ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Conversione di interfacce grafiche legacy verso Windows Form e Web

Relazione finale in

Ingegneria Del Software

Relatore

Chiar.mo Prof.

Stefano Rizzi

Presentata da

Tommaso Bodini

Sessione II

Anno accademico 2014-2015

Indice generale

1 Introduzione

La continua evoluzione del software porta con sé nuove tecnologie e nuove tendenze. Molto spesso però i costi di sviluppo del software sono così elevati che l'adeguamento alle nuove tecnologie avrebbe un forte impatto economico. Inoltre le software-house difficilmente comprendono da subito i benefici che si otterrebbero dall'adozione di nuovi metodi di sviluppo del software e dall'adozione di nuovi linguaggi e tecnologie; inoltre il passaggio a tecnologie non compatibili con quelle passate scoraggia ulteriormente l'adozione di nuovi linguaggi poiché sarebbe necessario la re-implementazione di tutto il sistema e, nel caso di grossi sistemi, il cui processo evolutivo è durato decenni, diventa di fatto impraticabile.

D'altronde i nuovi linguaggi di programmazione emersi negli ultimi decenni portano con sé notevoli vantaggi; la programmazione ad oggetti ad esempio rende più facile lo studio di problemi complessi; i linguaggi di programmazione che si basano su macchina virtuale come Java e .NET forniscono al programmatore un ambiente di sviluppo protetto in cui egli non deve più gestire la memoria (grazie al garbage collector), oltre a fornire sistemi per il controllo delle eccezioni run-time e ricchissime librerie già comprese nel linguaggio.

Quando però alcuni decenni or sono molte piccole software-house iniziarono lo sviluppo di ERP, i linguaggi di programmazione disponibili erano pochi: C e COBOL erano i più gettonati: il C per la sua versatilità, il secondo per l'attitudine ad essere utilizzato nella manipolazione di dati. Le uniche interfacce disponibili con l'utente erano quelle a riga di comando ed i software erano in esecuzione su mainframe UNIX. Successivamente con l'avvento dei sistemi PC dotati di interfacce grafiche visuali GUI, le software-house hanno dovuto modificare i loro software sostituendo l'interfaccia a riga di comando con interfacce visuali ricompilando i loro software per sistemi operativi desktop come MS Windows.

Oggi però nuove tendenze rendono necessarie ulteriori trasformazioni al software; nuove interfacce grafiche sviluppate con i nuovi sistemi operativi rendono obsolete quelle precedenti, spesso create con ambienti di sviluppo non più aggiornati che le rendono incompatibili con i sistemi grafici dei nuovi sistemi operativi. Una tendenza molto attuale è quella di fornire al cliente l'applicazione, non più come strumento, ma come servizio («as a Service»), con la necessità di adattare il software a piattaforme Cloud. Infine, rispetto al passato, oggi sono disponibili molti più device, e molto diversi; ad esempio Smartphone e Tablet, dai quali l'utente vorrebbe interfacciarsi con il software

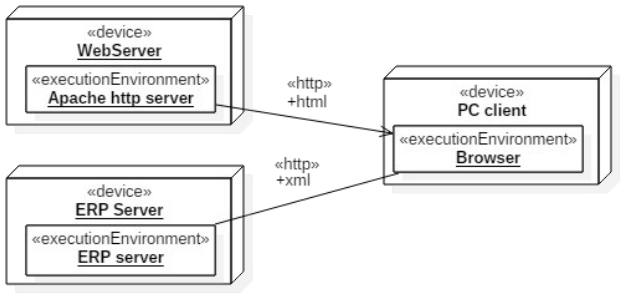
che ha sempre utilizzato su un sistema PC. Tali dispositivi per loro natura hanno una dotazione hardware e software completamente diversa rispetto ai tipici PC, per cui le applicazione ERP legacy sviluppate per PC difficilmente possono essere ricompilate per questi nuovi dispositivi.

La mia esperienza in una piccola software-house mi ha permesso di toccare con mano le problematiche che si hanno per mantenere gestionali ERP legacy su nuovi sistemi software ed hardware, anche a fronte di nuove richieste da parte dei clienti, su nuove piattaforme Web e su device differenti dai tradizionali PC.

2 Oggetto della tesi

La tematica che mi si è presentata durante la mia esperienza lavorativa, riguarda la sostituzione delle vecchie interfacce grafiche di un software ERP con interfacce grafiche più moderne, che potessero meglio interagire con i nuovi sistemi operativi. Il software ERP sul quale ho lavorato è completamente scritto in linguaggio C; è il frutto di oltre trent'anni di sviluppo ed è quindi composto da milioni di righe di codice; le prime versioni infatti erano disponibili solo con interfaccia utente a riga di comando, in ambiente UNIX; successivamente il software ha subito una notevole ristrutturazione per il passaggio ad interfacce grafiche. Il C non supporta nativamente alcun tipo di interfaccia grafica, non essendo presente nella libreria standard del linguaggio alcuna libreria per l'implementazione di GUI. Per implementare una GUI in linguaggio C è quindi necessario appoggiarsi a librerie software di terze parti. Per quanto riguarda il software ERP sul quale ho lavorato, esso adotta un'interfaccia grafica sviluppata da una software-house americana, che ormai non aggiorna più tale prodotto, lasciando la relase corrente della libreria software alla metà degli anni novanta. Tale versione non è più completamente compatibile con il sistema grafico dei nuovi sistemi operativi di casa Microsoft per il quale il software ERP è rilasciato. Da questa esigenza è scaturita la necessità di sostituire l'interfaccia grafica ma, non essendo disponibile né una nuova versione del prodotto, né un prodotto sostitutivo compatibile, si è resa necessaria la realizzazione di una libreria compatibile a livello di interfaccia software con quella originale, che adottasse inoltre una nuova libreria di interfaccia grafica tra quelle disponibili sul mercato. Una scelta alternativa, come ad esempio l'adozione di una interfaccia grafica disponibile sul mercato per il linguaggio C, ma diversa da quella originaria, avrebbe dovuto portare ad una revisione completa del codice; trattandosi di un software ERP completo, formato da centinaia di moduli, tale scelta avrebbe comportato un lavoro quantificabile nell'ordine degli anni con evidenti costi enormi.

Considerando inoltre le nuove tendenze che spingono la migrazione del software da ambienti desktop su servizi Cloud, si vuole realizzare una versione Web del gestionale ERP, realizzando in pratica una web application, con la possibilità di estendere i device dai quali è possibile utilizzare il software ERP, anche ad altri dispositivi quali PC con sistema operativo differente da quello Microsoft, o dispositivi mobile come tablet o smartphone. L'idea di base per questo secondo tipo di conversione sta nella sostituzione di un interfaccia desktop ad una in HTML5; trattandosi però di un software ERP per PC presenta innumerevoli differenze rispetto ad una tecnologia Web per la generazione di contenuti dinamici, come CGI o PHP. Le tecnologie web tradizionali infatti consistono in software o script che vengono eseguiti al momento della richiesta della pagina web da parte del client, per poi terminare l'esecuzione una volta terminato l'invio della pagina web appena creata. Un comune software per PC, come nel nostro caso, rimane in esecuzione dell'avvio al momento in cui l'utente termina il proprio lavoro e termina l'esecuzione del processo. A causa di queste differenze non si può quindi richiedere al gestionale di funzionare come una normale tecnologia web: come illustrato nei capitoli successivi si è deciso di adottare una separazione tra i livelli software lasciando alla pagina HTML il compito di logica di presentazione e al gestionale ERP i compiti di logica di business e strato dati, mentre la comunicazione tra i livelli avviene attraverso la tecnologia AJAX. Il diagramma seguente illustra il concetto che verrà sviluppato nei capitoli successivi.



3 Soluzione adottata per la conversione dell'interfaccia desktop

Come già illustrato brevemente nel capitolo precedente, l'idea di base per la sostituzione dell'interfaccia grafica è stata la creazione di una libreria software compatibile, dal punto dell'interfaccia pubblica esposta, con la libreria grafica originaria. Tale scelta permette di mantenere il codice C inalterato; è infatti sufficiente ricompilare i moduli software dell'ERP che si intende dotare della nuova interfaccia grafica, sostituendo alcune direttive di compilazione, e di linking, in maniera da far puntare dalla libreria software originaria alla nuova libreria sviluppata. Per nuova libreria software creata si è deciso di utilizzare delle librerie a collegamento dinamico, con estensione .dll; come linguaggio di programmazione si è deciso di adottare il C++ e il linguaggio C#, in ambiente MS Visual Studio.

Per quanto riguarda invece la nuova libreria grafica da utilizzare, dopo uno studio sulle varie alternative disponibili sul mercato, la scelta è ricaduta sulle Windows Form, grazie alla loro facilità di implementazione, supportata da editor visuali per GUI in ambiente MS Visual Studio. Inoltre questa scelta potrebbe, in futuro, permettere di re-implementare i singoli moduli del gestionale con linguaggi più evoluti e nuove tecnologie, mantenendo inalterata la GUI realizzata.

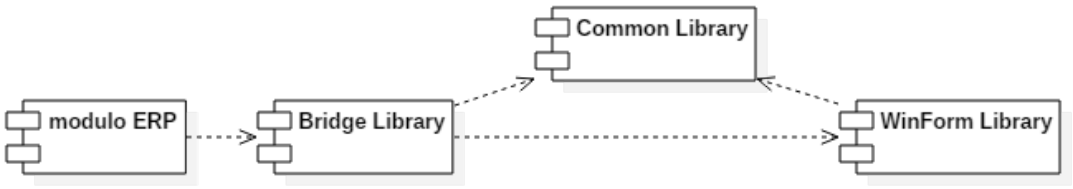
La scelta del C# è stata effettuata perché, oltre ad essere un linguaggio ad oggetti che supporta nativamente Windows Form, mette a disposizione del programmatore un ambiente protetto; basandosi infatti sul Framework .NET di casa Microsoft, dispone di garbage collector, oltre ad una ricca collezione di librerie per la manipolazione efficace dei dati. L'utilizzo del C++ si è reso necessario poiché, dovendo produrre una libreria a collegamento dinamico, il C++ è l'unico linguaggio che permette compatibilità con il C per quanto riguarda i tipi di dato e le strutture dati. Per arricchire le funzionalità del C++, e per meglio integrarlo ai componenti software scritti in C#, si è deciso di adottare una versione modificata del C++ disponibile in ambiente MS Visual Studio, denominata C++ CLI. La versione del C++ che integra CLI, arricchisce le funzionalità del C++ permettendo allo sviluppatore di utilizzare, all'interno del codice C++, oggetti .NET, e permettendo l'utilizzo del garbage collector per gli oggetti .NET creati all'interno del codice C++.

3.1 Descrizione architetturale della soluzione

La libreria software sviluppata si compone di tre moduli principali, ognuno di essi viene compilato in file .dll distinto e l'insieme dei tre moduli compone la libreria. I moduli sono denominati: Bridge Library, Common Library e WinForm Library

Tale separazione si è resa conveniente per rendere più estendibile il progetto; è infatti possibile fornire una nuova implementazione nella libreria WinForm adattandola ad altre interfacce grafiche senza la necessità di re-implementare le prime due che invece sono indipendenti dal funzionamento della libreria originaria e dal gestionale.

Il diagramma di interazione tra i moduli risulta essere:



Si analizzano ora le funzioni e le caratteristiche di ogni singolo modulo:

- **Bridge Library**

Questo componente software espone l'interfaccia pubblica della libreria sostituiva a quella originaria; implementa quindi la stessa interfaccia pubblica. Benché scritta il linguaggio C++ CLI non si tratta di un componente modellato con programmazione orientata agli oggetti poiché il suo scopo è solamente quello di implementare l'interfaccia delle librerie originarie, progettate esclusivamente per il linguaggio C. Il compito principale di questo modulo è quello di fungere appunto da ponte tra i moduli del gestionale e i successivi moduli software della libreria, con l'importante compito di adattare i tipi di dato e le strutture dati, da quelli standard del C, agli oggetti .NET equivalenti, possibile, adottando la tecnologia Microsoft CLI. Un esempio di tale importante funzionalità è la conversione di array di caratteri ad oggetti stringa del .NET e viceversa. La Bridge Library fa riferimento sia alla Common Library che alla WinForm Library.

- **Common Library**

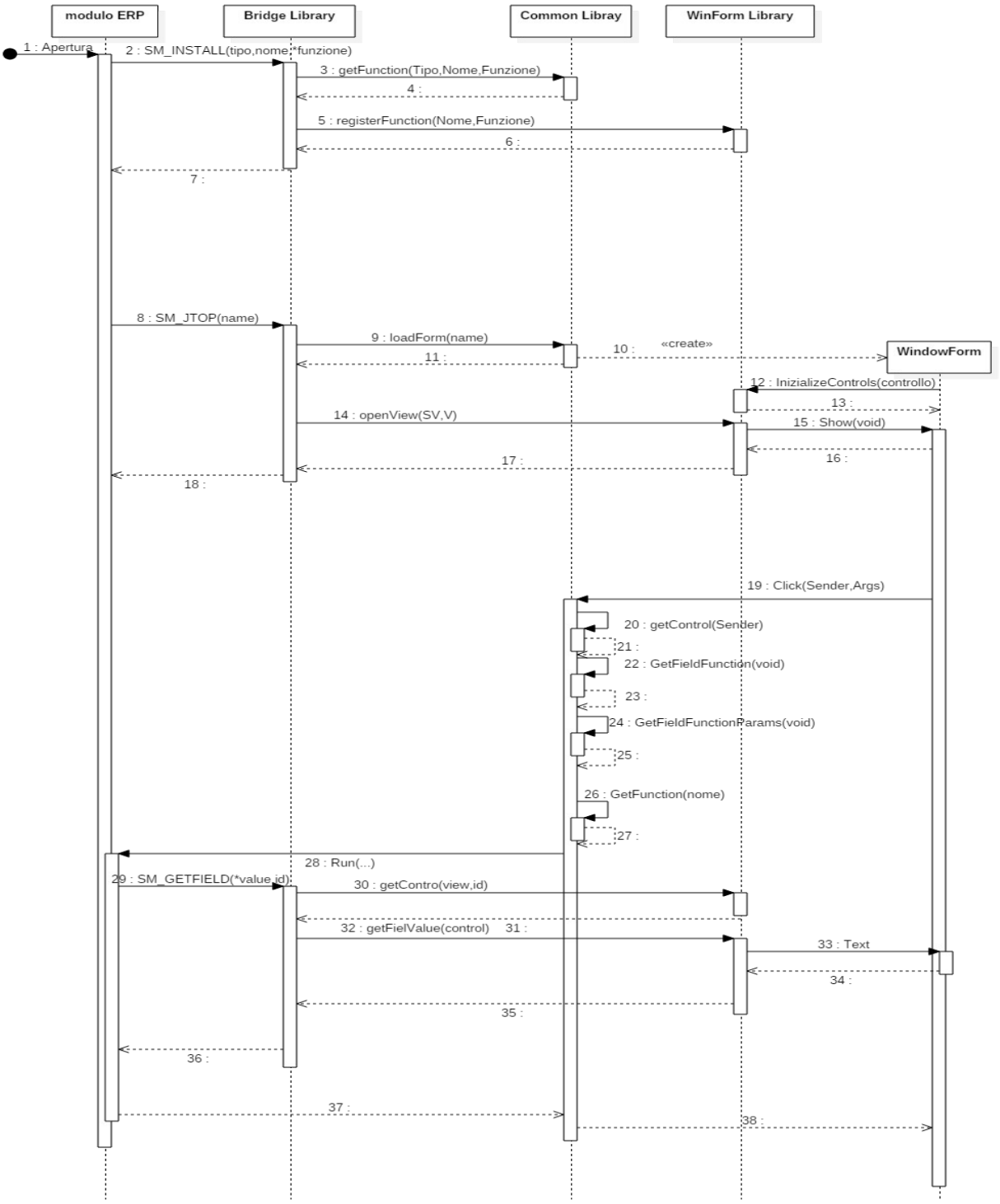
Anche questo componente della libreria è scritto in C++ CLI, ma a differenza di quello precedente, sfrutta la programmazione orientata agli oggetti. Non ha riferimenti alle altre componenti, al fine di evitare la creazione di riferimenti circolari. Il suo compito è di fornire l'interfaccia che la WinForm Library, o altra implementazione sostitutiva, deve implementare. Inoltre questo modulo deve effettuare quelle operazioni che richiedono accesso allo spazio di memoria del gestionale, che un linguaggio ad alto livello non potrebbe effettuare, come chiamare le funzioni a seguito di un evento sulla finestra. Definisce inoltre anche le interfacce per molti degli oggetti utilizzati nella WinForm Library, che sono indipendenti dall'effettiva implementazione delle stesse poiché legate o alla logica della libreria originaria, o alla logica del gestionale.

- **WinForm Library**

Questo componente a differenza dei precedenti è interamente scritto in C#, fa riferimento esclusivamente alla Common Library, evitando quindi riferimenti circolari, vietati dall'ambiente di sviluppo. Il compito di questo modulo è di implementare tutte quelle funzionalità che dipendono strettamente dall'interfaccia grafica effettiva, in questo caso windows form. L'utilizzo del C# permette una semplificazione della programmazione di questo complesso modulo, grazie alle funzionalità del linguaggio ed alla sua perfetta integrazione con l'interfaccia grafica. Le effettive schermate però non fanno parte di questo modulo; sono bensì presenti in una libreria dinamica esterna, in cui sono presenti tutte le schermate di tutti i moduli del gestionale. Sarà una funzione della Common Library, richiamata dalla Bridge Library, mediante reflection, a caricare la schermata dalla libreria dinamica che le contiene.

3.2 Descrizione dell'interazione tra moduli

Per chiarire meglio le interazioni che avvengono tra i moduli sopra descritti, e il gestionale, si riporta un diagramma di interazione e lo si commenta in ogni sua parte.



- 1: Apertura di un modulo dell' ERP dal menù di scelta
- 2: Come prima funzione il software registra tutte le funzioni di gestione degli eventi, ovvero la funzione da richiamare nel momento in cui si scatena l'evento, per questa operazione è

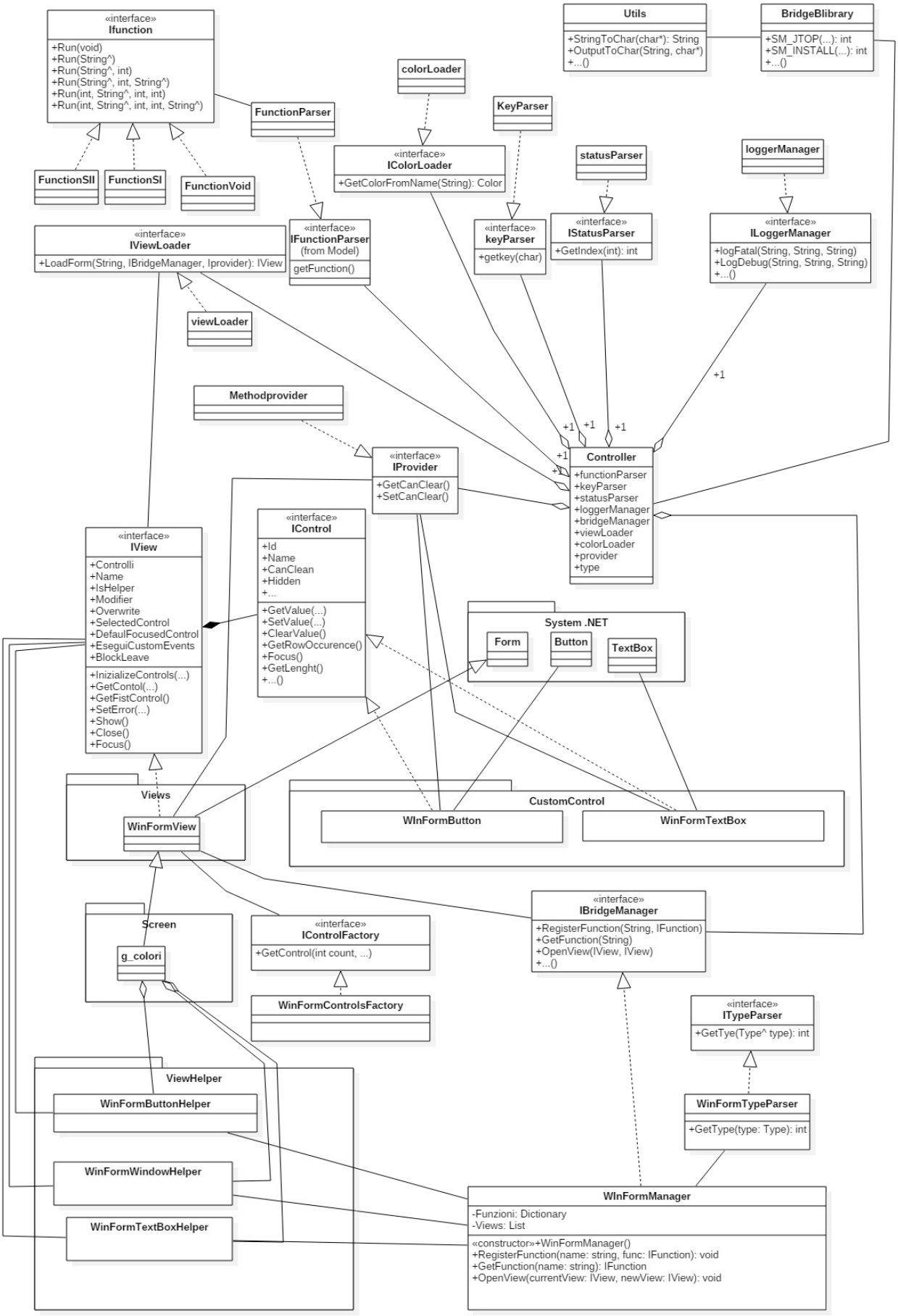
necessario fornire alla funzione SM_INSTALL il tipo di funzione, il suo nome simbolico e il puntatore alla funzione.

- 3: Bridge Libray chiama la funzione getFunction che crea un oggetto di tipo IFunction prendendo come parametri di input il tipo della funzione, il nome e il puntatore alla funzione. L'oggetto di tipo IFunction restituito rappresenta una funzione di gestione dell'evento.
- 5: Tale oggetto appena creato viene passato alla WinForm Library per essere inserito in una lista interna contenente l'elenco delle funzioni registrate associate al relativo nome.
- 8: Terminata la registrazione delle funzioni, il modulo del gestionale chiede alla libreria grafica l'apertura di una videata attraverso la funzione SM_JTOP, specificando come parametro il nome della funzione.
- 9: La Bridge Library chiede alla Common Library di provvedere al caricamento della schermata dalla libreria dinamica esterna al progetto che contiene tutte le schermate del gestionale ERP compilate.
- 10: Viene quindi istanziato da parte della Common Library un nuovo oggetto della classe che rappresenta la schermata desiderata, l'oggetto creato è di tipo Iview che estende ovviamente dalla classe del .NET WinForm.
- 12: Nel costruttore della classe Iview, viene chiamata la funzione InizializeControls che ricorsivamente provvede ad inizializzare tutti i controlli presenti nella screen, creando per ogni controllo presente nella screen, un relativo Icontrol, ovvero un oggetto che rappresenta al meglio un controllo nella libreria. La creazione di questo nuovo oggetto è necessaria per colmare le differenze che vi sono tra i controlli utente presenti nelle Windows Form e i controlli presenti nella libreria grafica originaria.
- 14: Una volta istanziata la WinForm richiesta, essa non ancora visibile. La Bridge Libray invoca quindi il metodo openView, passando come parametro la nuova finestra e la finestra attualmente aperta; compito di tale metodo è infatti mantenere l'ordine delle finestre aperte per permettere il passaggio ordinato da una finestra alla successiva.
- 15: La WinForm Library esegue in seguito il compito di mostrare la schermata a video, chiamando il metodo Show della classe WinForm contenuta nel framekork .NET
- 19: Questo che si descrive ora è un tipico scenario d'uso; una volta terminate tutte le operazioni di inizializzazione, l'utente, durante l'utilizzo dell'applicativo, effettua un click su un controllo utente; tale evento viene gestito dalla WinForm Library attraverso un handler presente in un suo oggetto, come illustrato più approfonditamente in seguito;

- 20: Vengono eseguite alcune operazioni preliminari: si ottiene l'Icontrol che ha scatenato l'evento
- 22: Si ottengo il nome della funzione da eseguire
- 24: Si ottengono i parametri di lancio della funzione
- 26: Si recupera la funzione dall'elenco di funzioni registrate, specificandone il nome, passato come parametro
- 28: Dopo aver ottenuto la funzione da richiamare, completa dei parametri, viene quindi richiamata ad opera della Common Library, che come già spiegato essendo scritta in C++ ha compatibilità con la strutture dati del C, tra cui i puntatori a funzione, di cui si avvale per eseguire la funzione.
- 29: Questa è una tipica funzione della libreria grafica richiamata dal gestionale, ovvero la funzione SM_GETFIEL che, prendendo in input l'identificatore di un campo, restituisce, per referenza nel puntatore passato come parametro, il contenuto di un controllo. Altre funzioni come ad esempio la SM_PUTFIELD che esegue il compito contrario, non illustrata in questo diagramma, hanno una struttura analoga e un comportamento molto simile.
- 30: Viene chiesto alla WinForm Library di restituire il l'Icontrol di cui è specificato l'identificatore
- 32: Viene chiesto all'Icontrol il valore del campo
- 33: L'Icontrol per ottenere il valore effettivo del campo richiama la proprietà Text tipica del controlli presenti delle WinForm .NET.

3.3 Descrizione strutturale

Benché il modulo Bridge Library non sfrutti la programmazione orientata ad oggetti per la sua strutturazione, per i motivi già illustrati, gli altri componenti si avvalgono invece della programmazione ad oggetti; si riporta il diagramma UML delle classi e se ne illustrano di seguito i compiti e le interazioni delle principali classi presenti; nel diagramma sono illustrate sia le classi della Common Library, sia quelle della WinForm Library. Come già illustrato, la Common Library contiene le funzioni indipendenti dell'interfaccia grafica adottata e mette in contatto la Bridge Library con la libreria che implementerà effettivamente la libreria grafica. La WinForm Library dipende fortemente dalla libreria grafica utilizzata, in questo caso windows form; per meglio interagire con tale libreria grafica è scritta in C#. Si riporta il diagramma UML delle classi e lo si illustra:


«interface»
Ifunction
+Run(void)
+Run(String^)
+Run(String^, int)
+Run(String^, int, String^)
+Run(int, String^, int, int)
+Run(int, String^, int, int, String^)
FunctionSII
FunctionSI
FunctionVoid
FunctionParser
colorLoader
«interface»
IColorLoader
+GetColorFromName(String): Color
KeyParser
statusParser
loggerManager
Utils
+StringToChar(char*): String
+OutputToChar(String, char*)
+...()
BridgeBlibrary
+SM_JTOP(...): int
+SM_INSTALL(...): int
+...()
«interface»
IViewLoader
+LoadForm(String, IBridgeManager, Iprovider): IView
«interface»
IFunctionParser
(from Model)
getFunction()
«interface»
keyParser
+getkey(char)
«interface»
IStatusParser
+GetIndex(int): int
«interface»
ILoggerManager
+logFatal(String, String, String)
+LogDebug(String, String, String)
+...()
viewLoader
Methodprovider
«interface»
IProvider
+GetCanClear()
+SetCanClear()
Controller
+functionParser
+keyParser
+statusParser
+loggerManager
+bridgeManager
+viewLoader
+colorLoader
+provider
+type
«interface»
IView
+Controlli
+Name
+IsHelper
+Modifier
+Overwrite
+SelectedControl
+DefaulFocusedControl
+EseguiCustomEvents
+BlockLeave
+InizializeControls(...)
+GetContol(...)
+GetFistControl()
+SetError(...)
+Show()
+Close()
+Focus()
«interface»
IControl
+Id
+Name
+CanClean
+Hidden
+...
+GetValue(...)
+SetValue(...)
+ClearValue()
+GetRowOccurence()
+Focus()
+GetLenght()
+...()
System .NET
Form
Button
TextBox
Views
WinFormView
CustomControl
WInFormButton
WinFormTextBox
Screen
g_colori
«interface»
IControlFactory
+GetControl(int count, ...)
WinFormControlsFactory
«interface»
IBridgeManager
+RegisterFunction(String, IFunction)
+GetFunction(String)
+OpenView(IView, IView)
+...()
«interface»
ITypeParser
+GetTye(Type^ type): int
WinFormTypeParser
+GetType(type: Type): int
ViewHelper
WinFormButtonHelper
WinFormWindowHelper
WinFormTextBoxHelper
WInFormManager
-Funzioni: Dictionary
-Views: List
«constructor»+WinFormManager()
+RegisterFunction(name: string, func: IFunction): void
+GetFunction(name: string): IFunction
+OpenView(currentView: IView, newView: IView): void
+1

3.4 Classi ed interfacce della Common Library

Controller: Questa classe è la principale della libreria; non ha nulla a che vedere con il controller nel pattern MVC, il suo scopo è quello di mantenere un'istanza di ogni altra classe presente nella libreria, facilitando l'accesso a tutti gli oggetti della Common Library in ogni punto in cui sono richiesti, è quindi sufficiente che un oggetto abbia riferimento ad essa per poter accedere a tutti gli oggetti della classe. Si presenta quindi come una classe di utilità, implementata attraverso il design pattern Singleton.

IFunction: Si tratta dell'interfaccia che rappresenta una funzione del gestionale che è stata registrata come hander per un evento. Per implementare tale interfaccia è necessario implementare il metodo Run in una delle diverse configurazioni possibili, infatti nell'interfaccia è presente l'override del metodo Run per tutte le configurazioni di parametri previste dal gestionale.

FunctionI-FunctioII,... Si tratta delle effettive implementazioni dell'interfaccia IFunction sopra descritta, ogni classe rappresenta un tipo di funzione richiamabile per la gestione dell'evento. Le proprietà di ognuna di queste classi sono il nome della funzione e il relativo puntatore a funzione del linguaggio C. Ogni classe implementa realmente un solo metodo Run, tra gli override presenti dell'interfaccia; i metodi non corrispondenti al tipo di classe generano un'eccezione.

FunctionParser: È una classe di utilità il cui compito è di effettuare un cast per le funzioni. Il metodo getFunction che prende in input il tipo della funzione (rappresentato da una enum) e un oggetto di tipo FunctionVoid, effettua il cast dalla FunctionVoid nell'IFunction corrispondente.

KeyParser: Si tratta di un'altra classe di utilità, ha il compito di convertire il codice ASCII dei tasti speciali delle tastiere, nel corrispondente formato utilizzato dal .NET per simulare la pressione dei tasti. Si è resa necessaria poiché il gestionale utilizza una funzione per la simulazione della pressione di un tasto nella finestra grafica.

StatusParser: È un ulteriore classe di utilità che ha lo scopo di convertire il formato utilizzato dalla vecchia libreria grafica per identificare la proprietà dei controlli utente, al nuovo formato utilizzato nella WinForm Library.

LoggerManager: Fornisce un insieme di metodi per effettuare il log della libreria. Tale funzione potrebbe sembrare secondaria, ma è fondamentale durante il processo di sviluppo.

ViewLoader: Questa classe fornisce il metodo LoadForm che, prendendo come parametro principale il nome della finestra da caricare apre, via reflection, il file .dll in cui sono contenute le schermate compilate, e restituisce un oggetto di tipo IView corrispondente alla finestra caricata.

ColorLoader: Ha il compito di convertire il nome di un colore, nel corrispondente valore RGB; il gestionale infatti utilizza i nomi dei colori per modificare alcune proprietà visive della finestra, poiché il formato RGB è l'unico supportato pienamente da tutti i sistemi grafici, si rende necessaria la conversione fornita da questa classe.

MethodProvider: Questa classe si è resa necessaria a causa delle differenze che vi sono tra la vecchia libreria grafica e le altre librerie grafiche disponibili. Molte proprietà dei controlli presenti nella vecchia libreria grafica infatti non trovano alcuna corrispondenza. Compito di questa classe è quindi di mantenere in memoria tutte quelle proprietà che non trovano corrispondenza.

Utill: Classe che contiene metodi di utilità generale, sopratutto per quanto riguarda la conversione di dati dalle strutture e tipi del C agli oggetti .NET.

IControl: È l'interfaccia che rappresenta un controllo utente nella libreria. Tale interfaccia è necessaria poiché, rispetto ai controlli standard presenti nelle librerie grafiche come Windows Form, i controlli grafici della vecchia libreria sostituita presentano caratteristiche differenti e metodi che non trovano una esatta corrispondenza. Ogni controllo utilizzato nella nuova interfaccia dovrà quindi implementare tale interfaccia per essere compatibile con la vecchia libreria grafica.

IView: Interfaccia che rappresenta una schermata nella libreria. Si è resa necessaria per motivazioni del tutto analoghe a quelle dei controlli a causa delle caratteristiche presenti nelle vecchie schermate, che non trovano corrispondenza nelle nuove. Le classi che implementano tale interfaccia diventano quindi compatibili con la vecchia libreria grafica a livello di proprietà.

IBridgeManager: Interfaccia che deve essere implementata dal gestore dell'interfaccia grafica effettiva, rende quindi possibile l'utilizzo di diverse interfacce grafiche, purché rispondano a tale interfaccia.

IControFactory: Interfaccia per la classe che dovrà essere definita dalla libreria dipendente dall'interfaccia grafica che espone il metodo per la creazione di un IControl, dato un generico controllo del tipo utilizzato nell'interfaccia grafica scelta, in questo caso Windows Form.

ITypeParser: Interfaccia che espone il metodi GetType che, preso in ingesso un oggetto che rappresenta il tipo di controllo utente disponibile nella nuova libreria grafica, restituisce il corrispondente tipo per il controllo nella vecchia interfaccia grafica.

3.5 Classi della WinForm Library

WinFormManager: È la classe principale della libreria poiché, implementando l'interfaccia IBridgeManager, risponde a tutti i metodi per la gestione dell'interfaccia grafica che dipendono dalla libreria grafica adottata. Tali metodi riguardano l'accesso ai controllo utente, la registrazione delle funzioni di gestione degli eventi, la lettura e l'impostazione di tutte le proprietà dei controlli utente presenti nell'interfaccia grafica e altre funzionalità che dipendono dall'effettiva libreria grafica utilizzata, in questo caso windows form.

WinFormControlsFactory: Classe presente nel pakage Extra, implementa l'interfaccia IControlFactory che nel suo metodo GetControl, che prende come parametro principale un controllo utente di qualsiasi tipo, restituisce un nuovo oggetto di tipo IControl corrispondente. Realizza di fatto il design pattern Factory.

WinFormTypeParser: Classe anche questa presente nel pakage Exra, implementa ITypeParser e quindi, nel suo metodo GetType, restituisce il corrispondente tipo di controllo grafico della vecchia libreria grafica, dato un controllo grafico Windows Form.

WinForm...Helper: Le classi appartenenti al pakage ViewHelper, come ad esempio WinFormButtonHelper, contengono i metodi richiamati dai rispettivi controlli utente allo scatenarsi di un evento; per i bottoni, ad esempio, i metodi previsti e quindi presenti nella classe WinFormButtonHelper sono Eneter, Leave e Click, che sono registrati come handler nei controlli utente effettivi. Il compito di questi metodi è di adattare il comportamento delle windows form, al comportamento della vecchia interfaccia; al verificarsi di un evento infatti viene chiamata la funzione del modulo del gestionale associata all'evento stesso e registrata nell'apposita lista, come spiegato nel capitolo precedente. Visto il funzionamento del gestionale, per quanto riguarda la gestione degli eventi, il nome di alcune funzioni chiamate varia a seconda del controllo chiamante, altri eventi invece, come ad esempio l'uscita da un controllo, chiamano sempre una funzione predefinita in ogni controllo di ogni schermata.

WinFormView: È una classe parziale, che rappresenta una schermata del gestionale. Eredita quindi dalla classe Form del Framework .NET. Contiene la gestione degli eventi che sono comuni a tutte le schermate del gestionale; non è istanziabile direttamente, è quindi necessario estendere tale classe nelle effettive schermate del gestionale che sono presenti in un progetto esterno, come già illustrato.

WinForm...: Le classi presenti nel pakage CustomControl, come ad esempio la classe WinFormButton, rappresentano i controlli utente personalizzati. Ogni classe presente nel pakage CustomControl implementa l'interfaccia Icontrol al fine di adattare le proprietà dei controlli presenti nella libreria grafica windows form alla libreria grafica originaria. Ogni «CustomControl» mantiene un riferimento all'oggetto .NET relativo controllo corrispondente; un oggetto della classe WinFormButton, per esempio, ha un riferimento ad un oggetto Button che è il bottone effettivo presente nella finestra con cui l'utente interagisce.

3.6 Implementazione delle viste

Come già accennato, le schermate sono presenti in un progetto esterno, nel diagramma UML delle classi, tali classi sono presenti nel pakage Screen. Ogni singola schermata è l'estensione della classe WinFormView, che realizza le proprietà comuni a tutte le schermate. L'ambiente di sviluppo mette a disposizione la possibilità di un designer visuale per la modellazione delle schermate, facilitando il lavoro di conversione. È però comunque necessario aggiungere del codice manualmente in ogni schermata; bisogna infatti aggiungere quelle proprietà dei controlli utente presenti nella libreria originaria ma non presenti nei controlli windows form; come già illustrato, la classe preposta a mantenere tali informazioni in memoria, fino al successivo utilizzo, è la classe MethodProdiver. Tale operazione viene effettuata nel costruttore di ogni singola finestra. Il costruttore necessita quindi di una referenza all'oggetto MethodProvider. Necessita inoltre di una referenza al gestore dell'interfaccia, con un oggetto di tipo IBridgeManager, per poter creare all'interno del costruttore stesso gli oggetti «Helper», della classe ViewHelper, che hanno il compito di registrare gli handler degli eventi scatenati dai controlli. Il costruttore degli oggetti «Helper» necessita infatti di una referenza al gestore dell'interfaccia grafica, per questo motivo il costruttore della schermata richiede sia un oggetto della classe MethodProvider che un oggetto che risponda all'interfaccia IBridgeManager.

Nel costruttore delle finestre vengono creati gli oggetti «Helper» per i relativi controlli presenti:

```
formHelper = new WinFormWindowHelper(this, manager, provider);
textBoxHelper = new WinFormTextBoxHelper(this, manager);
buttonHelper = new WinFormButtonHelper(this, manager);
```

Successivamente si registrano gli handler degli eventi per ogni controllo presente della finestra:

```
Load += formHelper.Load;
Activated += formHelper.Load;
Leave += formHelper.Leave;
KeyUp += formHelper.KeyUp;
Button1.Enter += buttonHelper.Load;
Button1.Click += buttonHelper.Click;
...
```

Nell'esempio precedente sono stati registrati gli eventi della finestra stessa: caricamento, attivazione, uscita dalla finestra e pressione di un tasto. In seguito si sono registrati gli eventi del bottone Button1. Si procede a tale registrazione degli handler di tutti gli eventi in tutti i controlli.

Terminata la registrazione degli hander, si procede al popolamento dell'oggetto «provider», di tipo MethodProvider, con tutte le proprietà che non trovano corrispondenza piena tra le proprietà dei controlli Windows Form.

```
…
provider.SetCanClean(Button1,true);
provider.SetEntryFunction(Button1,"s_entry");
provider.SetEntryParameter(Button1,"btn1");
provider.SetEntryMemo1(Button1,"Descrizione bottone");
...
```

In questo esempio di codice, all'oggetto Button1, vengono impostate quelle proprietà che non trovano corrispondenza nelle Windows Form, come, per esempio, la possibilità di cancellare il contenuto del bottone, la funzione associata all'ingesso, il suo parametro di lancio, e una descrizione del bottone, da visualizzare quando si va in focus su tale controllo.

4 Miglioramenti alla soluzione

La fase di realizzazione delle schermate, mediante l'apposito tool visivo messo a disposizione dall'ambiente di sviluppo, richiede un'ulteriore parte di programmazione con scrittura di codice nel costruttore della schermata, come spiegato nel capitolo precedente. Trattandosi di un gestionale ERP completo, formato da centinaia di videate differenti, nel caso di conversione di tutto il gestionale, sarebbe necessaria una notevole quantità di lavoro. Inoltre volendo realizzare anche un'implementazione web, lo sforzo per popolare l'oggetto «provider», illustrato nel capitolo precedente, con le proprietà dei controlli non replicabili, andrebbe effettuato due volte, uno per l'implementazione desktop e una per l'implementazione web della stessa interfaccia. Da ultimo il codice da inserire nel metodo costruttore delle schermate rende meno chiara la leggibilità dello stesso, «sporcandolo» con oggetti che non hanno nulla a che vedere con le form .NET in sé.

Per queste motivazioni si è provveduto ad una ristrutturazione del codice eliminando dalle form sia la gestione degli eventi, con gli oggetti «Helper», sia il popolamento dell'oggetto della classe MethodProvider, in fase di caricamento dell'interfaccia grafica. Inoltre è stata modificata la struttura della classe MethodProvider, ora infatti contiene coppie chiave-valore in cui la chiave non è più un oggetto generico, bensì una stringa contenete il nome del controllo utente che risulta evidentemente di più facile lettura dal file XML rispetto ad un oggetto generico.

Le differenze strutturali riguardano le classi «Helper» che sono ora in relazione con la classe WinFormView, nel capitolo precedente invece erano in relazione con le singole schermate. Nel costruttore della classe WinForView, si itera su tutti i controlli per realizzare gli oggetti Icontrol corrispondenti. In tale fase si registrano ora anche gli handler degli eventi, nei corrispondenti «Helper». Per il popolamento dell'oggetto MethodProvider si utilizza ora un file XML esterno al progetto; è stata quindi aggiunta una classe denominata xmlParser, che ha il compito di leggere il file XML e popolare l'oggetto del tipo MethodProvider passato come parametro. Tale funzione viene richiamata dalla classe ViewLoader dopo aver caricato la schermata richiesta. Un'ultima nota riguarda il metodo con cui si è effettuato il parsing per la lettura di documenti XML, in questo punto e nelle successive modifiche effettuate, è stata utilizzata la classe .NET Xdocument che permette non solo la navigazione all'interno del documento XML ma anche la sua interrogazione attraverso la tecnologia LINQ.

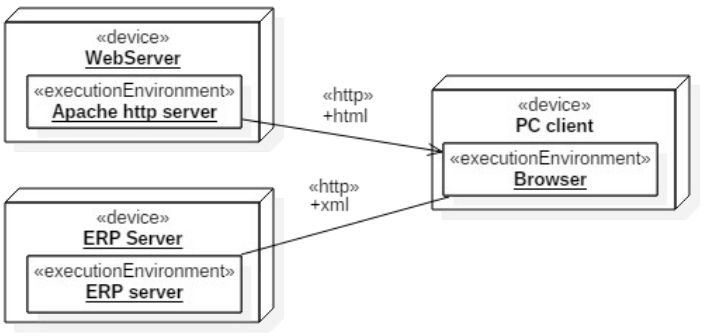
5 Soluzione adottata per la conversione web

Per la realizzazione di un interfaccia sostitutiva web, implementata in HTML5, si sono da subito, come già accennato nel capitolo introduttivo, presentati alcuni problemi: la tecnologia di un software desktop è infatti differente da una tecnologia web; le tecnologie web producono pagine HTML, il software di una tecnologia web rimane in esecuzione solamente per il tempo necessario per generare il codice HTML; in questo caso invece il processo di un modulo del gestionale, che rappresenta la sessione di lavoro dell'utente, rimane in esecuzione fino al termine della sessione stessa, con la necessità di comunicare con il browser web ad ogni aggiornamento dell'interfaccia. È quindi necessario aggiornare la pagina web ad ogni azione dell'utente e, viste le caratteristiche del gestionale, tali eventi sono molto comuni: anche la sola uscita da un capo comporta l'esecuzione di un evento. Le possibilità per l'aggiornamento della pagina web sono due:

- Ricaricare l'intera pagina web, operazione che richiede una nuova richiesta http dell'intera pagina, con i dati aggiornati e un'operazione di rendering dell'intera pagina a carico del browser.
- Aggiornare solamene i dati modificati, richiedere quindi l'aggiornamento dei dati a seguito di un evento tramite la tecnologia AJAX, codificando l'intero contenuto informativo di una schermata in XML ed incaricare uno script lato client di aggiornare l'interfaccia grafica.

Dopo una breve analisi delle due alternative è risultato subito evidente che la seconda presenta notevoli vantaggi; innanzitutto in termini di performance. Inviando sulla rete un solo documento XML contenente solamente i dati da visualizzare, in alternativa all'intera pagina web comprensiva dei dati aggiornati, si ottiene un risparmio di utilizzo del canale trasmissivo. In secondo luogo, la tecnologia AJAX permette un notevole miglioramento nella reattività dell'interfaccia grafica, non essendo più necessario il rendering dell'intera pagina web ad ogni evento. Ultima, ma non meno importante motivazione, riguarda la struttura della soluzione: adottando la seconda alternativa infatti, si ottiene una alta separazione tra interfaccia grafica e i dati in essa contenuti, rendendo possibile, in futuro, un implementazione diversa dell'interfaccia grafica, mantenendo invariata la logica di produzione dei dati su documenti XML. Sarebbe infatti possibile, ad esempio, utilizzare un interfaccia desktop scritta in Java che comunichi via http con il gestionale e visualizzi i dati in una interfaccia costruita appoggiandosi sulla libreria grafica Swing, senza modificare la libreria web del gestionale.

Il diagramma di deployment della soluzione adottata risulta essere:

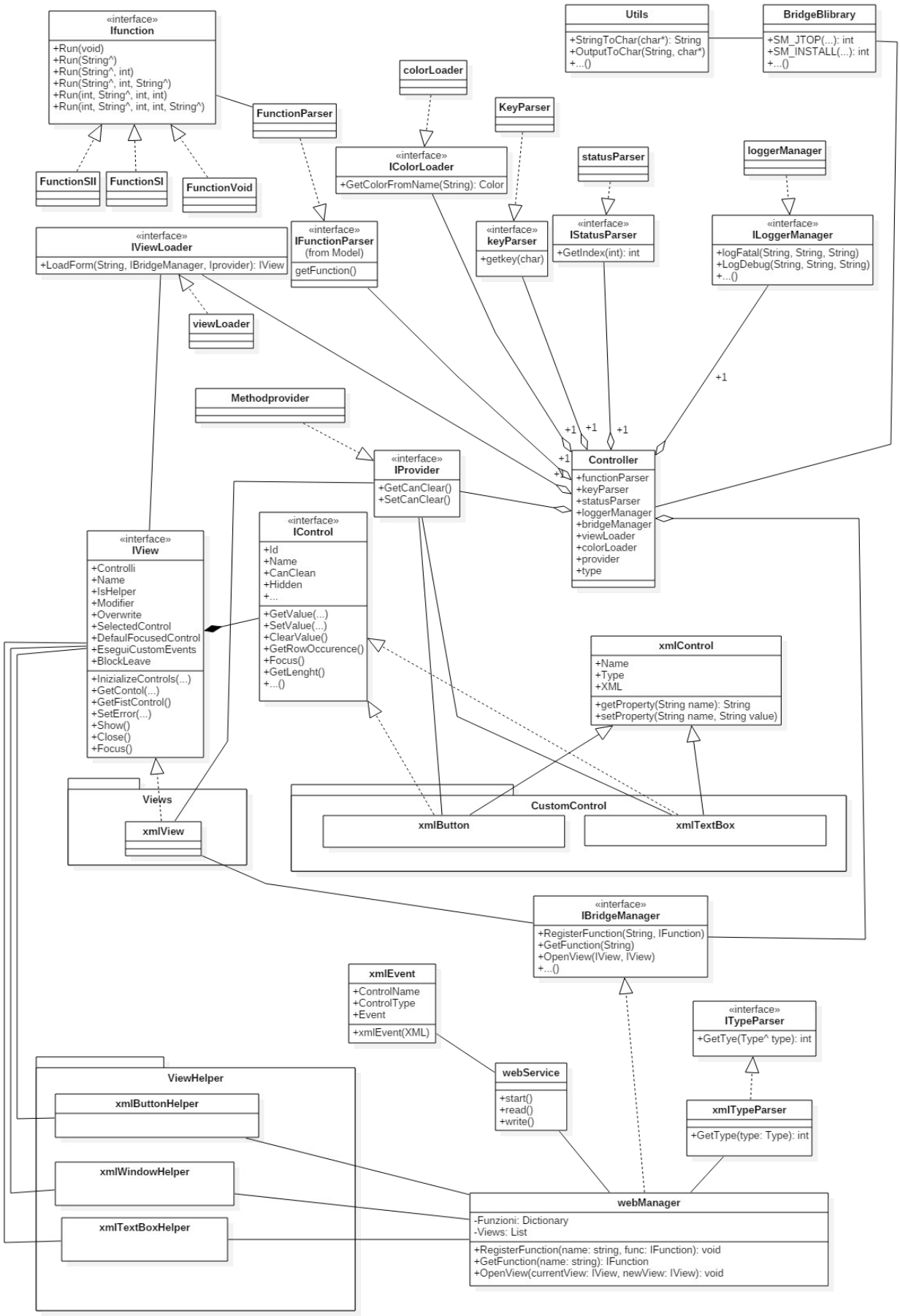


Come si può vedere dallo schema, è possibile separare le due funzioni di restituzione della pagine web e comunicazione con il gestionale ERP. Per ottenere le pagine web, che contengono i form che replicano le schermate originarie, è stato utilizzato un normale server http, in questo esempio Apache HTTP Server. La comunicazione in questo caso è monodirezionale in quanto il compito del web server è di restituire solamente la pagina web senza dati, senza alcuna elaborazione. Il server contenente il gestionale ERP invece comunica sempre con il protocollo HTTP, ma a differenza della prima volta, la comunicazione è bidirezionale e si inviano solo i dati codificati in XML e la comunicazione è instaurata dalla tecnologia AJAX. L'utente accede al servizio tramite un normale browser, può quindi utilizzare tipologie di device completamente differenti come PC tradizionali, smartphon o tablet.

Si analizzano ora le caratteristiche strutturali della nuova libreria.

5.1 Descrizione strutturale

Si riporta il diagramma UML delle classi:


«interface»
Ifunction
+Run(void)
+Run(String^)
+Run(String^, int)
+Run(String^, int, String^)
+Run(int, String^, int, int)
+Run(int, String^, int, int, String^)
FunctionSII
FunctionSI
FunctionVoid
FunctionParser
colorLoader
«interface»
IColorLoader
+GetColorFromName(String): Color
KeyParser
statusParser
loggerManager
Utils
+StringToChar(char*): String
+OutputToChar(String, char*)
+...()
BridgeBlibrary
+SM_JTOP(...): int
+SM_INSTALL(...): int
+...()
«interface»
IViewLoader
+LoadForm(String, IBridgeManager, Iprovider): IView
«interface»
IFunctionParser
(from Model)
getFunction()
«interface»
keyParser
+getkey(char)
«interface»
IStatusParser
+GetIndex(int): int
«interface»
ILoggerManager
+logFatal(String, String, String)
+LogDebug(String, String, String)
+...()
viewLoader
Methodprovider
«interface»
IProvider
+GetCanClear()
+SetCanClear()
Controller
+functionParser
+keyParser
+statusParser
+loggerManager
+bridgeManager
+viewLoader
+colorLoader
+provider
+type
+1
«interface»
IView
+Controlli
+Name
+IsHelper
+Modifier
+Overwrite
+SelectedControl
+DefaulFocusedControl
+EseguiCustomEvents
+BlockLeave
+InizializeControls(...)
+GetContol(...)
+GetFistControl()
+SetError(...)
+Show()
+Close()
+Focus()
«interface»
IControl
+Id
+Name
+CanClean
+Hidden
+...
+GetValue(...)
+SetValue(...)
+ClearValue()
+GetRowOccurence()
+Focus()
+GetLenght()
+...()
xmlControl
+Name
+Type
+XML
+getProperty(String name): String
+setProperty(String name, String value)
Views
xmlView
CustomControl
xmlButton
xmlTextBox
«interface»
IBridgeManager
+RegisterFunction(String, IFunction)
+GetFunction(String)
+OpenView(IView, IView)
+...()
xmlEvent
+ControlName
+ControlType
+Event
+xmlEvent(XML)
«interface»
ITypeParser
+GetTye(Type^ type): int
webService
+start()
+read()
+write()
xmlTypeParser
+GetType(type: Type): int
ViewHelper
xmlButtonHelper
xmlWindowHelper
xmlTextBoxHelper
webManager
-Funzioni: Dictionary
-Views: List
+RegisterFunction(name: string, func: IFunction): void
+GetFunction(name: string): IFunction
+OpenView(currentView: IView, newView: IView): void

Come si può notare dal confronto con il diagramma del capitolo precedente, le classi presenti nella «Bridge Library» e quelle della «Common Library» non hanno sostanzialmente subito modifiche, se non per il fatto che il metodo che caricava le schermate, presente nella Common Librery, ora non ha più senso di esistere. Modifiche sostanziali hanno invece riguardato la terza parte della libreria, quella dipendente dalla nuova libreria grafica. La nuova componente è stata denominata xmlLibrary, Le modifiche hanno riguardato:

xmlControl: Questa nuova classe modella un generico controllo xml. Presenta un costruttore che necessita del nome del controllo e del tipo di controllo, un metodo setPropriety per aggiungere una proprietà al controllo e un metodo getPropriety per ottenere una proprietà del controllo. Nome del controllo, tipo e xml del controllo sono impostate come proprietà pubbliche che è possibile leggere. Da notare è la proprietà XML in sola lettura, che è utilizzata in fase di creazione del messaggio XML destinato all'interfaccia grafica.

xmlEvent: Questa classe rappresenta l'evento da eseguire, il suo scopo è di fornire un parser per il documento XML per l'evento.

CustomControl: Ora i controlli personalizzati non contengono più alcun riferimento al controllo grafico corrispondente. In alternativa a ciò estendono la classe xmlControl. In questo pakage sono stati implementati momentaneamente i due soli controlli più diffusi, sufficienti comunque alla conversione di un buon numero di schermate.

Helper: Le classi presenti in questo pakage non hanno subito notevoli cambiamenti, se non per il fatto che, non essendoci più riferimenti ai controlli utente del .NET, essi sono stati sostituiti con delle stringhe contenenti il nome dei controlli.

xmlView: Questa classe sostituisce la classe WinFormView e non estende più la classe form. Implementa comunque l'interfaccia IView che rappresenta ancora una schermata del gestionale.

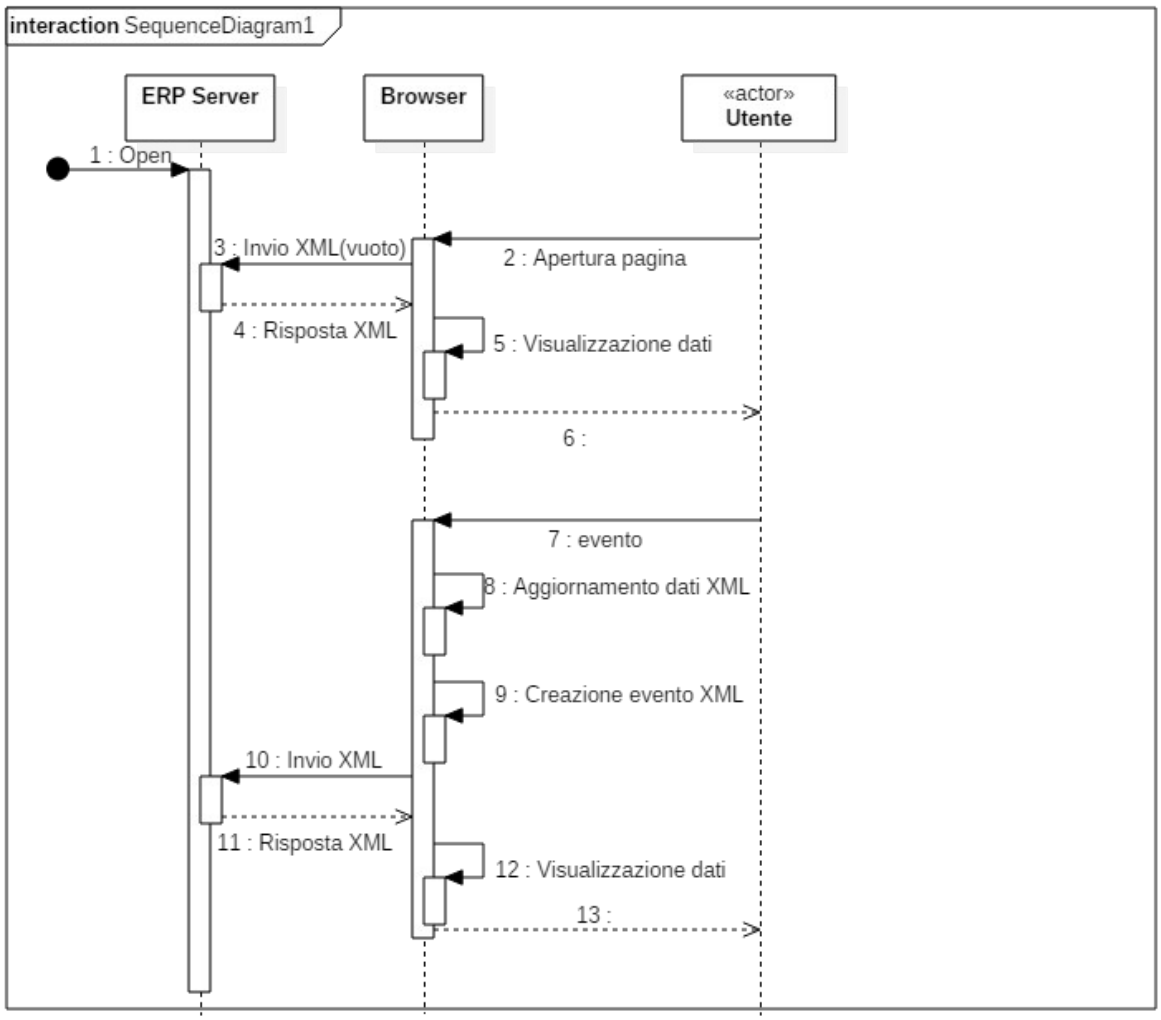
webService: Questa nuova classe incorpora le funzionalità per gestire la connessione con il client sul protocollo HTTP, si occupa infatti di rispondere alle richieste del client. Per questa funzione deve: decodificare il messaggio XML in arrivo, aggiornare il valore dei controlli utente presenti nel messaggio in arrivo, chiamare la funzione da eseguire presente nel messaggio attraverso l'apposito «Helper» ed infine creare il messaggio XML di risposta, leggendo i dati da inserirvi dai controlli presenti nella «xmlView».

webManeger: Questa classe sostituisce la classe WinFormManager presente nel diagramma del capitolo precedente, la sua funzione è invariata, è stata rinominata per distinguerla da quella presente nel capitolo precedente.

5.2 Descrizione comportamentale lato client

L'elaborazione dei dati in arrivo, la loro visualizzazione e la creazione del documento XML da inviare al server ERP, a seguito di un evento da parte dell'utente, sono compiti svolti da uno script lato client. Lo script è sviluppato in linguaggio Javascript, si sfrutta la tecnologia AJAX per la comunicazione con il server ERP dei soli dati e dei comandi impartiti al gestionale.

Lo script lato client ha quindi solamente il compito di effettuare una richiesta al server e di aggiornare la vista con i dati in arrivo da esso; per questo motivo il codice dello script ha una struttura molto semplice, si è infatti preferito lasciare quanta più logica possibile lato server. L'aggiornamento dell'interfaccia avviene solamente a seguito di un evento scatenato dall'utente; anche grazie a questo fatto, il funzionamento del sistema risulta semplice. I passi svolti dallo script client sono i seguenti, come illustrato nel diagramma di interazione:



- 2: L'utente apre la pagina web.
- 3:Lo script lato client invia un messaggio vuoto; tale messaggio viene interpretato dal server come prima apertura della schermata eseguendo quindi l'evento corrispondente.
- 4: Dalla connessione instaurata al punto 3, arriva in risposta un documento XML
- 5: Lo script provvede ad analizzarlo leggendo le proprietà e il valore dei controlli per aggiornare la schermata.
- 7: L'utente esegue un'operazione che scatena un evento
- 8: Lo script intercetta tale azione e provvede ad aggiornare il documento XML che rappresenta il messaggio. Per eseguire tale operazione aggiorna il contenuto informativo del precedente messaggio ricevuto; l'operazione è inversa a quella illustrata al punto 5.
- 9: Terminato l'aggiornamento del documento XML con il valore dei controlli, a tale documento viene aggiunto un nodo che rappresenta l'evento scatenante con la specificazione del controllo chiamante, tipo chiamante e tipo di evento.

- 10: Si invia il messaggio XML in una nuova connessione, tale documento è necessario all'oggetto webService per richiamare l'apposito «Helper», come illustrato nei capitoli precedenti.
- 11: Dalla connessione instaurata al punto 10 arriva il documento XML di risposta.
- 12: Lo script aggiorna la schermata analogamente al punto 5.

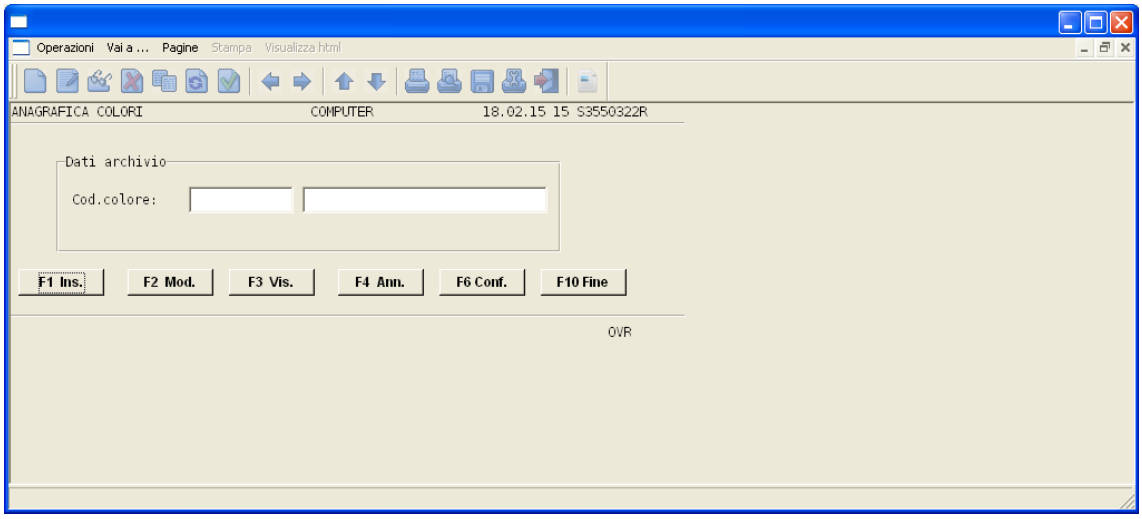
6 Conclusioni

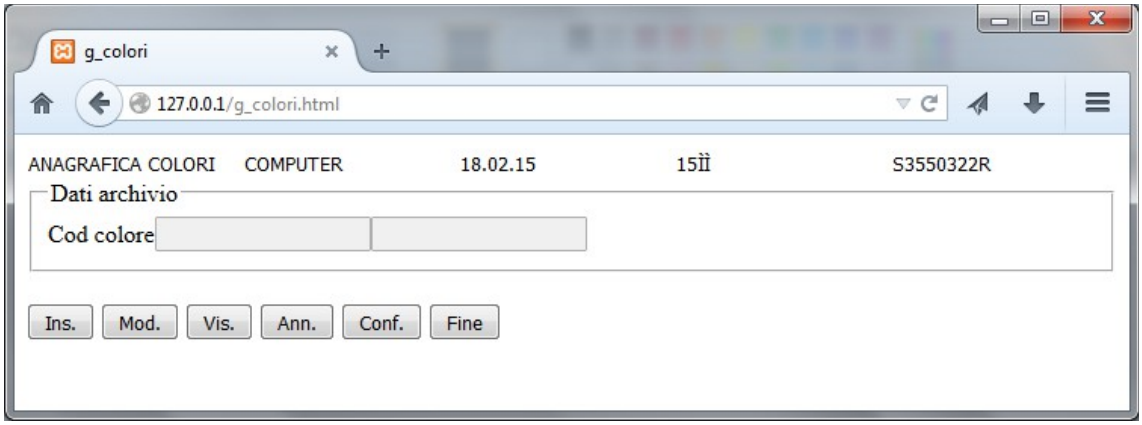
Il lavoro illustrato in queste pagine è un esempio concreto delle soluzioni che si adottano per mantenere compatibilità con i vecchi sistemi legacy.

Per quanto riguarda la conversione delle schermate in Windows Form, sviluppata prima del mio arrivo in azienda, e da me solo studiata e migliorata, come illustrato nel capitolo 4, risulta all'atto pratico completa, tranne per qualche piccola difformità rispetto all'interfaccia grafica originaria.

Per quanto riguarda la conversione a Web, benché sia stata implementata la giusta struttura per la soluzione, sono presenti ancora molti aspetti che non sono ancora stati trattati, la cui risoluzione richiede anche competenze di Web Designing. In particolare è necessario realizzare quei controlli utente lato Web che non sono presenti in HTML5 ma sono presenti nell'interfaccia grafica originaria, come ad esempio le tabelle. È inoltre necessario realizzare un sistema di lancio dei moduli del gestionale da interfaccia web, creando anche un menù di scelta dei moduli del gestionale. Inoltre sarà necessario studiare gli effetti di accessi simultanei alla web application che si desidera creare: ogni modulo del gestionale impiega una porta TCP sulla quale mette in ascolto il proprio web service, ciò renderà necessaria la realizzazione di un sistema di assegnazione delle porte TCP ai vari moduli del gestionale ai quali si accede contemporaneamente.

Vengono ora riportati alcuni screenshot che illustrano i risultati ottenuti.





Nella prima immagine si può vedere un modulo del gestionale con l'interfaccia grafica originaria, la seconda immagine rappresenta lo stesso modulo del gestionale con un interfaccia web HTML5, la terza immagine è l'interfaccia in Windows Form.

Si può notare la struttura simile delle tre schermate e dati analoghi nella barra di intestazione.

Bibliografia-Sitografia

- JYACC Inc, *Application Development Guide,* New York, agosto 1995

- Microsoft Inc, *MSDN Classe Xdocument*,
  https://msdn.microsoft.com/it-it/library/system.xml.linq.xdocument%28v=vs.110%29.aspx

- Microsoft Inc, *MSDN* , Classe HttpListener
  https://msdn.microsoft.com/it-it/library/system.net.httplistener%28v=vs.110%29.aspx

- Mozilla, *Mozilla Developer Network Cross-site HTTP requests*
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS