

- M

INDICE

1. INTRODUZIONE

2. ARCHITETTURA TARGET E STRATEGIA DI MIGRAZIONE

3. MIGRAZIONE DI UN PROGRAMMA SCRITTO IN ACUCOBOL-GT

- *FASE DI PRE-PROCESSING*
- *FASE DI RESTRUCTURING AND WRAPPING*
- *FASE DI GUI REENGINEERING*
- *RUN DELL'APPLICAZIONE MIGRATA*

4. APPENDICE

INTRODUZIONE

Questa guida descrive una strategia per la migrazione di sistemi legacy (LIS) scritti in ACUCOBOL-GT verso sistemi web multi-tier.

Prima di procedere con un esempio pratico, è utile presentare le fasi del processo di migrazione che prevede di separare il layer relativo alle Graphical User Interface (GUI) dall'application layer e dal data layer in modo da wrappare il sistema legacy solo a livello di user interface.

ARCHITETTURA TARGET
E STRATEGIA DI
MIGRAZIONE

L'architettura target adottata è rappresentata nella Figura 1. E' stato sviluppato un *Wrapper* sottoforma di DLL, necessario alla comunicazione tra le interfacce migrate e l'application layer del LIS. In particolare il Wrapper funge da middleware in quanto permette di eseguire il sistema LIS ristrutturato e di gestire la sincronizzazione tra quest'ultimo e le interfacce reingegnerizzate. Sono due le interfacce di cui si compone il wrapper. La prima è fornita al LIS ristrutturato che è stato trasformato in un programma batch. La seconda è fornita al *GUIDeliverer*, componente che permette alle interfacce grafiche reingegnerizzate di accedere alle funzionalità del LIS. Le pagine dinamiche generate ed i relativi controlli costituiscono il componente *ReengineeredGUI*.

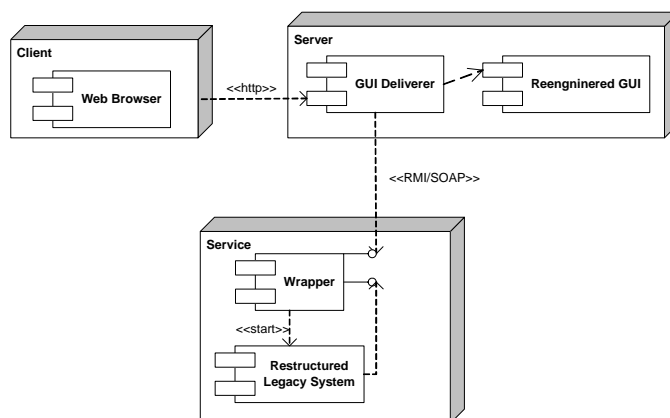


Figura 1 Architettura target

La prima fase della strategia, detta *Pre-processing*, fornisce dettagli sul tipo e sugli attributi di ogni oggetto grafico che compone la GUI del sistema legacy. In questa fase è possibile classificare le procedure associate ad ogni ENTRY-FIELD:

- procedure addette al controllo sulla validità del formato del campo e dunque migrabili lato client con un linguaggio di scripting quale JavaScript.
- procedure che accedono al DB ed operazioni di logica applicativa invocate dal *Wrapper*

Ogni SCREEN SECTION gestisce input ed output usando gli statement DISPLAY ed ACCEPT.

Nella fase di *Restructuring and Wrapping* ogni call relativa a questi due statement, viene rimpiazzata da istruzioni che incapsulano la comunicazione con il *Wrapper component*. Sempre in questa fase verrà deciso quale delle procedure AFTER/BEFORE saranno migrate lato client con un linguaggio di scripting.

La fase di *GUI Reengineering* permette di creare le pagine web dinamiche per ogni sottosistema migrato.

Il passo finale dell'approccio prevede l'integrazione delle nuove GUI e del sistema legacy ristrutturato in modo da produrre il sistema target.

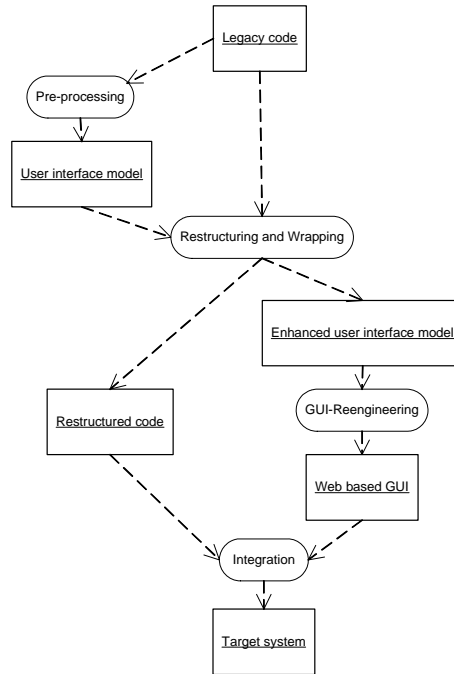


Figura 2 Strategia di migrazione

INTERFACCE DEL WRAPPER

Vediamo ora quali sono le interfacce messe a disposizione dal wrapper per comunicare con il sistema cobol e con l'applicazione web (Figura 3).

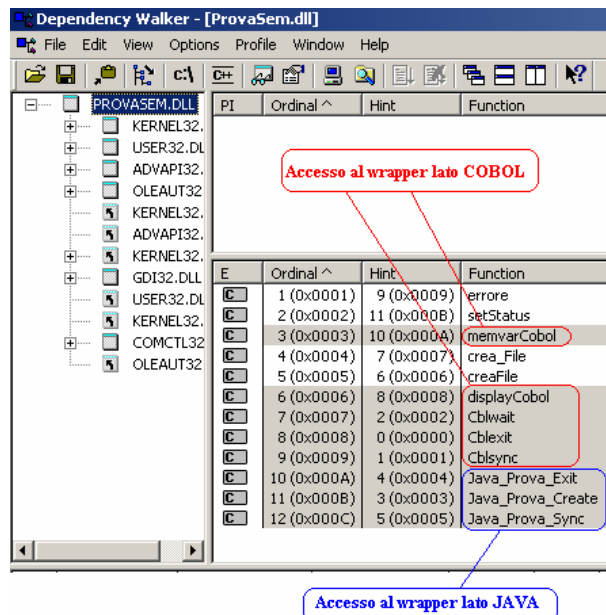


Figura 3 Interfacce wrapper

Il wrapper è stato implementato come una DLL (*PROVASEM.DLL*) scritta in

DELPHI questo perché ACUCOBOL-GT v6.0, non permette di comunicare direttamente con sistemi JAVA. Ogni qualvolta si deve far uso del wrapper lato COBOL, è necessario che prima di ogni sua chiamata venga inserito:

```
set environment 'DLL_CONVENTION' to "1".
call 'c:\ProvaSem.dll'.
```

.....

e, subito dopo il termine delle operazioni, venga inserito:

```
cancel 'c:\ProvaSem.dll'
set environment 'DLL_CONVENTION' to "0".
```

INTERFACCE LATO COBOL

Come già detto nella fase di *Restructuring and Wrapping*, vanno sostituite tutte le coppie di chiamate ACCEPT/DISPLAY con una call al Wrapper e vanno individuate le AFTER/BEFORE da migrare lato client. E' inoltre necessario inserire due funzioni addette alla corretta sincronizzazione tra i due sistemi. Per semplicità saranno chiamate WAIT ed EXIT. La WAIT permette di bloccare l'esecuzione del sistema cobol solo dopo la sua inizializzazione: così facendo, si resta in attesa mentre il sistema web sta eseguendo. La EXIT invece, serve a terminare l'esecuzione del sistema COBOL. Queste due procedure vanno inserite una volta sola. La prima va inserita, di norma, subito dopo la MAIN PROGRAM SECTION che da avvio al sistema LIS. La seconda invece, va inserita nel punto in cui si vuol far terminare l'esecuzione del LIS. Si riporta di seguito il codice per le due procedure:

```

WAIT      set environment 'DLL_CONVENTION' to "1".
          call 'c:\ProvaSem.dll'.
          call 'Cblwait' using by content session.
          cancel 'c:\ProvaSem.dll'.
          set environment 'DLL_CONVENTION' to "0".

EXIT      set environment 'DLL_CONVENTION' to "1".
          call 'c:\ProvaSem.dll'.
          call 'Cblexit' using by content session.
          cancel 'c:\ProvaSem.dll'.
          set environment 'DLL_CONVENTION' to "0".
```

Vediamo ora in che modo il wrapper permette di gestire il passaggio di informazioni dal sistema COBOL all'applicazione WEB. La funzione cobol in questione è la seguente:

```
set environment 'DLL_CONVENTION' to "1".
call 'c:\ProvaSem.dll'.
call 'memvarCobol' using
      by content name-VAR
      by content value-VAR
      giving RET-DLL.
if RET-DLL not = 1
  *insert error
end-if.
.....
cancel 'c:\ProvaSem.dll'
set environment 'DLL_CONVENTION' to "0".
```

Qui è possibile inserire altre call `memvarCobol` per ogni dato da migrare

La funzione `memvarCobol` prende in INPUT `name-VAR` e `value-VAR` e

restituisce un valore booleano *RET-DLL*. Il compito della funzione è quello di scrivere una stringa del tipo *name-VAR,value-VAR* nel file di testo *var.txt* che l'applicazione web andrà poi a leggere per ottenere i dati. Questa operazione va eseguita per ogni campo che si vuole wrappare. Così facendo la DISPLAY di una certa SCREEN SECTION sarà stata sostituita e tutti i dati relativi agli oggetti grafici che la compongono saranno riportati nel file. A questo punto il sistema COBOL deve fermare la sua esecuzione passando il controllo all'applicazione web. Per far ciò bisogna far uso della funzione *displayCobol*

```
set environment 'DLL_CONVENTION' to "1".
call 'c:\ProvaSem.dll'.
call 'displayCobol' using by content SESSION
    by content NOME-PROG
    by content NOME-SCREEN
    by content op-dll
    giving RET-DLL.
cancel 'c:\ProvaSem.dll'.
set environment 'DLL_CONVENTION' to "0".
```

La funzione prende in INPUT i parametri *NOME-PROG* che contiene il nome del programma cobol in esecuzione, *NOME-SCREEN* ovvero il nome della SCREEN SECTION da wrappare, *op-dll* che rappresenta un tipo di operazione e restituisce un valore booleano *RET-DLL*. Il compito della funzione è quello di creare un file dal nome *file.xml* in cui verrà scritto il nome del programma cobol e la screen da wrappare.

INTERFACCE LATO JAVA

Dopo aver visto in che modo usare il wrapper lato cobol, vediamo come avviene l'interazione con il sistema JAVA. La DLL mette a disposizione tre metodi da utilizzare nell'applicazione web:

- *Create(String session)* permette di creare un semaforo necessario alla sincronizzazione tra i due sistemi. Questo metodo va chiamato prima di lanciare il sistema COBOL migrato perché l'inizializzazione di quest'ultimo è legata proprio al semaforo con quella data SESSION. Se il semaforo non è stato creato, COBOL non riuscirà a partire.
- *Sync(String session)* permette al sistema JAVA di mettersi in attesa per far eseguire le operazioni al sistema COBOL. In effetti solo dopo che COBOL avrà scritto i file *var.txt* e *file.xml* il sistema JAVA potrà riprendere la sua esecuzione. Allo stesso modo, COBOL riprenderà la sua esecuzione dopo che il sistema JAVA ha creato il file *pass.txt* contenente i nuovi dati.
- *Exit(String session)* permette di terminare l'esecuzione del sistema JAVA distruggendo il semaforo creato.

MIGRAZIONE DI UN PROGRAMMA SCRITTO IN ACUCOBOL-GT

La cartella presente sul desktop contiene sia i sorgenti COBOL da migrare, sia due file BAT necessari alla compilazione ed esecuzione del programma cobol da una finestra del PROMPT di MS-DOS. Entrambi gli script prevedono un parametro:

- `c:> compila.bat SORGENTECOBOL.cbl`
- `c:> run.bat SORGENTECOBOL.acu`

Per editare qualsiasi sorgente COBOL è possibile utilizzare qualunque TEXT EDITOR.

FASE DI PRE-PROCESSING

Nella fase di *Pre-processing* è necessario identificare ogni elemento grafico presente in una SCREEN SECTION. In questa sezione sarà proposto un esempio relativo alla migrazione di un ENTRY-FIELD e di una LABEL DINAMICA. I passi da seguire saranno i seguenti:

1. identificare la dichiarazione dell'oggetto grafico in questione
2. identificare eventuali procedure AFTER o BEFORE da migrare lato client
3. inserire delle variabili di supporto necessarie all'incapsulamento dei dati nel wrapper
4. identificare coppia di funzioni DISPLAY/ACCEPT da wrappare
5. inserire SECTION di supporto per gestire la comunicazione con il wrapper

Vediamo come eseguire tutte le operazioni appena elencate.

Passo 1.

Ogni programma cobol ha una SCREEN-SECTION in cui sono inserite le varie SCREEN che contengono gli oggetti grafici.

```

77 PLUTO pic X(20).
77 PIPPONUM pic 9(2).
.....
SCREEN SECTION.
  01 SCREEN-1 , exception esci.
  02 label "ciao! "
  02 label PLUTO
  02 entry-field using PIPPONUM
     after after-PIPPONUM.
     before before-PIPPONUM.

```

Dichiarazione variabile PLUTO di tipo STRINGA di 20 caratteri
Dichiarazione variabile PIPPONUM di tipo INTERO a due cifre

LABEL

LABEL DINAMICA

ENTRY-FIELD creato usando la variabile PIPPONUM. E' presente una procedura AFTER ed una procedura BEFORE.

Passo 2.

Analizziamo ora la procedura AFTER legata al campo PIPPONUM

```

1 after-PIPPONUM SECTION.
2 if PIPPONUM > 5 or PIPPONUM < 15
3   move "CIAO" to ALERT
4 exit.

```

call esterna alla routine ALERT

Come si vede dal codice, questa procedura può essere migrata sul client con un linguaggio di scripting per cui ci limiteremo a commentare le linee 2 e 3 della

procedura.

Vediamo la procedura BEFORE associata al campo PIPPONUM

```
1 before-PIPPONUM SECTION.
2 readFile(fileDB)
3 exit.
```

lettura dati da un file

Poichè tale procedura utilizza una funzione per leggere dei dati da un file, non può essere migrata lato client.

Passo 3.

In questo passo è necessario inserire delle variabili di supporto nella WORKING-STORAGE SECTION. Un primo gruppo di variabili serviranno per l'inizializzazione del wrapper. Poi verranno inserite variabili di supporto per ogni oggetto grafico (ENTRY-FIELD/LABEL DINAMICHE) da migrare:

Variabili per l'inizializzazione del wrapper

```
WORKING-STORAGE SECTION.
77 NOME-PROG PIC X(22) VALUE 'provaSW'.
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'.
```

file che conterrà i dati provenienti dalle interfacce grafiche reingegnerizzate

Variabili di supporto per l'ENTRY-FIELD PIPPONUM

```
77 name-PIPPONUM PIC X(100) VALUE 'PIPPONUM'.
77 value-PIPPONUM PIC X(100).
```

Variabili di supporto per la LABEL DINAMICA PLUTO

```
77 name-PLUTO PIC X(100) VALUE 'PLUTO'.
77 value-PLUTO PIC X(100).
```

Per ogni campo saranno dunque utilizzate due variabili di supporto. La prima serve a contenere il nome della variabile originale, la seconda il suo valore. Da notare che entrambe le variabili vanno dichiarate come STRINGHE perché dovranno essere scritte in un file di testo.

FASE DI *RESTRUCTURING AND WRAPPING*

Passo 4.

A questo punto è necessario identificare la coppia di istruzioni DISPLAY/ACCEPT da wrappare.

```
display screen-1
accept screen-1 on exception continue.
```

Queste due chiamate vanno commentate e sostituire con delle chiamate al WRAPPER.

la funzione PASSAGGIODATI-SCREEN-1 sostituisce la call *display screen-1*. Gli altri dettagli saranno forniti in seguito.

```
* display screen-1
perform PASSAGGIODATI-SCREEN-1.
MOVE 'SCREEN-1' to NOME-SCREEN.
MOVE x'00' to NOME-PROG (8:1).
MOVE x'00' TO NOME-SCREEN (9:1).
MOVE 'insert' to op-dll.
MOVE x'00' to op-dll (7:1).
```

inserimento carattere di fine stringa *x'00'* nella posizione 8 perché 7 è il numero di lettere che compongono il nome "provaSW" dato al programma

chiamata al WRAPPER

```
set environment 'DLL_CONVENTION' to "1".
call 'c:\ProvaSem.dll'.
call 'displayCobol' using
by content SESSION
```

uso della funzione *displayCobol* del wrapper: in questo momento il sistema JAVA sta elaborando i dati appena inviati

la funzione LETTURADATI-SCREEN-1 sostituisce la call **accept screen-1** Gli altri dettagli saranno forniti in seguito.

```

by content NOME-PROG
by content NOME-SCREEN
by content op-dll
giving RET-DLL.
cancel 'c:\ProvaSem.dll'.
set environment 'DLL_CONVENTION' to "0".
    
```

by content: parametri funzione
giving: valore di ritorno

```

* accept SCREEN-1 on exception continue.
perform LETTURADATI-SCREEN-1.
    
```

L'operazione di commentare la coppia di chiamate DISPALY/ACCEPT va fatta per ogni screen presente nel sistema.

Non va dimenticato però che, commentando tali istruzioni, non saranno più considerate le procedure AFTER/BEFORE associate agli oggetti grafici della screen. Per ovviare a tale problema, bisogna che tutte le BEFORE (non migrate lato client), vengano inserite prima della chiamata PASSAGGIODATI-SCREEN-1 e allo stesso modo, tutte le AFTER (non migrate lato client) vengano inserite dopo la chiamata LETTURADATI-SCREEN-1.

```

.....
perform before-NON-MIGRATA-1
perform before-NON-MIGRATA-2
perform before-PIPPONUM.
perform PASSAGGIODATI-SCREEN-1.
    
```

esempio di before non migrate

```

.....
perform LETTURADATI-SCREEN-1.
perform after-NON-MIGRATA-1
perform after-NON-MIGRATA-2
perform after-NON-MIGRATA-3
.....
    
```

esempio di after non migrate

Passo 5.

Vediamo ora in che modo gestire l'invio e la ricezione dei dati utilizzando il wrapper. In realtà dobbiamo definire il contenuto delle funzioni PASSAGGIODATI-SCREEN-1 e LETTURADATI-SCREEN-1 viste in precedenza. Prima di far ciò si ricorda che la comunicazione tra il sistema cobol ristrutturato e l'applicazione web avviene attraverso l'uso di alcuni file:

- **file.xml** questo file creato dal sistema cobol tramite il wrapper contiene il nome del programma cobol in esecuzione ed il nome della screen che si sta wrappando. Questa informazione è necessaria al sistema WEB quando dovrà stabilire quale JSP visualizzare.
- **var.txt** questo file verrà creato dal sistema cobol tramite il wrapper e conterrà le seguenti informazioni:

```

nomecampo, valorecampo
nomecampo, valorecampo
.....
        
```

 questo vale per ogni oggetto grafico migrato.
- **pass.txt** questo file verrà creato dall'applicazione WEB e conterrà i dati da inviare a COBOL formattati come segue:

```

nomecampo1, valorecampo
nomecampo1, valorecampo
.....

```

E' importante inserire il numero 1 subito dopo il nome del campo affinché il sistema cobol riceva correttamente i dati.

Vediamo ora in dettaglio il codice che implementa le funzioni PASSAGGIODATI-SCREEN-1 e LETTURADATI-SCREEN-1

PASSAGGIODATI-SCREEN-1 SECTION.

```

set environment 'DLL_CONVENTION' to "1".
call 'c:\ProvaSem.dll'.
call "creaFile" giving RET-DLL.
if RET-DLL not = 1
  * insert error
end-if.

move x'00' to name-PIPPONUM(9:1).
move 0 to RET-DLL.
move PIPPONUM to value-PIPPONUM.
move x'00' to value-PIPPONUM (100:1).
call 'memvarCobol' using
  by content name-PIPPONUM
  by content value-PIPPONUM giving RET-DLL.
if RET-DLL not = 1
  *insert error
end-if.

move x'00' to name-PLUTO(6:1).
move 0 to RET-DLL.
move PLUTO to value-PLUTO.
move x'00' to value-PLUTO (100:1).
call 'memvarCobol' using
  by content name-PLUTO
  by content value-PLUTO giving RET-DLL.
if RET-DLL not = 1
  *insert error
end-if.

.....
cancel 'c:\ProvaSem.dll'.
set environment 'DLL_CONVENTION' to "0".
EXIT.

```

chiamata al WRAPPER

carattere di fine stringa in posizione 9 perché PIPPONUM ha 8 lettere

la funzione *memvarCobol* permette di memorizzare nel file *var.txt* una linea del tipo *PIPPONUM, valore*

inserisco l'eventuale valore di PIPPONUM nella variabile di supporto

la funzione *memvarCobol* permette di memorizzare nel file *var.txt* una linea del tipo *PLUTO, valore*

chiusura comunicazione con il wrapper e fine della SECTION di supporto PASSAGGIODATI

Subito dopo la chiamata a questa funzione, va inserita la call al wrapper *DISPLAYCOBOL* vista in precedenza. Così facendo il sistema cobol rimane in attesa mentre il sistema web esegue. Solo dopo il termine delle operazioni dell'applicazione web il controllo ritornerà al programma cobol che riprenderà la sua esecuzione leggendo i dati contenuti nel file *pass.txt* mediante la funzione LETTURADATI-SCREEN-1.

LETTURADATI-SCREEN-1 SECTION.

```

move 'N' TO ret-x.
move '1' to name-PIPPONUM (9:1).
call 'c:\leggInput.wob' using
  nome-file
  name-PIPPONUM
  value-PIPPONUM
  ret-x.
if ret-x = 'S'
  move value-PIPPONUM to PIPPONUM CONVERT
end-if.

move 'N' TO ret-x.
move '1' to name-PLUTO (6:1).
call 'c:\leggInput.wob' using
  nome-file
  name-PLUTO
  value-PLUTO
  ret-x.
if ret-x = 'S'
  move value-PLUTO to PLUTO
end-if.

.....
.....
EXIT.
    
```

la funzione *leggInput* è stata appositamente creata per leggere dal file *pass.txt* i valori passati dal sistema JAVA. Prende in INPUT il nome del file ed il nome del campo e restituisce in OUTPUT il suo valore ed un flag booleano

E' importante aggiungere 1 dopo il nome della variabile.

poiché PIPPONUM è di tipo intero, è necessario convertire il valore in arrivo da stringa ad intero con la clausola CONVERT

La section LETTURADATI-SCREEN-1 usa la funzione LEGGINPUT sviluppata ad hoc per poter leggere dal file *pass.txt* i valori scritti dal sistema JAVA. Dopo aver compiuto queste operazioni il sistema COBOL continuerà la sua esecuzione. fino a che una nuova screen non verrà wrappata.

FASE DI GUI REENGINEERING

In questa fase vedremo in che modo creare le interfacce grafiche reingegnerizzate e tutte le componenti dell'applicazione web così come mostrato in **Figura 4**.

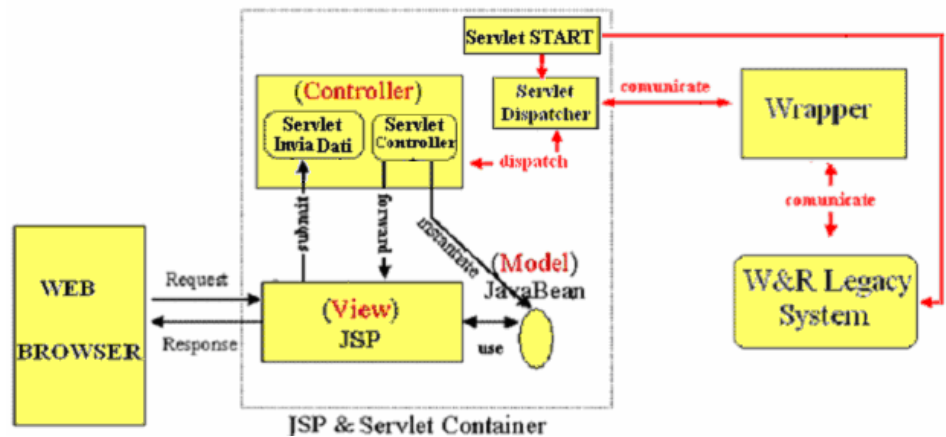


Figura 4 Componenti del sistema web

La *Servlet Start* è necessaria a creare il semaforo di sincronizzazione tra i due sistemi e ad avviare il programma cobol dopodichè passa il controllo alla *Servlet Dispatcher* che ha il compito di gestire la corretta esecuzione dell'intero sistema web: gestisce la comunicazione e sincronizzazione con il wrapper e stabilisce, in base alle richieste del LIS, quale pagina dinamica deve essere caricata in quel preciso momento. Ad ogni pagina JSP sarà associato un javabeen necessario a contenere i dati provenienti dal sistema legacy, e due servlet necessarie alla sincronizzazione con il Wrapper ed alla gestione di invio/ricezione dati col sistema legacy. In questa guida, per semplicità queste ultime quattro componenti saranno definite col nome di *Sottosistema Grafico*.

Vediamo ora alcuni dettagli implementativi di ciascun componente.

Servlet Start

Questa servlet (fornita all'utente) svolge due compiti fondamentali:

- creazione del semaforo necessario alla sincronizzazione tra i due sistemi:

```
Prova p = new Prova();
p.Create(sessionID);
```

la classe Prova sarà fornita all'utente

- start del sistema COBOL ristrutturato

```
String args[] = { "wrun32", "c:\\DIR\\file.acu" };
Runtime.getRuntime().exec(args);
```

File COBOL compilato

Terminate queste due operazioni, la servlet passerà il controllo attraverso una FORWARD alla *servlet Dispatcher*.

Servlet Dispatcher

Le funzioni svolte da questa servlet sono le più importanti perché da una parte essa gestisce la sincronizzazione con il LIS ristrutturato, dall'altra legge il file *file.xml* per stabilire quale sottosistema grafico deve essere eseguito in quel momento. La prima operazione da compiere è quella di aspettare che il sistema

COBOL termini le sue operazioni:

```
Prova p = new Prova();
p.Sync(sessionID);
```

A questo punto, nel momento in cui la servlet torna ad eseguire le sue operazioni, deve stabilire quale sottosistema grafico deve essere utilizzato per visualizzare i dati provenienti dal sistema COBOL:

la classe
ReadNextScreen
sarà fornita all'utente

```
ReadNextScreen nextScreen = new ReadNextScreen("C:\\file.xml");
String nameScreen = nextScreen.getNameScreen();

if (nameScreen.compareToIgnoreCase("SCREEN-1")==0)
    gotoPage("/SCREEN1_servletController", request, response);

if (nameScreen.compareToIgnoreCase("SCREEN-2")==0)
    gotoPage("/SCREEN2_servletController", request, response);
.....
```

Al termine delle operazioni il controllo passa dunque al sottosistema grafico appropriato.

Java Bean

Il bean deve contenere tutti i campi migrati (ENTRY-FIELD/LABEL DINAMICHE) del sistema COBOL e per ognuno di essi un metodo *getXxx* e *setXxx*. Considerando l'esempio cobol visto in precedenza, vediamo come generare i metodi del bean associati all'ENTRY-FIELD *PIPPONUM* :

```
private String PIPPONUM = "";
public String getPIPPONUM() {
    return PIPPONUM;
}
public void setPIPPONUM(String new_PIPPONUM) {
    this.PIPPONUM = new_PIPPONUM;
}
```

Servlet Controller

Il primo compito di questa servlet è quello di ISTANZIARE il java bean appena visto in modo da potervi inserire i dati letti dal file *var.txt* scritto da COBOL. Un esempio di file *var.txt* è il seguente:

var.txt

```
.....
PIPPONUM,7
PLUTO,cane
.....
.....
```

Il salvataggio in questo file avviene nel seguente modo: *nomecampo, valorecampo*
La servletController dovrà leggere il campo *PIPPONUM* ed utilizzare il metodo *setPIPPONUM(valorePIPPONUM)* del bean creato.

Dopo aver letto tutti i valori dal file e dopo averli inseriti nel bean, la servlet Controller passerà il controllo alla pagina JSP.

Pagina JSP

La pagina JSP deve riprodurre un'interfaccia grafica simile a quella del sistema legacy originale, deve usare il bean creato in precedenza per poter visualizzare i

MIGRATION GUIDE LINES

**RUN DELL'APPLICAZIONE
MIGRATA**

E' possibile eseguire l'applicazione migrata all'interno dell'ambiente stesso così come proposto in Figura 5.

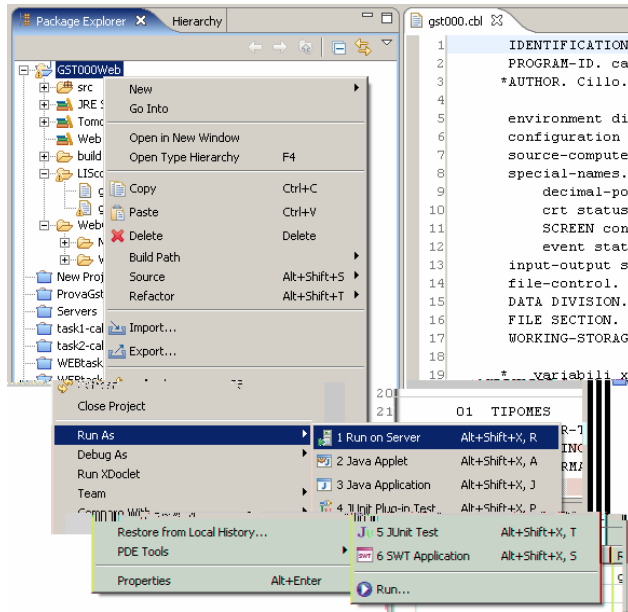


Figura 5 Run del sistema migrato

La Figura 6 riporta il sistema target in esecuzione.

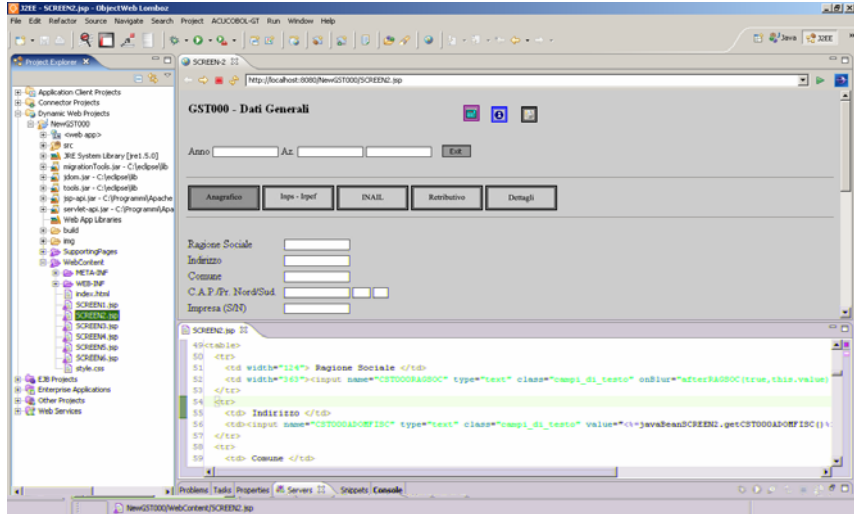


Figura 6 Sistema target in esecuzione

- 1) **Passaggio corretto di parametri numerici.** Tutti i dati scambiati tra il LIS ristrutturato ed il sistema Web attraverso il *Wrapper* sono in effetti delle STRINGHE. A tal proposito è necessario inserire l'istruzione cobol CONVERT nel momento in cui il sistema LIS si aspetta di ricevere dei dati di tipo NUMERICO dal sistema WEB. Supponiamo per esempio di avere dichiarato nel LIS un campo numerico di nome NUMERO: quando il wrapper restituirà il valore proveniente dall'interfaccia grafica ristrutturata per il campo NUMERO, è necessario aggiungere il comando CONVERT.

```

move 'N' TO ret-x.
move '1' to name-anno (5:1).
call 'c:\leggInput.wob' using
    nome-file name-NUMERO
    value-NUMERO ret-x.
if ret-x = 'S'
    move value-NUMERO to NUMERO CONVERT
end-if.

```

- 2) **Controllare i SALTII CONDIZIONATI.** Quasi tutti i sorgenti COBOL sono ricchi di *GOTO*, il che rende difficile gestire in modo corretto il flusso di controllo del sistema. E' necessario dunque porre molta attenzione a questi salti condizionati e commentare i GOTO quando serve.

```

if w-tasto not = 2 and 27 go to mp0.
if w-tasto=27 go to mp9.

```

- 3) **COMMENTARE call non necessarie.** E' stato più volte detto che la strategia di migrazione descritta, permette di migrare il sistema LIS a livello di interfaccia utente. Per questo motivo, è possibile commentare tutte quelle CALL relative alle interfacce grafiche: gestione degli eventi sui bottoni, inizializzazione del nome delle finestre etc.
- 4) **PROBLEMI con la SESSIONE.** A causa di alcuni problemi con la sessione, che funge da ID per istanziare correttamente il Wrapper, è necessario inserire **nel codice COBOL**, subito dopo la MAIN PROGRAM SECTION, il seguente codice:

```

move 'ABCDEFGHILMNOPQRSTUVWXYZ1234567890F' to SESSION.
MOVE x'00' to SESSION(33:1).

```

Lo stesso problema si presenta **lato JAVA** per cui nelle *servletStart* e *servletDispatcher* va inserita la seguente stringa:

```

static final public String sessionID =
    "ABCDEFGHILMNOPQRSTUVWXYZ1234567890F";

```