lease on life. Changes may range from a simplification of the system through a reduction in size and complexity, to preventive maintenance operations such as redocumentation, restructuring, and reengineering, to an adaptive maintenance process entailing interface modification, wrapping, and migration [1]. These alternatives are not mutually exclusive, and the decision as to which approach, or combination of approaches, to take is generally based on an assessment of the quality and business value of the system [2–5]. However, often other non-technical factors influence the decision as to how to deal with legacy systems, factors such as the need to move to a modern Internet-based infrastructure in order to remain competitive in the global market [6]. In this case, system migration is the only viable alternative [6], since the risk of replacing the legacy system might be unsustainable [7]. This is mainly due to the effort required to redevelop the system from scratch when its documentation is lacking, the business logic is encoded in the programs, and the original implementers are no longer available [8].

Generally, the migration of a legacy system is a complex task, which is influenced by several concerns. One concern, pointed out by Brodie and Stonebraker [8], is that the migration of a legacy system depends on its decomposability. A legacy system is classified as being decomposable, semi-decomposable, or non-decomposable, depending on how its user interface, application logic, and database software components are separated. Application logic, databases, and user interfaces are inseparable in non-decomposable systems, whereas in semi-decomposable systems the user interfaces are separated from the application logic and the database. On the other hand, application logic, databases, and user interfaces are all separable in decomposable systems. Of course, the less the system decomposability, the more difficult the migration will be [8]. The separation of the presentation logic from the application logic and the database access logic is a labor-intensive activity [9], which depends on the quality of the system and the number of interactive programs. The encapsulation of the legacy system using wrapping technologies is a viable alternative to preserving past investments and reducing risks and development costs [10–14]. Legacy information systems or part of them can be encapsulated in a modernized set of legacy components, thus enabling the integration with newly developed or purchased applications through the wrapper interface. This approach also enables an incremental migration of the original system [6,8,15–17].

In this paper, we present the research results of an ongoing technology transfer project conducted in cooperation with a small Italian software company—MTSys srl. The project goal is to develop and transfer to the partner company methods and tools to migrate multi-user COBOL legacy systems to a multi-tier Web-based architecture. To identify the most suitable migration strategy, we first performed an assessment of the legacy systems of the partner company in terms of business value and software quality. The assessment revealed that these systems had a low decomposability level and spaghetti-like code. It also unveiled several complex problems, such as embedded control flow with database accesses in the user interface description. As a result of the assessment, we decided to adopt an incremental migration strategy aimed at wrapping the legacy system at the user interface level.

The adopted migration strategy is based on the reengineering of the user interface using Web technology, the transformation of interactive legacy programs into batch programs, and the wrapping of the programs. A communication middleware enables the new client to interact with the wrapped legacy programs. The middleware is a generic component developed so that it can be used in the migration of any legacy system, thus enabling the software engineer to focus only on the reengineering of the user interfaces and on the wrapping of the legacy programs. We also decided to develop an Eclipse plug-in, named migration environment for legacy information systems (MELIS), to fully support the migration of the legacy systems of our partner company according to the phases of the proposed strategy. The migration strategy and tool have been then applied within a pilot migration project conducted on two essential subsystems of the business partner. The pilot project was carried out by two teams, each composed of an academic researcher and a practitioner with the purpose of evaluating the migration process and tool in the company environment. Indeed, the final goal of the project is to make the partner company an owner of the MELIS plug-in and the accompanying methods as well.

The remainder of this paper is organized as follows: related work is discussed in Section 2; the proposed migration strategy is described in Section 3; and the migration process and supporting tool are outlined in Section 4. The evaluation of the migration process and supporting tool is presented in Section 5. Discussion and final remarks conclude the paper.

2. RELATED WORK

Several approaches have been presented during the past two decades for the migration of monolithic and procedural legacy systems to distributed architectures such as client–server architectures [13,18,19], distributed object architectures [11,12,15,20], and Web-based architecture and service-oriented architecture (SOA) [9,10,21–25]. Encapsulation is one of the three main alternatives to reuse legacy systems within distributed software architectures. The other two are reengineering and redevelopment. Encapsulation involves the least costs and the least risks reusing existing programs and databases with a minimum of change [7,26]. Encapsulating legacy systems usually requires reengineering the legacy user interface using technologies of the new environment in which the legacy system has to be accessed. User interface reengineering involves reverse engineering to abstract a user interface conceptual model and forward engineering to re-implement the user interface. Several approaches based on data-flow analysis [27], state transition diagrams (STDs) [10,28], knowledge engineering [29,30], and business process information [31] have been proposed to reengineer legacy user interfaces.

To encapsulate legacy systems at different granularity levels, wrapping techniques can be adopted. In general, four kinds of wrappers [32] have been identified in the literature depending on what is being wrapped: database wrappers, system services wrappers, application wrappers, and function wrappers. Database wrappers are gateways to existing databases. They allow newly developed applications to access data stored in a legacy database. System service wrappers provide a customized access to standard system services. User programs can also invoke such services without the knowledge of their internal interfaces. Application wrappers encapsulate batch processes or online transactions to allow new client applications to include the legacy components as objects. Finally, function wrappers offer an interface to invoke individual functions within a wrapped program. Wrappers are also used in the reference architecture proposed by Zdun [25] for bringing a legacy application to the Web. This architecture takes into account several other aspects, including authentication, session management, dynamic content creation, and presentational abstractions. In this way the author provides a conceptual understanding of which components are required for reengineering a larger system to the Web.

The idea of wrapping existing software components for reuse in a new architecture did not come out of the research community. Indeed, the idea of encapsulating legacy systems in wrappers comes from industry, where it was born as a child of expediency. However, wrapping technologies are now mature and include commercial solutions, i.e. WebSphere Studio Enterprise Developer [33] offered by IBM. For example, this tool can be used to develop Struts-based Web applications that access existing COBOL programs running on CICS.

As a rule, wrappers should use some kind of communication middleware to connect themselves to the user applications. On the input side, the wrapper receives incoming requests, whereas on the output side, it takes the results from the Wrapped Legacy System and sends them back to the user program. This is, in essence, what wrapping is all about [26]. Different wrapping approaches have been presented in the literature, which differ mainly in how they support the migration strategy and in the technology they use to encapsulate the legacy software. Generally, there are two types of approaches to wrap legacy systems: the white-box and the black-box. Weiderman [34] categorized the white-box and the black-box wrapping techniques as follows: ‘the white-box transformation encompasses a form of reverse engineering that emphasizes deep understanding of individual modules and internal restructuring activities. The black-box transformation encompasses a form of reverse engineering that emphasizes shallow understanding of module interfaces and wrapping activities’. However, there is no universally best way to wrap legacy systems or part of them [14]. The choice of a wrapping technique depends on the program type, the availability of source code, and its quality.

2.1. Black-box wrapping approaches

Lin *et al.* [12] propose a black-box wrapping technique to reuse MS-Windows software as a CORBA object. These ready-made applications rarely have their source code available and most of them are non-decomposable; thus, they can be regarded as black-box entities. The wrapper implementation requires the wrapper to redirect the input/output data streams and to generate a CORBA interface. The windows task input channel is redirected using an event message simulation for mouse and keyboard devices, whereas the wrapped application saves the output data in the clipboard space using the output redirection mechanism. The CORBA-IDL is adopted to create the CORBA interface. Wrapping through black-box technique is also used in [35] for integrating COTS MS-Windows applications in a distributed system using Java technologies. The proposed architectural style supports a three-phase process, where the components are first encapsulated, constructing a server-side Java object by wrapping an MS-Windows application, secondly, by including a coordinator to integrate the wrapped applications, and thirdly by constructing a user interface implemented in Java to manipulate the integrated applications.

A similar approach is presented in [36]. The authors propose a process consisting of three phases (i.e. adaptation, encapsulation, and integration) to reuse the functionality of MS-Windows applications. MS-Windows applications can be easily encapsulated so that software engineers can integrate them using a specific standard three-tier framework, such as CORBA or Java RMI. To this end, three design patterns I/O adapter, wrapper façade, and coordinator have been adopted.

The stepwise usage of these patterns creates a processing sequence of reusing applications into a new software system. Similarly, Goedicke and Zdun [37] propose a pattern-based approach to wrap legacy components as black-box entities. Different patterns are proposed to wrap a system at different granularity levels. An architectural pattern language is used to generate flexible black-box architectures. To assess the migration strategy and the wrapping techniques, the authors also propose a pilot project on a C legacy system for Windows NT.

Bovenzi *et al.* [10] use wrapping technology to transform character-based user interfaces to any Web-based client device. This approach exploits a black-box technique for capturing the dynamic and static models of the user interfaces and reproduces them on the client devices with the support of a software wrapper. XML and XSLT technologies are used to reproduce the user interface of the original legacy system. The wrapper is designed to satisfy service stability, data integration, and application integration requirements. STDs are used to specify the behavior of the wrapper: each state of the STD is associated with a different state of the user interface, whereas transitions allow the passage between consecutive states to be defined. The authors propose a methodology and a toolkit to design the wrapper and to support its application. In particular, the tool supports the user in the specification of the STD nodes. An extension of the approach for migrating to SOAs in which the reverse engineer can model the interactions is proposed in [38].

The black-box wrapping techniques have the advantage that they can be reused without intimate knowledge of the component's internals. Unfortunately, incremental migration strategies are not effectively supported as the wrapped component can be neither customized nor adapted. In fact, these strategies require activities (such as user interface reengineering, decomposition and restructuring of legacy programs, and encapsulation of legacy systems at different granularity levels) that can be performed only when the source code is available. In our paper, we also focus on white-box wrapping techniques.

2.2. White-box wrapping approaches

Sneed [14] proposes the use of wrapping techniques and tools to fully automate the wrapping process and provides guidelines on how to wrap batch programs, subprograms, and online transaction (interactive) programs. Batch programs are adapted to read and write XML-documents *in lieu* of files. For subprograms, parameters are set from an incoming XML-message. Finally, online programs are transformed into data-driven subprograms that process an XML-document. Indeed, to wrap an online program, it is necessary to switch off all of those operations that are communicating with the environment. The masks with their fixed fields and attributes are replaced with an XML-type document, whereas the logic of the program is left as it was. The difference with respect to our approach is the communication with the environment, which is implemented using a shared memory.

Sneed [39,40] extends the approach provided in [14] to enable the reuse of legacy components in an SOA. However, this kind of migration is often a complex task and generally requires a careful analysis of the feasibility and magnitude of the effort involved [23]. Indeed, several characteristics of the system such as its language, age, and architecture as well as the target SOA can affect the migration results. The service-oriented migration and reuse technique [23] utilizes this information to identify the risks of a migration project in a systematic way, producing as output a service migration strategy. It also provides decision elements supporting the feasibility of the migration.

Many of the strategies proposed to migrate legacy systems to the Web use white-box wrapping techniques and are conceived for decomposable or semi-decomposable software systems [6,21,22,41]. For example, Aversano *et al.* [6,21] propose integrating an existing COBOL system into a Web-enabled infrastructure. The original system is semi-decomposable, with a client component represented by graphical user interface and a server component represented by COBOL programs including the application logic and database. The graphical user interface is manually migrated to Microsoft ASP Web technologies, whereas the server component is wrapped by means of dynamic link libraries. The difference with respect to our approach is the decomposability of the system to migrate, which in our case is not decomposable; this makes the synchronization between the Web user interface and the wrapped legacy components more complex.

Bodhuin *et al.* [22] describe an approach to migrate decomposable COBOL systems into a Web-enabled architecture based on model view controller (MVC). The software components of the original legacy system are identified using slicing techniques. These components are then restructured and converted into JAVA classes by using the PERCobol tool [42]. These Java classes correspond to a model of the target architecture, whereas the model of the migrated system is represented by JSP pages. Similar to our approach, the Web pages are generated from XML files extracted from the original character-based graphical user interface. Our approach is different because of the non-decomposability of the legacy systems to migrate. Rather than decomposing and restructuring the legacy system to wrap it at the application level, the low decomposability in our case required wrapping the legacy system at the presentation level and designing and developing a generic middleware component enabling the communication and synchronization of the Web user interface and the wrapped legacy code. The same authors [41] present an approach and a tool based on PERCobol to migrate non-decomposable legacy systems to a two-tier Web-enabled architecture. A screen proxy is introduced for handling the requests coming from or going to the user interface. A temporary file is used to enable the communication between the screen proxy and the reengineered graphical user interface. This approach comes very close to our approach, although our approach is suited for migrating monolithic legacy systems to multi-tier Web architectures. Moreover, the authors do not tackle problems related to embedded side effects and control flow in the user interface description. User interfaces in COBOL are described using SCREEN SECTIONs, which define dialogs that ACCEPT and DISPLAY statements use. These statements are used to display output data on the user interface and to receive input data from the user interface, respectively. Furthermore, an ACCEPT statement can trigger ACUCOBOL-GT procedures both before and after the user has filled in a value in an entry field of a SCREEN SECTION. As discussed in the following sections, embedded control flows in the user interface definition make the automatic migration of the legacy user interface very challenging and might require that the software engineer evaluates different alternatives to improve the efficiency of the reengineered user interface.

3. THE MIGRATION STRATEGY

The goal of the project is to devise a strategy and develop the supporting technology to enable the software personnel of the industrial partner to migrate their core legacy systems from a client/server to a Web-based environment. The partner company has been developing and maintaining standard business-oriented software packages for 30 years. It started with the development of COBOL systems for minicomputers in the 1970s. In the 1980s, the company first moved its COBOL development to UNIX workstations and then to personal computers with MS-DOS. The legacy systems developed in the previous decade were migrated to the PC in order to broaden their market segment of smaller users. As part of this migration, the partner transferred its software first to Micro Focus COBOL for MS-DOS in the 1980s and then to ACUCOBOL for MS-DOS in the 1990s. At the end of the 1990s, the ACUCOBOL development environment was upgraded to the ACUCOBOL-GT version, which supports the development of graphical user interfaces for Windows. The ACUCOBOL-GT compiler generates intermediate code from the COBOL SCREEN SECTIONs, which is executed by the runtime environment.

3.1. Assessing the current systems

The first step in preparing a migration should be to assess the current systems. In this project, to identify the most suitable migration strategy, we assessed the quality of the legacy systems with the greatest business value that the partner company has developed and marketed in the years. These systems were selected interviewing the management of our partner company as they had knowledge on the needs of the customers and of the number and costs of the systems' usage licenses. Among the legacy systems that the management identified as business critical for the company, we selected in particular the ones that the customers required to access on the Web.

Concerning the quality of the legacy systems, all the available resources should be considered in terms of documentation, source code, fault history, operational profile, provided services, end users, maintainers, and managers. Owing to the lack of documentation, we could analyze only the source code (static analysis) and the system execution behavior (dynamic analysis) and put clarifying questions to the maintenance personnel who had some knowledge of the systems. Unfortunately, many of the original developers of these systems were no longer working for the company. Thus, we had to use commercial tools to comprehend the legacy systems and to collect various metrics on the software complexity. These tools only partially met our requirements, due to the fact that analysis and reverse engineering tools for the ACUCOBOL dialect were lacking. For this reason, we had to develop some *ad hoc* tools that were later integrated into the migration environment described in Section 4.

The assessment of the legacy systems revealed a very low level of decomposability, due to the fact that the presentation logic was not separated from the application logic and data access logic [8]. This was a result of the compiler constructor's decision to allow application logic and database accesses to be built into the display and accept operations of the SCREEN SECTIONs (presentation layer). This situation, combined with the fact that the code was mainly unstructured with extensive use of GOTOs, forced us to define a migration strategy aiming at reengineering the user interface using Web technologies and wrapping the legacy programs at the presentation level. This strategy requires the definition and the implementation of a communication middleware to connect the new Web user interface with the wrapped programs of the legacy system. It is worth noting that this strategy is the less expensive and risky in the case of non-decomposable legacy systems. Redeveloping the systems was not considered a suitable alternative due to the lack of analysis and design documentation and the high risks involved. Furthermore, the partner company management wished to change as least as possible in order to reach the primary goal, i.e. running the system on the Web.

3.2. Defining the target environment

The next step in preparing a migration is to define the target environment. The target architecture defined here is depicted by the deployment diagram of Figure 1. The interactive programs of the legacy system have to be converted into batch programs (Wrapped Legacy System) and all interactions with the user, i.e. accept and display operations, have to be redirected to the Middleware component, which establishes a communication link between the migrated user interface and the application logic of the legacy system. The user interface is divided into two components, the Reengineered GUI and the GUI Deliverer. The Reengineered GUI includes the Java Server Pages[‡] replacing the SCREEN SECTIONs of the original system, whereas the GUI Deliverer includes the Java Servlets and Beans used to manage the control logic of the new Web user interface and access the functionalities of the Wrapped Legacy System through the Middleware component. The GUI Deliverer accesses the Reengineered GUI to obtain the Web pages required to accomplish a given function. The GUI Deliverer component is accessed by the Web Browser using the HTTPS (HTTP over secure socket layer) protocol.

We planned the implementation of the Middleware as a dynamic link library (DLL) running on the same node as the Wrapped Legacy System. The DLL was developed using the programming language Delphi[§]. To enable the communication between the GUI Deliverer and the Middleware

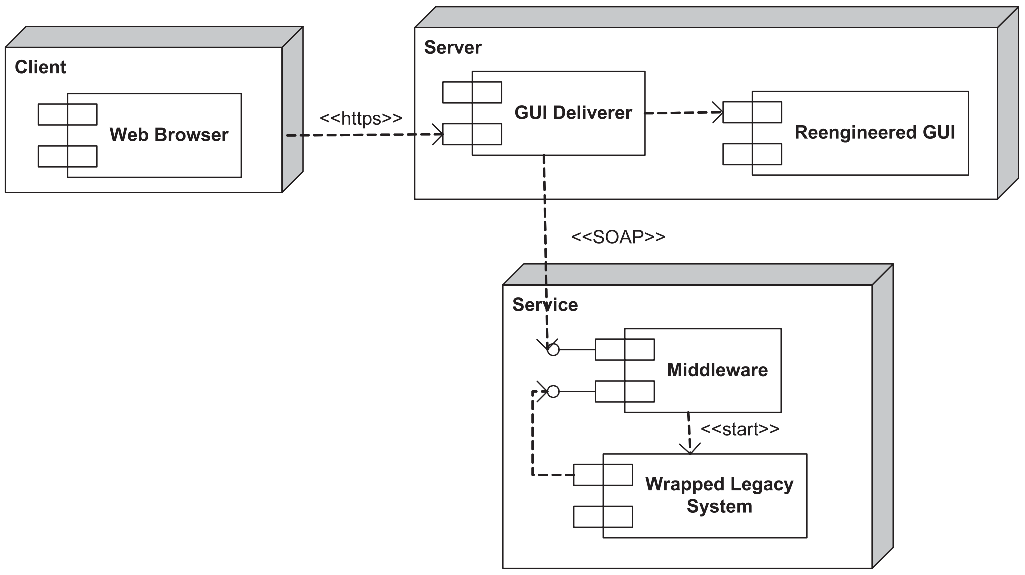


Figure 1. The target software architecture.

[‡]The technologies used for the implementation of the Web-based user interface were imposed as a constraint by our partner company.

[§]Delphi was required by the partner company due to the experience of the practitioners involved in the project.

component, we wrapped the latter using JNI (Java Native Interface). In case the Web server and the Middleware run on the same node, no further communication middleware is needed. Otherwise, when it is required that these components run on different nodes, a communication middleware (e.g. RMI or SOAP) has to be used. In our case, we used SOAP as the partner company wished to access the Wrapped Legacy System over the Web.

The Middleware first starts the Wrapped Legacy System when required. Successively, it manages the communication between the reengineered user interface and the Wrapped Legacy System using a shared memory and semaphores. The Middleware component provides two different interfaces to the wrapped legacy program and the new user interface, respectively. The first interface is provided to the wrapped legacy program to set and obtain information from the Web-based user interface. This interface includes six functions, whose names and corresponding descriptions are given in Table I. To establish a communication link between the new Web user interface and the application logic of the legacy system, the software engineer has to use these functions to redirect the accept and display COBOL operations to the Middleware component.

Figure 2 shows an excerpt of a wrapped legacy program. In particular, it shows how the *display-Cobol* function in Table I is employed to wrap a display operation on the SCREEN SECTION SCREEN-2 of the subsystem GST000 used to evaluate the migration strategy and the tool (see Section 5). The function displayCobol is invoked after having called the section PASSAGGIODATI-SCREEN-2, which is in charge of writing in the shared memory the data to visualize in the Web user interface. On the other hand, an accept operation requires to access only the shared memory to obtain the data coming from the new Web-based user interface. This is performed in the section LETTURADATI-SCREEN-2.

The second interface is provided to the GUI Deliverer software component to access the functionalities of the Wrapped Legacy System. Details on this interface are reported in Table II. Figure 3 shows how the method *GetSharedMem* in Table II is used to obtain the name of the next SCREEN

Table I. Middleware component interface provided to the wrapped COBOL program.

Function	Description
Cblwait	This function suspends the execution of the legacy system until the Middleware component does not return a value. It takes as input parameter the session id that is passed by content.
CblClearSharedMemory	This function removes any data from the shared memory. The input parameter is the session id that is passed by content.
CblSetSharedMemory	This function is used to write data in the shared memory. It takes as input the session id and two strings. These strings represent the variable name and its value. All the variables are passed by content.
CblGetSharedMemory	This function is used to read data from the shared memory. It takes as input two parameters. The former is the session id, whereas the latter is a reference to the data coming from the Web user interface that the Middleware component has previously written.
displayCobol	This function is used to simulate the DISPLAY statement. It takes as input parameters the session id, the name of the legacy program, the screen name, and the display type. All the parameters are passed by value.
CblExit	This function is invoked to stop the communication between the wrapped legacy code and the reengineered Web-based user interface. The input parameter is the session id, which is passed by content.

```
.........
01 SCREEN-2, exception esci-2.
   02 LABEL "Ragione Sociale" size 21 line 5 COL 2
                                  COLOR E-COL-1.
   02 ENTRY-FIELD USING CST000RAGSOC lines 0,5 COL + 2
                                  COLOR E-COL-ACC
                                  BEFORE INIT-RAGSOC
                                  AFTER after-RAGSOC.
.........
main-program section.
.........
* display screen-2
  perform PASSAGGIODATI-SCREEN-2.
  MOVE 'screen-2' to NOME-SCREEN.
  MOVE x'00' to NOME-PROG (7:1).
  MOVE x'00' TO NOME-SCREEN (9:1).
  MOVE 'insert' to op-dll.
  MOVE x'00' to op-dll (7:1).
  set environment 'DLL_CONVENTION' to "1".
  call 'C:\WRAPPER.dll'.
  call 'displayCobol' using by content SESSION
        by content NOME-PROG
        by content NOME-SCREEN
        by content op-dll
        giving RET-DLL.
  cancel 'C:\WRAPPER.dll'.
  set environment 'DLL_CONVENTION' to "0".
* accept screen-2 on exception continue.
  perform LETTURADATI-SCREEN-2.
  perform after-RAGSOC.
  perform after-natgiu.
  perform ctr-sedediv.
  perform ctr-sesso.
.........
PASSAGGIODATI-SCREEN-2 SECTION.
  set environment 'DLL_CONVENTION' to "1".
  call 'C:\WRAPPER.dll'.
  call 'CblClearSharedMemory' using by content SESSION
                              giving RET-DLL.
  move x'00' to name-CST000RAGSOC(13:1).
  move 0 to RET-DLL.
  move CST000RAGSOC to value-CST000RAGSOC.
  move x'00' to value-CST000RAGSOC (100:1).
  call 'CblSetSharedMemory' using by content SESSION
        by content name-CST000RAGSOC
        by content value-CST000RAGSOC giving RET-DLL.
.........
LETTURADATI-SCREEN-2 SECTION.
  move 'N' TO ret-x.
  move '1' to name-CST000RAGSOC (13:1).
  move space to nome-file.
  set environment 'DLL_CONVENTION' to "1".
  call 'C:\WRAPPER.dll'.
  call 'CblGetSharedMemory' using by content SESSION
        by reference nome-file
  giving RET-DLL.
.........
```

Figure 2. Wrapped legacy code.

SECTION to visualize. This figure also shows how the GUI Deliverer redirects the control to the appropriate Web-based subsystem.

The dynamic behavior of a given migrated legacy system in terms of interactions between the user and the software components of the target architecture is depicted by the sequence diagram

Table II. Middleware component interface provided to the GUI Deliverer.

Function	Description
int startCobol(String session)	This method is used to start the Wrapped Legacy System. It takes as input parameter a string containing the session id.
int Sync(String session)	This method is used to make the new user interface in a wait state. The new user interface waits until the Middleware component returns the control. It takes as input a string containing the session id.
int CreateSharedMem(String session)	This method creates the shared memory enabling the communication between the reengineered Web-based user interface and the wrapped legacy code. It takes as input parameter a string containing the session id.
int SetSharedMem(String session, String content)	This method is used to write in the shared memory, thus enabling the communication with the wrapped legacy code. It takes as parameters two strings. The former string represents the session id, whereas the latter contains the data that the legacy code will process.
String GetSharedMem(String session)	This method reads data from the shared memory. It takes as input parameter a string representing the session id. The output parameter is a string containing the whole content of the shared memory. Data in the string are organized in pairs. Each pair is constituted of a variable name and its value. This string also contains the name of the SCREEN SECTION to visualize.
int Exit(String session)	This method is invoked to stop the communication between the reengineered Web-based user interface and the legacy code. It takes as input a string representing the session id.

```
.........
Middleware dllCall = new Middleware();
SharedMemoryCont line = dllCall.GetSharedMem(sessionID);
String nameScreen = line.getNameScreen();
if (nameScreen.compareToIgnoreCase("SCREEN-1")==0){
       gotoPage("/SCREEN1_servletController", request, response);
}
if (nameScreen.compareToIgnoreCase("SCREEN-2")==0){
       gotoPage("/SCREEN2_servletController", request, response);
}
.........
```

Figure 3. An excerpt of the GUI Deliverer component.

of Figure 4. To better clarify how the communication between the reengineered user interface and the application logic of the legacy system is synchronized, we have considered an object for each interface of the Middleware software component. In particular, *Middleware Interface A* and *Middleware Interface B* represent the interface objects provided to the Wrapped Legacy System and the new Web-based user interface, respectively. All messages sent from the GUI Deliverer to the Middleware B or from the COBOL program to the Middleware A are synchronous. It is worth noting that for clarity reason this diagram does not show how the shared memory and semaphores are used to enable the communication and synchronization between the reengineered user interface and the Wrapped Legacy System. Rather, we use asynchronous messages between the two interface objects of the middleware to model such a communication and synchronization.

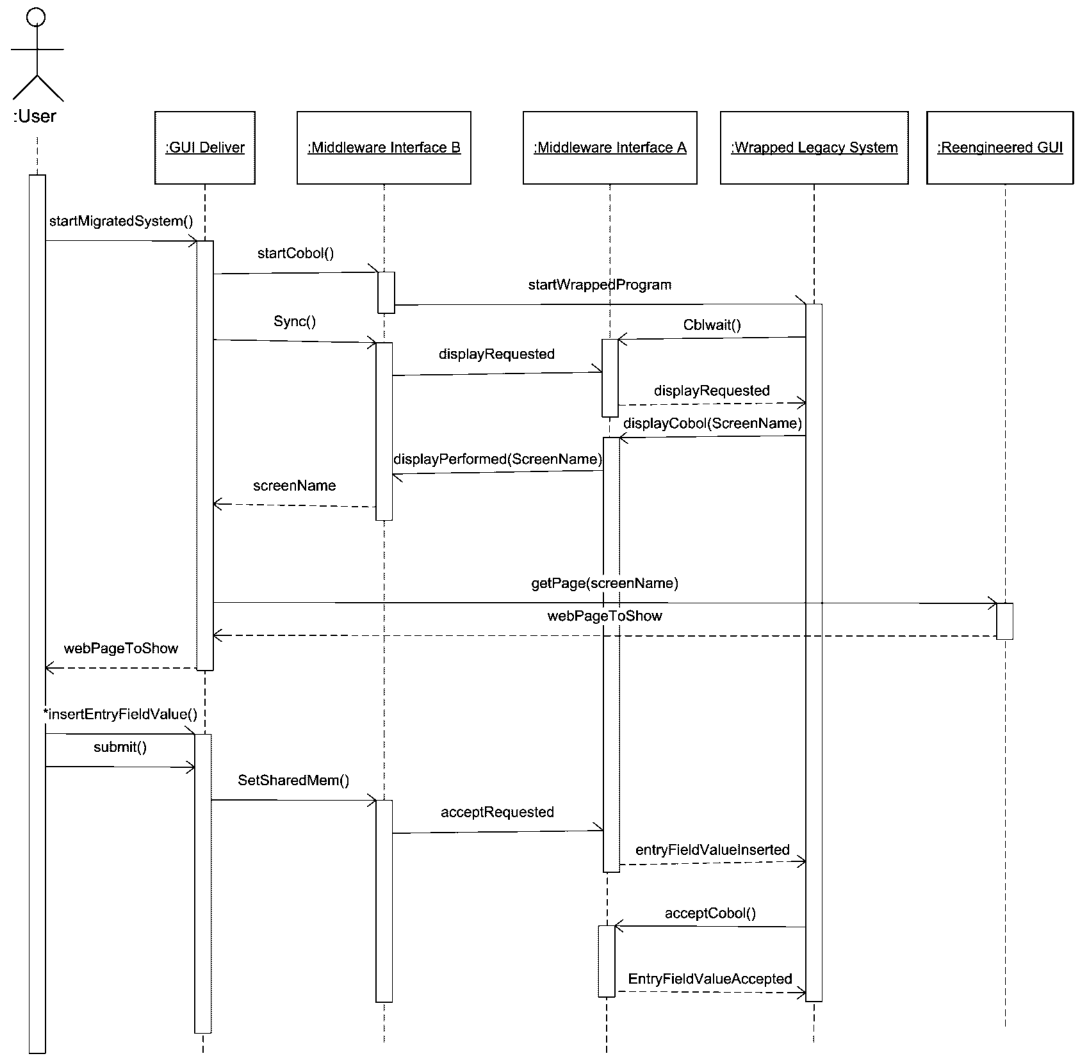


Figure 4. Interactions among the components of the migrated legacy system and the user.

The GUI Deliverer component starts the Wrapped Legacy System through the Middleware Interface B. Once the COBOL program has been started, it calls the Middleware Interface A through the function *Cblwait*, so that it can wait for a display request from the Web user interface. At the same time, the Web user interface calls the *Sync* method asking for the next Web page to show. The control is then returned to the COBOL program, which calls the *displayCobol* function, passing the information concerning the next SCREEN SECTION to display. This information is in turn returned to the GUI Deliver, which gets the Web page corresponding to the SCREEN SECTION and shows it to the user. Once the user has submitted the form, the values of the entry fields are written in the shared memory and the control is returned to the COBOL program, which performs the accept operation by calling the *acceptCobol* function of the middleware that reads the values of the entry fields from the shared memory. At that point, the application logic of the COBOL program processes the information coming from the user interface until next display is performed and the information on the Web page to visualize is returned to the GUI Deliverer.

3.3. Identifying problems and risks

The third step in preparing a migration is to identify potential problems and risks involved. In this case, the migration was complicated by the extensive inclusion of embedded control flows and database accesses in the user interface descriptions within the SCREEN SECTIONs. A COBOL SCREEN SECTION contains the user interface input and output operations similar to javascript, where the program logic can be embedded into the HTML forms. As mentioned before, the COBOL statements for displaying output data on the screen and receiving input data from the screen are the DISPLAY and ACCEPT statements. An ACCEPT statement can trigger ACUCOBOL-GT procedures (COBOL paragraphs similar to methods in Java) both before and after the user has filled in an ENTRY FIELD on the screen. This is performed by specifying the name of the procedures to be invoked in the BEFORE and AFTER clauses of the ACCEPT statement. The BEFORE clause allows a procedure to be performed or the ENTRY FIELD to be initialized before an input value is accepted, whereas an AFTER clause performs a specified procedure after the input value has been accepted. Figure 5 shows an excerpt of an ACUCOBOL-GT program. In particular, this figure shows a chunk of the declaration of the SCREEN SECTION SCREEN-2 and how the clause AFTER of the ENTRY FIELD CST000SN and the clause BEFORE of the ENTRY FIELD C0001SMOLDIV are used. The code of the procedures invoked in these clauses, i.e. CTR-NORDSUD and INIT-CODICI, is shown as well.

Generally, the clauses BEFORE and AFTER increase the difficulty of separating the presentation logic from the application logic and database access logic and make it almost impossible to completely automate the migration process. Unfortunately, this bad practice is not limited to COBOL. It can also be found in modern applications where Java methods and database accesses are built into Web pages.

In case the legacy program to migrate includes BEFORE clauses within a given SCREEN SECTION that trigger ACUCOBOL-GT procedures accessing the system database, these procedures have to be wrapped as they cannot be migrated to the client. In most cases, this requires that these procedures are placed within the legacy program before invoking the function display-Cobol of the Middleware component. On the other hand, when the legacy program contains AFTER clauses within a given SCREEN SECTION triggering procedures accessing the system database, the effect of each procedure has to be properly assessed. For example, in case AFTER clauses trigger procedures that do not affect the values of other entry fields of the SCREEN SECTION, the procedures can be grouped and processed after the SCREEN SECTION has been accepted. For example, in Figure 2 the server-side checks on the entry field values filled in by the user are placed after the accept operation has been wrapped by performing the section LETTURADATI-SCREEN-2, which includes the call to the function CblGetSharedMemory of the Middleware. More complex is the case when the procedures triggered by the AFTER clauses affect the values of other entry fields in the SCREEN SECTION. Dependencies among entry fields require to be analyzed by the software engineer, who might decide to change the interaction

```
.........
01 SCREEN-2, exception esci-2.
.........
02 entry-field using CST000APERI lines 0,5 col + 2  UPPER
                                                  COLOR E-COL-ACC
                                                  AFTER CTR-IMPRESA. .........
01 SCREEN-4, exception esci-4.
  02 entry-field using C0001SMOLDIV(1) PIC Z(5),9(2) lines 0,5
                                                 col + 2
                                                 right
                                                 COLOR E-COL-ACC
                                                 BEFORE INIT-CODICI.
.........
CTR-IMPRESA SECTION.
CO01.
    IF CST000APERI NOT = 'S' AND NOT = 'N'
    MOVE ' VALORE ERRATO ' TO MESSAG1
    perform errore
    SET W-SCR-GOTO TO TRUE
    GO CO09.
    DISPLAY SCREEN-2.
CO09. EXIT.
.........
INIT-CODICI SECTION.
CT1.
    IF C0001ACODCON = SPACES GO TO CT9.
    PERFORM SETTA-FILE-2
    PERFORM OPEN-FILE-2
    MOVE C0001ACODCON TO TBCODAZ-KEY
    READ TBCODAZ INVALID KEY
.........
    PERFORM CLOSE-FILE-2.
EXIT.
```

Figure 5. An excerpt of an ACUCOBOL-GT program.

sequence according to some design goals, while preserving the semantics of the SCREEN SECTION dialogue.

Another potential problem involved in the migration of COBOL legacy concerns the programming of events. Events are used to communicate actions (e.g. a key pressed or a mouse button pressed) taken by the user on graphical objects (i.e. buttons, tabs, etc.). In the wrapping of a legacy program, events have to be properly considered to make the communication between the new user interface and the wrapped legacy code effective. For example, in case the actual user interface contains tabs, these have to be properly simulated in the wrapped legacy code.

4. THE MIGRATION PROCESS

The Middleware in the target architecture shown in Figure 1 is a generic component developed so that it can be used in the migration of any legacy system. The migration effort can then be concentrated on the wrapping of the legacy programs and on the reengineering of the user interface. Figure 6 shows the migration process as an activity diagram enhanced by object flow. The rounded rectangles represent process phases, whereas the rectangles represent the intermediate artifacts produced during the process phases.

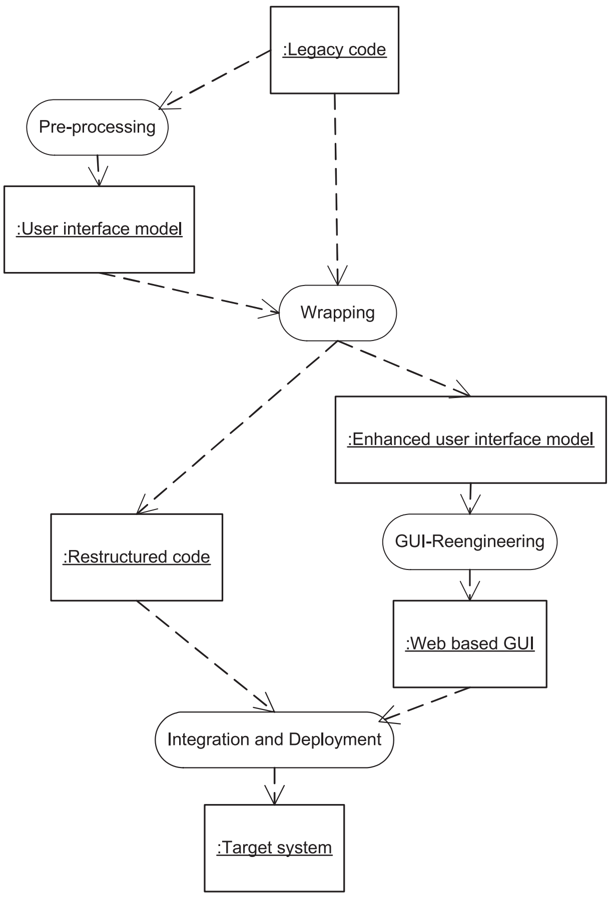


Figure 6. The migration process.

Owing to the embedded control flows and database accesses within the SCREEN SECTION descriptions, the migration process could not be fully automated. There has to be human interaction. Human interaction, however, requires a deep comprehension of the original system, which can be achieved only through dynamic and static analyses. This was the rationale for developing the MELIS tool kit. It was meant to support the software engineer during all phases of the migration process, including the compilation and deployment of the reengineered user interface and of the wrapped legacy programs. In the following subsections, we describe the four phases of the migration process:

- Pre-processing;
- Wrapping;
- GUI-Reengineering;
- Integration and Deployment;

and how they are supported by MELIS.

4.1. Pre-processing

The *Pre-processing* phase provides details on the types and attributes of each graphical object composing a SCREEN SECTION. Information on each SCREEN SECTION is collected, such as its name, its position on the screen, and the number of ENTRY FIELDs and LABELs composing it. As a result of this phase, the BEFORE and AFTER clauses associated with an ENTRY FIELD are classified as follows:

- format checking operations, which can be easily migrated to the client. The code implementing these types of operations is embedded in the XML file produced in the Pre-processing phase as client-side scripting code;
- database accesses or application logic operations, which cannot be migrated to the client and have to be placed in the wrapped program.

MELIS uses an ACUCOBOL-GT parser to transform the structures of the SCREEN SECTIONs of a legacy program into an XML file. This file is hierarchically organized to describe the code associated with each screen. As depicted in Figure 7, the first level shows the names of the SCREEN SECTIONs. The graphical objects of a SCREEN SECTION with their corresponding BEFORE and AFTER clauses are shown at the second and third levels, respectively. These clauses are depicted as leaf nodes of a tree together with the names of the associated control procedures (see left-hand side of Figure 7).

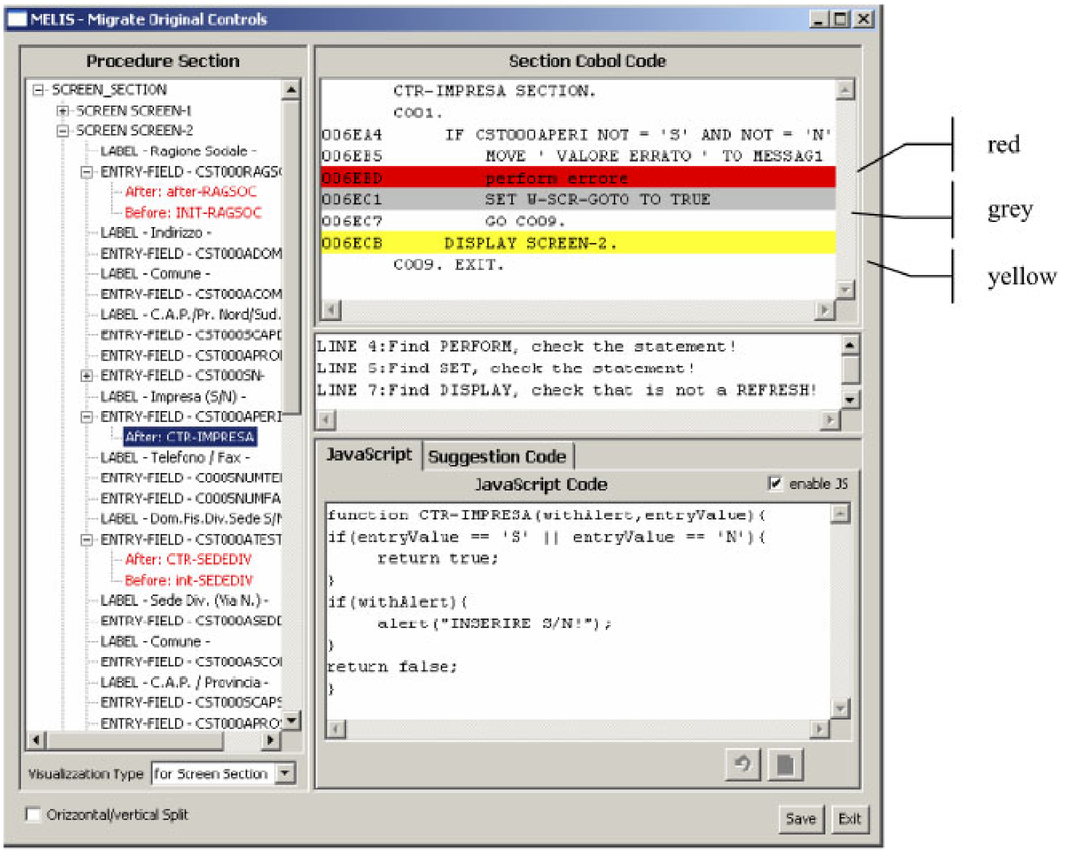
4.2. Wrapping

During the *Wrapping* phase, the interactive programs of a legacy system are transformed into batch programs. To achieve this objective, the software engineer has to transform the embedded procedures into client beans and restructure the COBOL code. In particular, he/she has to decide whether to migrate the BEFORE and AFTER clauses to the client or to keep them in the wrapped legacy code. Information on the BEFORE and AFTER clauses to be migrated to the client is encoded in the user interface model, thus providing an enhanced user interface model.

MELIS automatically classifies the control checks and highlights the corresponding legacy code with different background colors[¶] (see Figure 7). For example, the code highlighted in dark shade contains either PERFORM statements or data access statements such as READ and WRITE. In case the legacy code contains a PERFORM statement, the software engineer has to verify whether or not the invoked code can be migrated to the client side as a javascript function. Conversely, the code highlighted in light shade does not have any interaction with the legacy system, as, for example, the validation of the date format. Therefore, it can be translated directly into javascript to be migrated to the client side.

The sample in Figure 7 shows the code associated with the entry field CST000APERI of the SCREEN SECTION SCREEN-2. The AFTER clause associated with that field verifies whether the field contains the character ‘S’ or ‘N’. Since interactions with the database are not required, this check can be migrated to the client. MELIS also proposes some default scripts that can be reused, as shown in the *Suggestion Code* window on the bottom right-hand side of Figure 7. The set of default

[¶]For black and white printing, the colors are reported on the right-hand side of the screen shot.



```
SCREEN_SECTION
    SCREEN SCREEN-1
    SCREEN SCREEN-2
        LABEL - Ragione Sociale -
        ENTRY-FIELD - CST000RAGS
            After: after-RAGSOC
            Before: INIT-RAGSOC
        LABEL - Indirizzo -
        ENTRY-FIELD - CST000ADOM
        LABEL - Comune -
        ENTRY-FIELD - CST000ACOM
        LABEL - C.A.P./Pr. Nord/Sud.
        ENTRY-FIELD - CST0005CAPE
        ENTRY-FIELD - CST000APROI
        ENTRY-FIELD - CST000SN-
        LABEL - Impresa (S/N) -
        ENTRY-FIELD - CST000APERI
            After: CTR-IMPRESA
        LABEL - Telefono / Fax -
        ENTRY-FIELD - C0005NUMTEI
        ENTRY-FIELD - C0005NUMFA
        LABEL - Dom.Fis.Div.Sede S/N
        ENTRY-FIELD - CST000ATEST
            After: CTR-SEDEDIV
            Before: init-SEDEDIV
        LABEL - Sede Div. (Via N.) -
        ENTRY-FIELD - CST000ASEDE
        LABEL - Comune -
        ENTRY-FIELD - CST000ASCOI
        LABEL - C.A.P. / Provincia -
        ENTRY-FIELD - CST0005CAPS
        ENTRY-FIELD - CST000APRO
```

```
          CTR-IMPRESA SECTION.
          C001.
006EA4        IF CST000APERI NOT = 'S' AND NOT = 'N'
006EB5              MOVE ' VALORE ERRATO ' TO MESSAG1
006EBD              perform errore
006EC1              SET W-SCR-GOTO TO TRUE
006EC7              GO C009.
006ECB        DISPLAY SCREEN-2.
          C009. EXIT.
```

```
LINE 4:Find PERFORM, check the statement!
LINE 5:Find SET, check the statement!
LINE 7:Find DISPLAY, check that is not a REFRESH!
```

```
function CTR-IMPRESA(withAlert,entryValue){
if(entryValue == 'S' || entryValue == 'N'){
     return true;
}
if(withAlert){
     alert("INSERIRE S/N!");
}
return false;
}
```

Figure 7. Migrating controls on the client side.

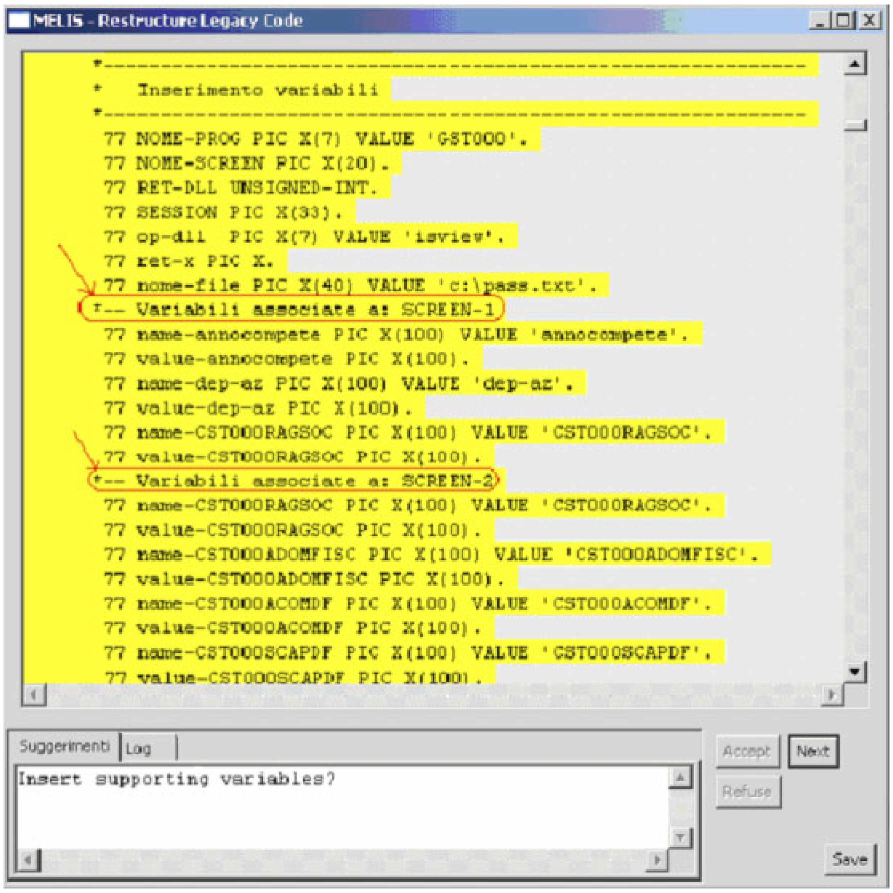
scripts can be modified introducing new javascript functions or modifying the existing ones. The javascript code associated with a BEFORE/AFTER clause is embedded in the XML file produced in the Pre-processing phase. An excerpt of the XML file corresponding to the AFTER clause of the entry field CST000APERI of the SCREEN SECTION SCREEN-2 is shown in Figure 8.

The check operations using the database or the application logic of the original COBOL program cannot be migrated to the client, and the code performing these checks has to be kept in the Wrapped Legacy System. The COBOL code is wrapped in three steps. In the first step, the WORKING STORAGE SECTION is enhanced with the declaration of variables to be associated with the graphical objects detected during the Pre-processing phase. For each graphical object, two new variables are introduced giving the name of the graphical object and its value provided as an input to the Wrapped Legacy System or as an output of the Middleware component. The declaration of these variables is automatically included in the wrapped legacy code by MELIS (see Figure 9).

In the second step, a set of instructions for enabling the legacy system initialization and synchronization with the Web application is inserted. For example, Figure 10 shows the COBOL code that initializes the communication between the Wrapped Legacy System and the Middleware

```
<LABEL> Impresa (S/N) <visualizza>7</visualizza>
<SIZE>14</SIZE>
<COLONNA>52</COLONNA>
</LABEL>
<ENTRY-FIELD> CST000APERI<visualizza>7</visualizza>
<COLONNA>+2</COLONNA>
<ALIGN>upper</ALIGN>
<After controllo="presente">CTR-IMPRESA<CodeJS><![CDATA[function
CTRIMPRESA(withAlert,entryValue){
        if(entryValue == 'S' || entryValue == 'N'){
                return true;
        }
        if(withAlert){
                alert("INSERIRE S/N!");
        }
        return false;
}]]></CodeJS>
</After>
```

Figure 8. The javascript code associated with an AFTER clause.



```
      *-------------------------------------------------------------
      *   Inserimento variabili
      *-------------------------------------------------------------
       77 NOME-PROG PIC X(7) VALUE 'GST000'.
       77 NOME-SCREEN PIC X(20).
       77 RET-DLL UNSIGNED-INT.
       77 SESSION PIC X(33).
       77 op-dll  PIC X(7) VALUE 'isview'.
       77 ret-x PIC X.
       77 nome-file PIC X(40) VALUE 'c:\pass.txt'.
      *-- Variabili associate a: SCREEN-1
       77 name-annocompete PIC X(100) VALUE 'annocompete'.
       77 value-annocompete PIC X(100).
       77 name-dep-az PIC X(100) VALUE 'dep-az'.
       77 value-dep-az PIC X(100).
       77 name-CST000RAGSOC PIC X(100) VALUE 'CST000RAGSOC'.
       77 value-CST000RAGSOC PIC X(100).
      *-- Variabili associate a: SCREEN-2
       77 name-CST000RAGSOC PIC X(100) VALUE 'CST000RAGSOC'.
       77 value-CST000RAGSOC PIC X(100).
       77 name-CST000ADOMFISC PIC X(100) VALUE 'CST000ADOMFISC'.
       77 value-CST000ADOMFISC PIC X(100).
       77 name-CST000ACOMDF PIC X(100) VALUE 'CST000ACOMDF'.
       77 value-CST000ACOMDF PIC X(100).
       77 name-CST000SCAPDF PIC X(100) VALUE 'CST000SCAPDF'.
       77 value-CST000SCAPDF PIC X(100).
```

```
Insert supporting variables?
```

Figure 9. Automatic declaration of supporting parameters.

```
set environment 'DLL_CONVENTION' to "1".
call 'C:\WRAPPER.dll'.
call 'Cblwait' using by content session.
cancel 'C:\WRAPPER.dll'.
set environment 'DLL_CONVENTION' to "0".
```

Figure 10. COBOL code used to start the execution of the wrapped legacy code.

```
set environment 'DLL_CONVENTION' to "1".
call 'C:\WRAPPER.dll'.
call 'Cblexit' using by content SESSION
cancel 'C:\WRAPPER.dll'.
set environment 'DLL_CONVENTION' to "0".
```

Figure 11. COBOL code used to stop the execution of the wrapped legacy code.

component. Let us note that MELIS automatically inserts these statements analyzing the data flow of the original legacy system. On the other hand, the COBOL code depicted in Figure 11 is used to interrupt the communication between the wrapped legacy code and the Middleware component and it is invoked when the migrated legacy system is closed. Also this COBOL code fragment is automatically inserted by MELIS.

In the third step, all of the DISPLAY and ACCEPT statements are commented out and replaced by calls to COBOL sections, thereby encapsulating the communication with the Middleware. MELIS automatically creates two COBOL sections and supports the software engineer in the replacement of the DISPLAY and ACCEPT statements with PERFORM statements to these two sections. Figure 2 shows an excerpt of wrapped legacy code, where ACCEPT and DISPLAY statements of the SCREEN SECTION SCREEN-2 (whose structure is shown in Figure 7) are commented and replaced with calls to the Middleware component. The excerpt of the COBOL code also shows the sections PASSAGGIODATI-SCREEN-2 and LETTURADATI-SCREEN-2, which are used to write and read data in the shared memory, respectively. In particular, the section PASSAGGIODATI-SCREEN-2 is used to pass the data from the wrapped legacy code to the new user interface. The data will be then formatted and properly visualized in the reengineered Web-based user interface. On the other hand, the wrapped legacy program uses the section LETTURADATI-SCREEN-2 to obtain the values that the user has filled in the entry fields of the SCREEN SECTION SCREEN-2. These values are then used to execute the application logic of the original legacy system.

Since server-side checks generally have to be handled in the code replacing the DISPLAY and ACCEPT statements with calls to the Middleware, the interaction between the new Web user interface and the wrapped system could be intensive. By default, MELIS put all server-side checks corresponding to BEFORE clauses before the wrapped display and all server-side checks corresponding to AFTER clauses after wrapped accept (see Figure 2), as this choice minimizes the interactions between the Web and the legacy parts of the migrated application. The software engineer can make different decisions about the AFTER clauses and change the code of the Web user interface and of the wrapped program in case the triggered procedures affect the values of other entry fields in the SCREEN SECTION, as discussed in Section 3.3. For example, the software engineer can preserve the sequence of the checks, or he/she can try to improve the performances by minimizing the interactions between the Web and the legacy components, while preserving the semantics of the SCREEN SECTION dialogue. In all cases, the error handling is delegated to the Wrapped Legacy System.

4.3. GUI-Reengineering

In the *GUI-Reengineering* phase, MELIS automatically generates all the components of the target Web application according to the MVC architectural pattern. In particular, MELIS generates a Web-based subsystem for each SCREEN SECTION contained in an ACUCOBOL-GT program.

These subsystems are composed of the following four components:

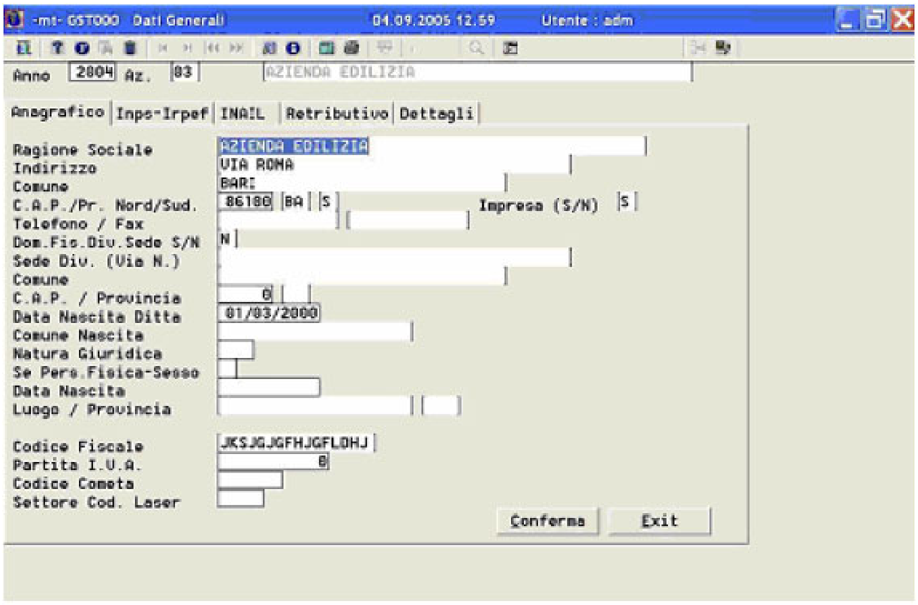
- a JSP page, representing the Reengineered GUI of the original system;
- a javabean, having as fields all the entry fields of the SCREEN SECTIONS. It is generated from the XML file created earlier when processing the SCREEN sections. It stores the data provided by the Wrapped Legacy System;
- two servlets, one dedicated to the Middleware synchronization and the other to the data exchange with the Wrapped Legacy System component. The first servlet instantiates the bean containing the information coming from the Middleware and redirects the control to the reengineered user interface displaying the information. The second servlet receives the input data used to fill in the bean, which in turn is passed to the Middleware when an input operation is required.

MELIS also generates two additional servlets for the system to be migrated. The first servlet enables the GUI Deliverer software component to start the Wrapped Legacy System, whereas the second servlet is responsible for the selection of the reengineered user interface component to display according to the legacy system request.

The appropriate look and feel obtained by adopting cascading style sheets (CSS) allows the developers to easily modify the reengineered user interface. Figures 12 and 13 depict a typical legacy user interface and the corresponding reengineered Web user interface. In particular, Figure 13 shows the JSP page within the browser integrated in MELIS together with the corresponding scripting code. It is worth noting that the look and feel of the Web-based interface was very similar to the original one. Indeed, the partner company imposed such a constraint to avoid forcing the end-users to change their working habits. On the other hand, new user requirements on the Reengineered GUI could be easily implemented by simply editing the associated CSS file.

In case the software engineer does not change the sequence of server-side checks defined by default by MELIS, the user interface is automatically generated. Otherwise, the software engineer has to make some changes that are compatible with the sequence of server-side checks.

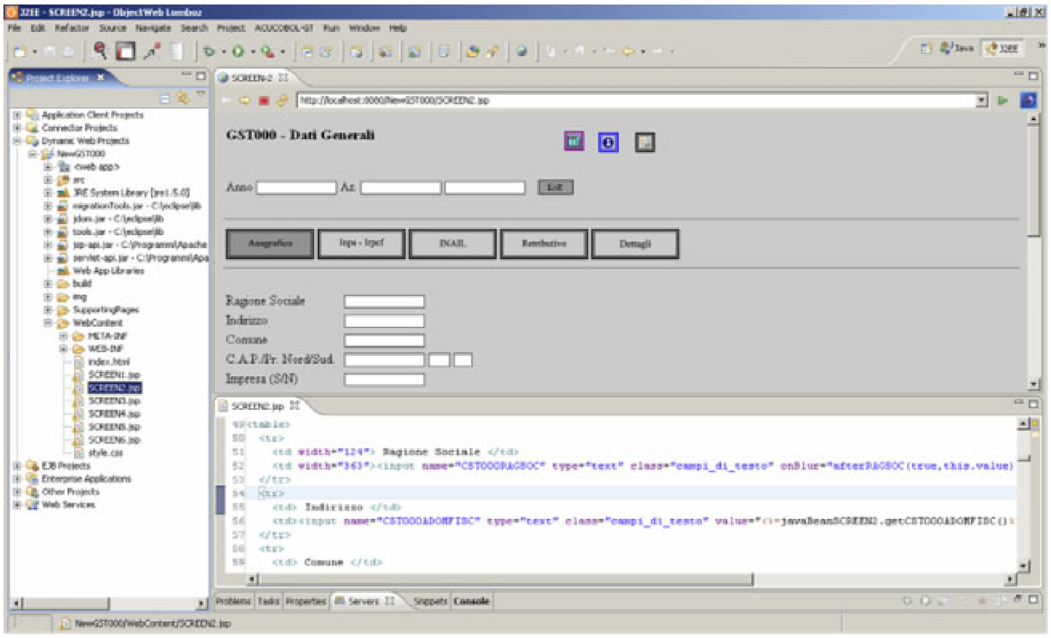
In the graphical user interface of the original legacy system, it could happen that some buttons are not active. Thus, the software engineer could decide of not including this button in the reengineered graphical user interface. For example, some buttons in Figure 12 are no longer needed; hence, they were not migrated. Furthermore, two kinds of buttons were identified in the analyzed legacy systems, native system calls and ACUCOBOL-GT program performing features, which are not dependent on the application logic of the running program. For example, the button on the right-hand side of the bar in Figure 13 invokes the native calculator of the client system, whereas the other button invokes a Web page presenting features of the original system.



Anno 2004 Az. 83 AZIENDA EDILIZIA

Ragione Sociale	AZIENDA EDILIZIA
Indirizzo	VIA ROMA
Comune	BARI
C.A.P./Pr. Nord/Sud.	86100 BA S
Impresa (S/N)	S
Telefono / Fax	
Dom.Fis.Div.Sede S/N	N
Sede Div. (Via N.)	
Comune	
C.A.P. / Provincia	0
Data Nascita Ditta	01/03/2000
Comune Nascita	
Natura Giuridica	
Se Pers.Fisica-Sesso	
Data Nascita	
Luogo / Provincia	
Codice Fiscale	JKSJGJGFHJGFLDHJ
Partita I.V.A.	0
Codice Cometa	
Settore Cod. Laser	

Figure 12. Legacy user interface.


J2EE - SCREEN2.jsp - ObjectWeb Lomboz
File Edit Refactor Source Navigate Search Project ACUCOBOL-GT Run Window Help
Project Explorer
Application Client Projects
Connector Projects
Dynamic Web Projects
NewGST000
<web app>
src
JRE System Library [jre1.5.0]
migrationTools.jar - C:\eclipse\lib
jdom.jar - C:\eclipse\lib
tools.jar - C:\eclipse\lib
jsp-api.jar - C:\Programmi\Apache
servlet-api.jar - C:\Programmi\Apa
Web App Libraries
build
img
SupportingPages
WebContent
META-INF
WEB-INF
index.html
SCREEN1.jsp
SCREEN2.jsp
SCREEN3.jsp
SCREEN4.jsp
SCREEN5.jsp
SCREEN6.jsp
style.css
EJB Projects
Enterprise Applications
Other Projects
Web Services
SCREEN-2
http://localhost:8080/NewGST000/SCREEN2.jsp
Exit
Anagrafico
Inps - Irpef
INAIL
Retributivo
Dettagli
SCREEN2.jsp
Problems Tasks Properties Servers Snippets Console
NewGST000/WebContent/SCREEN2.jsp

GST000 - Dati Generali

Anno Az.

Ragione Sociale	
Indirizzo	
Comune	
C.A.P./Pr. Nord/Sud	
Impresa (S/N)	

```
<table>
  <tr>
    <td width="124"> Ragione Sociale </td>
    <td width="363"><input name="CST000RAGSOC" type="text" class="campi_di_testo" onBlur="afterRAGSOC(true,this.value)
  </tr>
  <tr>
    <td> Indirizzo </td>
    <td><input name="CST000ADOMFISC" type="text" class="campi_di_testo" value="<%=javaBeanSCREEN2.getCST000ADOMFISC()%
  </tr>
  <tr>
    <td> Comune </td>
```

Figure 13. Reengineered user interface.

4.4. Integration and Deployment

Finally, in the *Integration and Deployment* phase, the Wrapped Legacy System and the Web-based user interface are integrated to produce the target system. During this phase, the code of the Wrapped Legacy System is re-compiled using the ACUCOBOL-GT compiler included in MELIS. The reengineered Web-based graphical user interface components (i.e. the generated beans, servlets, and dynamic pages) are deployed on the *Server* node, whereas the compiled code of the wrapped legacy program is deployed on the *Service* node. MELIS also allows Tomcat to be started inside Eclipse by using the Eclipse Web Tool Platform. In the Integration and Deployment phase, the software engineer should also perform regression testing to prove the functional equivalence between the original legacy system and the migrated system. Test cases can be derived from the legacy system and used to exercise the target system in order to identify possible differences in the behavior. Note that MELIS does not provide specific support to automatically derive test cases and prove the functional equivalence. Finally, when all the process phases are accomplished, the migrated application is enabled to run.

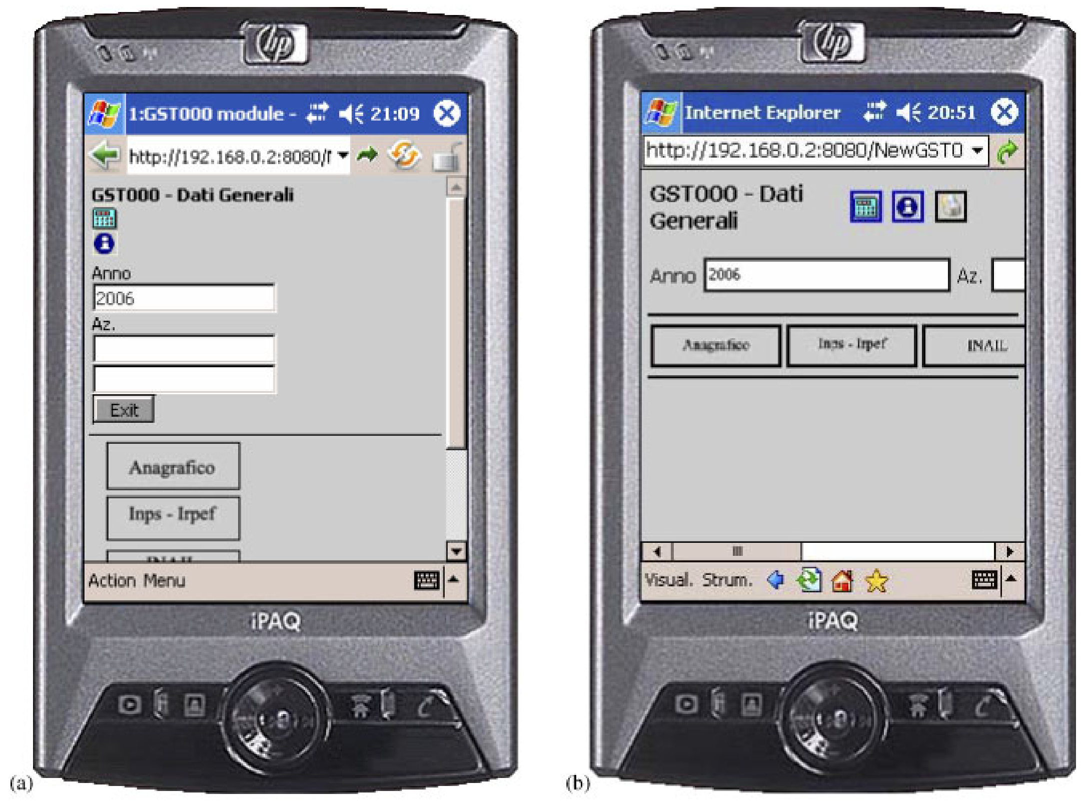


Figure 14. GST000 Web executed on mobile device with two different browsers.

Once the legacy application has been migrated, it can be used through any Web Browser, including browsers on mobile devices. In the latter case, the reengineered user interface appears differently depending on the browser. For example, Figure 14 shows the migrated interface within the Web Browsers Opera Mobile (Figure 14(a)) and Internet Explorer Mobile (Figure 14(b)) on the Palmtop iPAQ RX3115 equipped by 300 MHz processor with 64 Mb of RAM and Windows Mobile 2003 Second Edition as operating system. In particular, Opera Mobile attempts to better dispose the graphical objects, whereas Internet Explorer Mobile preserves the original layout.

5. EVALUATION

To evaluate the migration process and tool as well as to start transferring the migration technology to the practitioners of the partner company, we initialized a pilot migration project on the most business critical system of the company. This system was used by many key customers to support their payroll, tax, and social security management, as well as to regulate their employee relationships. This also happened to be the oldest legacy system of the partner company, having been developed, evolved, and marketed over the last 30 years. The evolution of this system was driven by all of the business and technology changes occurring during this time period. The first release dated 1978 and the system was finally migrated to ACUCOBOL-GT in 2002. It is a multi-user system with a two-tier client–server software architecture in which both the presentation and application layers are running on the client and the centralized database consisting of virtual storage access method (VSAM) files is situated on the server. The synchronization of concurrent accesses to VSAM files is handled by the ACUCOBOL-GT runtime environment.

Table III shows some descriptive statistics of the legacy system. There is a large number of *READ* and *WRITE* statements, which are used to access the data layer. The *READ* and *WRITE* statements are uniformly distributed among the programs. The static analysis also revealed an abnormally large number of GOTO statements spread across all of the programs in the system that provides an idea of the spaghetti-like structure of the consider legacy system: GOTOs are more than 5% of the total LOCs (not only the PROCEDURE DIVISION LOCs). The large number of GOTOs contributed

Table III. Statistics of the analyzed COBOL system.

Number of files	2875
Number of programs	569
Total LOCs	613 081
Total number of *CALL*	5355
Total number of *READ*	4755
Total number of *WRITE*	5082
Total number of *RE-WRITE*	635
Total number of GOTO	32 544
Number of programs with *SCREEN SECTION*s	376
Number of *SCREEN SECTION*s	474
Total number of *DISPLAY*	4991
Total number of *ACCEPT*	1272
Total number of *BEFORE*	3416
Total number of *AFTER*	6487
Total number of global variables	110

to the difficulty in comprehending the control flow structure and made the programs difficult to change. In a real reengineering project, it would have been necessary to remove the GOTOs and to restructure the code. The objective of this project was, however, not to increase the quality of the system but to migrate it to the Web. This is where migration projects differ from reengineering projects [26].

We also observed that 66% of the programs contained SCREEN SECTIONs and that 4.7% of LOCs in these programs were in the SCREEN SECTIONs. The DISPLAY and ACCEPT operations in these SCREEN SECTIONs were interleaved with the business rules and the database accesses. This fact combined with the GOTO-driven control flow made it very difficult to decompose the system. A code inspection also confirmed that the system was not decomposable into software layers. On the positive side, the system was divided up into independent subsystems, each implementing a different business function, and the communication among them was limited to common global variables and file sharing (see Table III). The large number of BEFORE and AFTER statements indicated that the interactions between the graphical user interface and the application logic and database component were very intensive. This was due to the continuous checking and updating of fields in the graphical user interface. These concerns made the migration of the legacy system complex and time consuming.

5.1. The pilot project

Our tool supported migration process was tested on two complex and essential subsystems of the total system. The migration project group was composed of two teams, each composed of an academic researcher and a practitioner from the partner company. Each team was responsible for migrating one of the two subsystems. The selected subsystems—one for supplier management and another for user registry—contained most of the problems that could be encountered during the migration of the other subsystems of the partner company. The COBOL source files of these two subsystems were named GST000.cbl and GST001.cbl, respectively; thus, in the following we will refer to them as GST000 and GST001.

The graphical objects in the user interfaces of GST000 and GST001 are shown in Tables IV and V, respectively. These tables also include information on the distribution of the BEFORE and AFTER operations. GST000 contains 26 BEFORE and 41 AFTER clauses, whereas GST001 contains 111 BEFORE and 112 AFTER clauses, respectively. Of course, the larger the number of BEFORE and AFTER statements, the more complex and time consuming the migration process is.

To better understand the complexity of the two pilot subsystems, some other measures provided by MELIS are shown in Table VI. Both GST000 and GST001 contain calls to subprograms. However,

Table IV. Screen sections of GST000.

SCREEN SECTION	Label	Entry-field	Push-button	BEFORE	AFTER
Screen-1	2	3	0	2	2
Screen-2	21	25	2	3	10
Screen-3	28	27	2	5	9
Screen-4	20	23	2	1	3
Screen-5	38	32	2	9	11
Screen-6	46	43	2	6	6

Table V. Screen section of GST001.

SCREEN SECTION	Label	Entry-field	Push-button	BEFORE	AFTER
Screen-1	1	3	0	1	1
Screen-2	31	38	2	20	21
Screen-3	38	54	2	37	37
Screen-4	33	32	2	4	4
Screen-5	32	31	2	0	0
Screen-6	29	29	2	0	0
Screen-7	38	41	2	18	18
Screen-8	39	41	2	28	28
Screen-9	17	19	2	1	1
Screen-10	13	9	2	2	2

Table VI. GST000 and GST001 statistics.

	GST000	GST001
LOC	6691	13 779
READ	12	27
WRITE	4	3
REWRITE	4	5
GOTO	89	223
CALL	127	498
PERFORM	81	251

Table VII. Some statistics of the migrated GST000 and GST001 subsystems.

	GST000	GST001
LOC	8268	18 789
LOC added by the tool	1447	4601
LOC manually added	130	389

the only interactive programs are the two main programs, whereas all the subprograms are batch programs (i.e. they contain no SCREEN SECTION).

Some descriptive statistics of the migrated subsystems are presented in Table VII. The first row reports the LOCs in the original subsystem, whereas the LOCs added by the tool and the developers are shown in the second and third rows, respectively. The manual coding was required because some of the procedures triggered by AFTER clauses affected the values of other entry fields in the SCREEN SECTIONs. Manual coding was also required to handle the tab pushed events of the graphical user interface of the original ACUCOBOL-GT subsystems, as MELIS currently does not provide an automatic support for events.

5.2. Discussion

The difference between the automatically added and manually coded LOCs reveals that the migration environment effectively supported the applied migration strategy. For example, the migration team

Table VIII. BEFORE/AFTER migration for GST000.

	BEFORE		AFTER	
SCREEN SECTION	Client	Server	Client	Server
Screen-1	0	2	0	2
Screen-2	0	3	8	2
Screen-3	0	5	6	3
Screen-4	0	1	1	2
Screen-5	0	9	6	5
Screen-6	0	6	6	0

Table IX. BEFORE and AFTER clauses migration for GST001.

	BEFORE		AFTER	
SCREEN SECTION	Client	Server	Client	Server
Screen-1	0	1	0	1
Screen-2	0	20	20	1
Screen-3	0	37	27	10
Screen-4	0	4	0	4
Screen-5	0	0	0	0
Screen-6	0	0	0	0
Screen-7	0	18	5	13
Screen-8	0	28	21	7
Screen-9	0	1	0	1
Screen-10	0	2	0	2

manually coded only 8.2% of the LOCs required to wrap GST000, whereas for GST001 the percentage of manually coded LOCs was only 7.7%. This low percentage of manual work indicates that MELIS effectively supported the wrapping of the legacy code, by reducing the number of LOCs that the developers had to manually code.

Tables VIII and IX denote the numbers of BEFORE and AFTER clauses whose code has been migrated to the Web-based user interface (client) or kept in the wrapped legacy programs (server) for GST000 and GST001. It can also be seen that the code associated with the BEFORE clauses was never migrated to the client, since the BEFORE clauses were used primarily to query the database.

The effort required to migrate the two subsystems (expressed in terms of person/h) as well as the distribution of effort among the different phases of the migration process is shown in Table X. Let us note that we considered together the efforts to perform the comprehension of the programs and the Pre-processing phase. This is due to the fact that the Pre-processing phase requires a little effort, as it is automatically performed by MELIS. As expected, most of the migration effort was required to comprehend the legacy system and to perform the Wrapping phase.

Some preliminary performance measurements were made to compare the migrated and the original legacy subsystems. For this, we used Apache (version 2.2.0) as a Web Server and Tomcat (version 5.5.16) as a Web container. Moreover, a PC equipped by a 3.6 GHz Intel Pentium IV with 1.5 GB of RAM, a 100 GB Hard Disk and Windows XP Professional SP 2 as operating system was

Table X. Migration efforts for GST000 and GST001 (person/h).

Migration phase	GST000	GST001
Comprehension and Pre-processing	19	26
Wrapping	10	16
GUI-Reengineering	5	9
Integration and Deployment	5	7
Total effort	39	58

Table XI. Times to display original SCREEN SECTIONs and new Web user interfaces of GST000.

	Original legacy subsystem		Migrated subsystem	
SCREEN SECTION	Display	Accept	Display	Accept
Screen-1	0.07	0.05	0.04	0.06
Screen-2	0.40	0.30	0.02	0.10
Screen-3	0.45	0.30	0.01	0.08
Screen-4	0.42	0.30	0.03	0.10
Screen-5	0.40	0.30	0.05	0.11
Screen-6	0.47	0.30	0.05	0.16
Mean time	0.31	0.22	0.02	0.08

Table XII. Times to display original SCREEN SECTIONs and new Web user interfaces of GST001.

	Original legacy subsystem		Migrated subsystem	
SCREEN SECTION	Display	Accept	Display	Accept
Screen-1	0.11	0.07	0.03	0.07
Screen-2	0.49	0.36	0.02	0.11
Screen-3	0.56	0.36	0.03	0.09
Screen-4	0.47	0.36	0.04	0.11
Screen-5	0.44	0.36	0.03	0.09
Screen-6	0.51	0.36	0.04	0.11
Screen-7	0.57	0.36	0.04	0.13
Screen-8	0.55	0.36	0.04	0.13
Screen-9	0.36	0.36	0.02	0.08
Screen-10	0.35	0.36	0.02	0.08
Mean time	0.40	0.30	0.02	0.09

used as a server, whereas the client machines were PCs connected using a LAN. Tables XI and XII show the times expressed in seconds required to display the original SCREEN SECTIONs and the corresponding Web-based user interfaces of GST000 and GST001. These tables also indicate the time to accept data from the original SCREEN SECTIONs and that required for the corresponding Web-based user interfaces. The reported times include the time to perform the field checks and the database accesses as well as the execution of the application logic for the original and the migrated subsystems.

Copyright © 2008 John Wiley & Sons, Ltd.

It is noteworthy that the visualization of the reengineered SCREEN SECTIONs required less time than the original user interface. In particular, we observed that each time a user passes from one SCREEN SECTION to another, the data of the SCREEN SECTIONs are repetitively reloaded by accessing the database. The difference between the input/output operations of the original subsystem is mainly due to the time required to perform read/write operations on the VSAM files. Since VSAM files are not indexed, the time required to write a record is less than the time required to read a record from the same file. We also observed that the time to accept input fields, to check them, and update the database is the same for all SCREEN SECTIONs except the *Screen-1* of both migrated subsystems. This is because once an accept operation is invoked within a given SCREEN SECTION, the legacy code uses a COBOL program to update the fields of each SCREEN SECTION. The significant difference between the times to display the original and the migrated user interfaces is a result of the graphical engine of the ACUCOBOL-GT virtual machine. Indeed, the graphical engine requires more time than the Web application to display the user interface.

This can be regarded only as a preliminary assessment of the performances of the migrated system. As the legacy system is multi-user, we need to compare the performances of the new and old versions of the system in a multi-user scenario, where different users concurrently access the system. Indeed, although the legacy system is based on a fat-client architecture, where both the presentation and the application logics are executed on the client, the migrated version is based on a thin-client Web architecture, where the application logic and also part of the presentation are executed on the server. Therefore, such an experimental assessment would be useful to identify the characteristics that the server machine should have to match the performance of the migrated version of the system with the performance of the original one.

However, we wish to point out that accesses to the centralized database embedded in the user interface of the legacy system also represent a bottleneck for the performance of the legacy version of the user interface; these accesses are generally optimized during the Wrapping phase of the migration process.

Finally, even if the new architecture were to result in worse performances than the legacy one, an advantage induced by the migration to the Web is the reduction in the management costs, as the system functionality is accessed through a Web Browser independent of the hardware/software platform of the user and without the need of re-installing the client application as a consequence of software evolution.

6. CONCLUSION AND LESSON LEARNED

This paper has presented a strategy for the migration of legacy information systems implemented in ACUCOBOL-GT to a Web-based multi-tier architecture using migration environment for legacy information systems (MELIS). MELIS is an integrated migration environment developed as an Eclipse plug-in to support all different phases of the migration process.

Both the migration process and the tool were developed and tested within a technology transfer project funded by MTSys s.r.1., a small Italian software company. This pilot project was intended to discover the most suitable method for migrating the ACUCOBOL-GT legacy software developed by the partner company to the Internet. We defined a migration strategy aimed at reengineering the user interface using Web technologies and at wrapping the presentation layer of the legacy systems. A middleware component was also developed to enable the communication of the new user interface with the Wrapped Legacy System. It is worth noting that the strategy is general, while its implementation is for legacy systems implemented using the ACUCOBOL-GT programming language. Extending MELIS to other COBOL dialects designed for PCs with Windows operating systems is not a difficult task, as this does not require major changes in the target infrastructure and in the DLL middleware component. On the other hand, adapting MELIS to other hardware/software platforms might be more complex.

Transferring innovative technologies and tools to practitioners has long been investigated in the area of software technology [43–46]. In particular, Pfleeger and Menzes [43] suggest the need for building a technology transfer model to gain a better understanding of how technology adoption works in individual organizations. Usually, such a model entails an evaluation performed by intended users of the technology to see whether it solves their problem. For this reason, we tested the proposed migration strategy and the tool MELIS in a pilot migration project to migrate two critical subsystems of the industrial partner. Each subsystem was migrated by a team composed of an academic researcher expert of Web technologies and a practitioner expert of the legacy environment. The need for building such mixed teams was due to the lack of Web and migration knowledge on the part of the industrial partner and due to the lack of business knowledge on the part of the academic partner. The motivation of the industrial partner for moving to the Web was less a desire to be technically innovative than the need to accommodate the changing business requirements of its customers. It is mostly the end users who wish to introduce new technologies. The software houses themselves are more conservative. They act only when they are forced to [46].

Such a position on the part of the partner company makes the technology transfer quite challenging. It requires a two-phase transfer of the migration technologies. In the first phase, we are doing training on the job by selecting other pilot projects where the most experienced practitioners of the partner company will be involved. We are also performing controlled experiments with practitioners with different levels of experience to evaluate the advantages of using the migration tool with respect to traditional software development tools (see [47] for details). In addition to the pilot projects, these experiments have the goal of introducing the migration technology in the partner company and getting feedback that might help to improve the migration process and tool.

Although in the first phase of the technological transfer project the company practitioners participated only as pilot subjects of the migration experiment, the final goal is to make the partner company an owner of the MELIS tool and the accompanying methods. For this reason, in the second phase, it is planned to train key practitioners in the use of the technologies used to develop MELIS and to involve them in sub-projects for evolving MELIS. Examples of sub-projects, whose need emerged from the pilot project and the controlled experiments, are as follows:

(i) improving the support for the comprehension of the legacy system and
(ii) improving the usability of MELIS, by adding functionality to support bug detection and fixing.

In accordance with the final goal of the technology transfer project, we have considered two important issues in the design of MELIS and of the target infrastructure of the migration process, as discussed in Section 3. The first issue is to take into account the technologies the practitioners are already familiar with, such as the selection of the hardware/software platform for the target architecture and the development of the communication middleware. The second issue is the design of an extensible architecture for MELIS based on the Eclipse framework. This will make the partner company able to address new problems that might emerge during the migration of their legacy systems. Some of these problems have already been identified during the assessment of the legacy systems, such as the opportunity of adopting typical load balancing techniques to distribute the load among different computational nodes and to improve the concurrency, scalability, and multi-access of the legacy systems migrated with MELIS. Furthermore, the assessment of the legacy system used in the pilot migration project showed that one of the bottleneck of the legacy systems is the database, implemented using VSAM files. Thus, it is likely that the industrial partner will soon plan a data migration to relational databases as well as the separation of the data access code from the application logic code.

The research value of this experimental migration project is mainly the experience gained in dealing with a real-life situation, in which the prerequisites for an ideal software migration are seldom present and in which there are many constraints. Changing a running system is always associated with risks. Therefore, the owner of that system wishes to change as least as possible in order to reach the primary goal. The primary goal here was to obtain the system to run on the Web. All other desirable goals, such as restructuring the programs and migrating the database, had to be sacrificed to achieve the main goal within the time and budget constraints imposed. It is the fault of researchers that they feel tempted to change everything necessary to create the perfect system. We managed to avoid that temptation in this project and restricted ourselves to the essential.

ACKNOWLEDGEMENTS

The authors would like to thank Vincenzo Venezia, Giovanni Vildacci, as well as the programmers of MTSys s.r.1. for the stimulating discussion and the precious suggestions. Special thanks are due to Marianna Borriello, Massimo Colosimo, Vincenzo Coppola, Rosario Di Leva, Aniello Napolitano, and Nicola Vitiello, who implemented different components of MELIS and the target architecture of the migration process. The authors are also grateful to the anonymous reviewers for their insights and feedbacks to several key issues covered in this work.

The work described in this paper is supported by the project METAMORPHOS (MEthods and Tools for migrAting software systeMs towards Web and service Oriented aRchitectures: exPerimental evaluation, usability, and technology tranSfer), funded by MiUR (Ministero dell'Università e della Ricerca) under grant PRIN-2006-2006098097. The work described in this paper is also supported by the project TOCAI (Tecnologie Orientate alla Conoscenza per Aggregazioni di Impresa in internet) under grant FIRB 2005.

REFERENCES

1. Pigoski TM. *Practical Software Maintenance—Best Practices for Managing Your Software Investment*. Wiley: New York, NY, 1997.
2. Bennett K, Ramage M, Munro M. Decision model for legacy systems. *IEE Proceedings Software* 1999; **146**(3):153–159.
3. De Lucia A, Fasolino AR, Pompella E. A decisional framework for legacy system management. *Proceedings of the International Conference on Software Maintenance*, Florence, Italy, 2001. IEEE CS Press: Silver Spring, MD, 2001; 642–651.
4. Sneed HM. Planning the reengineering of legacy systems. *IEEE Software* 1995; **12**(1):24–34.
5. Visaggio G. Value-based decision model for renewal processes in software maintenance. *Annals of Software Engineering* 2000; **9**(1–2):215–233.
6. Aversano L, Canfora G, De Lucia A. Migrating legacy system to the Web: A business process reengineering oriented approach. *Advances in Software Maintenance Management*: *Technologies and Solutions*, Polo M (ed.). Idea Group Publishing: U.S.A., 2003; 151–181.
7. Sneed HM. Risks involved in reengineering projects. *Proceedings of the 6th IEEE Working Conference on Reverse Engineering*, Atlanta, GA, 1999. IEEE CS Press: Silver Spring, MD, 1999; 204–211.
8. Brodie ML, Stonebraker M. *Migrating Legacy Systems*. Morgan Kaufmann: San Francisco, 1995.

9. Canfora G, Cimitile A, De Lucia A, Di Lucca GA. Decomposing legacy programs: A first step towards migrating to client–server platforms. *Journal of Systems and Software* 2000; **54**:99–110.
10. Bovenzi D, Canfora G, Fasolino AR. Enabling legacy system accessibility by Web heterogeneous clients. *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, Victoria, BC, Canada, 2003. IEEE CS Press: Silver Spring, MD, 2003; 73–81.
11. Chiang CC. Wrapping legacy systems for use in heterogeneous computing environments. *Information and Software Technology* 2001; **43**(8):497–507.
12. Lin JM, Hong ZW, Fang GM, Jiau HC, Chu WC. Reengineering windows software applications into reusable CORBA objects. *Information and Software Technology* 2004; **46**(6):403–413.
13. Sneed HM. Encapsulating legacy software for use in client/server systems. *Proceedings of Working Conference on Reverse Engineering*, Monterey, CA, 1996. IEEE CS Press: Silver Spring, MD, 1996; 104–119.
14. Sneed HM. Wrapping legacy COBOL programs behind an XML-interface. *Proceedings of Working Conference on Reverse Engineering*, 2001. IEEE CS Press: Silver Spring, MD, 2001; 189–197.
15. Canfora G, De Lucia A, Di Lucca GA. An incremental object oriented migration strategy for RPG legacy systems. *International Journal of Software Engineering and Knowledge Engineering* 1999; **9**(1):5–25.
16. Rahgozar M, Oroumchian F. An effective strategy for legacy systems evolution. *Journal of Software Maintenance: Research and Practice* 2003; **15**:325–344.
17. Wu B, Lawless D, Bisbal J, Wade V, Grimson J, Richardson R, O'Sullivan D. The butterfly methodology: A gateway-free approach for migrating legacy information systems. *Proceedings of 3rd IEEE International Conference on Engineering of Complex Computer Systems*, Como, Italy, 1997. IEEE CS Press: Silver Spring, MD, 1997; 200–205.
18. Butler JG. *Mainframe to Client/Server Migration*. Computer Technology Research Corp.: Charleston, SC, 1996.
19. Sneed HM, Nyary E. Downsizing large application programs. *Journal of Software Maintenance: Research and Practice* 1994; **6**(5):105–116.
20. Serrano MA, Carver DL, Montes de Oca C. Reengineering legacy systems for distributed environments. *Journal of Systems and Software* 2002; **64**(1):37–55.
21. Aversano L, Canfora G, Cimitile A, De Lucia A. Migrating legacy systems to the Web: An experience report. *Proceedings of the 5th European Conference on Software Maintenance and Reengineering*, Lisbon, Portugal, 2001. IEEE CS Press: Siver Spring, MD, 2001; 148–157.
22. Bodhuin T, Guardabascio E, Tortorella M. Migrating COBOL systems to the Web by using the MVC design pattern. *Proceedings of the 9th Working Conference on Reverse Engineering*, Virginia, U.S.A., 2002. IEEE CS Press: Silver Spring, MD, 2002; 329–338.
23. Lewis G, Morris E, Smith D. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. *Proceedings of the Conference on Software Maintenance and Reengineering*, 2006. IEEE CS Press: Silver Spring, MD, 2006; 15–23.
24. Litoiu M. Migrating to Web services: A performance engineering approach. *Journal of Software Maintenance and Evolution: Research and Practice* 2004; **16**:51–70.
25. Zdun U. Reengineering to the Web: A reference architecture. *Proceedings of 6th European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, 2002. IEEE CS Press: Silver Spring, MD, 2002; 211–216.
26. Sneed HM. Encapsulation of legacy software: A technique for reusing legacy software components. *Annals of Software Engineering* 2000; **9**(1–4):293–313.
27. Merlo E, Gagn PY, Gilard JF, Kontogiannis K, Hendren L, Panangaden P, De Mori R. Reengineering user interfaces. *IEEE Software* 1995; **12**:64–73.
28. Stroulia E, El-Ramly M, Iglinski P, Sorenson P. User interface reverse engineering in support of interface migration to the Web. *Automated Software Engineering* 2003; **3**(10):271–301.
29. Moore M. User interface reengineering. *PhD Dissertation*, College of Computing, Georgia Institute of Technology, Atlanta, GA, 1998.
30. Moore M, Moshkina L. Migrating legacy user interfaces to the internet: Shifting dialogue initiative. *Proceedings of Working Conference on Reverse Engineering*, Brisbane, Australia, 2000. IEEE CS Press: Silver Spring, MD, 2000; 52–58.
31. Zhang Q, Chen R, Zou Y. Reengineering user interfaces of E-commerce application using business processes. *22nd International Conference on Software Maintenance*, 2006. IEEE CS Press: Silver Spring, MD, 2006; 428–437.
32. Orfali R, Harkey D, Edwards J. *The Essential Distributed Objects Survival Guide*. Wiley: New York, NY, 1996.
33. IBM WebSphere software: Legacy modernization with WebSphere Studio Enterprise Developer, 2002. http://www.redbooks.ibm.com/redbooks/pdfs/sg246806.pdf [24 December 2007].
34. Weiderman N, Northrop L, Smith D, Tilley S, Wallnau K. Implications of distributed object technology for reengineering. http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97tr005.pdf [24 December 2007].
35. Lin JM, Hong ZW, Fang GH, Lee CT. A style for integrating MS-Windows software applications to client–server systems using Java technology. *Software: Practice and Experience* 2006; **37**(4):417–440.
36. Hong ZW, Lin JM, Fang GM, Jiau HC, Chiou CW. Encapsulating Windows-based software applications into reusable components with design patterns. *Information and Software Technology* 2006; **48**(7):619–629.

37. Goedicke M, Zdun U. Piecemeal legacy migrating with an architectural pattern language: A case study. *Journal of Software Maintenance and Evolution: Research and Practice* 2002; **14**(1):1–30.
38. Canfora G, Fasolino A, Frattolillo G, Tramontana P. Migrating interactive legacy systems to Web services. *Proceedings of the Conference on Software Maintenance and Reengineering*, 2006. IEEE CS Press: Silver Spring, MD, 2006; 24–36.
39. Sneed HM, Sneed SH. Creating Web services from legacy host programs. *Proceedings of 5th International Workshop on Web Site Evolution*, Amsterdam, The Netherlands, 2003. IEEE CS Press: Silver Spring, MD, 2003; 59–65.
40. Sneed HM. Integrating legacy software into a service oriented architecture. *Proceedings of the Conference on Software Maintenance and Reengineering*, 2006. IEEE CS Press: Silver Spring, MD, 2006; 3–14.
41. Bodhuin T, Guardabascio E, Tortorella M. Migration of non-decomposable software systems to the Web using screen proxies. *Proceedings of the 10th Working Conference on Reverse Engineering*, Victoria, BC, Canada, 2003. IEEE CS Press: Silver Spring, MD, 2003; 165–174.
42. Legacy J. http://www.legacyj.com/lgcyj_perc1.html [24 December 2007].
43. Pfleeger SL, Menezes W. Marketing technology to software practitioners. *IEEE Software* 2000; **17**(1):27–33.
44. Raghavan SA, Chand DR. Diffusing software engineering methods. *IEEE Software* 1989; **6**(4):81–90.
45. Redwine ST, Riddle WE. Software technology maturation. *Proceedings of 8th International Conference on Software Engineering*, London, U.K., 1985. IEEE CS Press: Silver Spring, MD, 1985; 189–200.
46. Rogers EM. *Diffusion of Innovation* (4th edn). Free Press: New York, NY, 1995.
47. Colosimo M, De Lucia A, Francese R, Scanniello G. Assessing legacy system migration technologies through controlled experiments. *Proceedings of 23rd IEEE International Conference on Software Maintenance*, Paris, France, 2–5 October 2007. IEEE CS Press: Silver Spring, MD, 2007; 365–374.