Università degli Studi di Padova

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Studi in Informatica

Laurea Triennale

Gestione software di accesso concorrente in ambiente di sviluppo dichiarativo: progettazione e realizzazione

Relatore:

Prof. Tullio Vardanega

Candidato:

Marco Pastorio

Anno Accademico 2008/2009 – Sessione I

Indice generale

Introduzione

Questo documento rappresenta una relazione sull'attività di stage svolta presso l'azienda Soluzioni Software SRL di Padova.

L'attività di stage in oggetto è la seconda parte del progetto per la creazione del software di controllo di accesso condiviso al programma aziendale Egen. Quest'ultimo, realizzato dalla stessa Soluzioni Software, è uno strumento per la generazione di applicazioni a partire da regole di business definite tramite un'interfaccia grafica. La logica così definita può quindi essere esportata come implementazione in diversi linguaggi tramite appositi modelli, che possono essere definiti per ogni linguaggio e per ogni destinazione d'uso della logica descritta. Il modello di sviluppo dichiarativo consiste perciò nel definire le sole regole di business, ovvero “cosa”, mentre si lascia alla generazione automatica l'implementazione vera e propria, vale a dire il “come”. Le regole di business, che rappresentano il codice sorgente, vengono memorizzate in file *XML*. Questi file, oltre ad avere le classiche problematiche di sviluppo condiviso di qualsiasi ambiente di sviluppo, presentano delle ulteriori difficoltà dovute al fatto che ad esempio non esiste per Egen un compilatore a garantire la coerenza del software. Infatti l'utente sviluppatore non compila il codice, ma scrive regole di business che verranno tradotte in codice e quindi compilate.

Le funzionalità del software vanno quindi ampliate in modo da mettere a disposizione dell'utente sviluppatore le classiche operazioni da effettuare in ambiente condiviso, nonché assicurare che alcune componenti critiche del sistema di Egen vengano mantenute coerenti e consistenti. Tali funzioni sono state raccolte sotto il componente d'ora in poi denominato Controller.

Lo scopo del progetto di stage nel suo complesso era quello di sviluppare il Controller, dall'analisi di fattibilità all'implementazione, integrandolo con le funzionalità di Egen. Vista la complessità del progetto ed il tempo previsto per lo svolgimento del lavoro, il progetto è stato diviso in due Stage consecutivi con una parziale sovrapposizione. Questa avrebbe consentito la partecipazione di entrambi gli stagisti alla fase cruciale del progetto, nonché un passaggio di consegna più agevole.

Questa relazione ha l'obiettivo di illustrare i concetti base sulle tecnologie

studiate e utilizzate durante lo Stage, l'evoluzione del progetto nelle sue varie fasi e le problematiche affrontate con le relative soluzioni adottate.

Prima di passare alla descrizione delle attività svolte nella seconda parte dello Stage, verranno descritte la situazione preesistente ed il percorso svolto durante la prima parte del progetto.

Convenzioni tipografiche

Per il presente documento verranno adottate le seguenti convenzioni tipografiche:

- *Corsivo*: termine tecnico
- Sottolineato : termine presente nel Glossario[1]
- *Riferimento*: sezione estratta dalla Definizione di Prodotto
- `Codice`: sezione di codice del software prodotto

1 Verrà sottolineato solo alla prima occorrenza del termine.

1 Presentazione dominio applicativo

Questo capitolo descriverà l'azienda ospitante ed il suo contesto operativo, che ha dato origine al progetto di stage.

1.1 L'azienda

Soluzioni Software SRL è un'azienda che progetta e realizza soluzioni per le PMI, conta più di 60 dipendenti e collaboratori, ha sede principale a Padova, ma comprende anche altre filiali a Milano, Roma e Pescara. L'offerta commerciale si può dividere in tre unità indirizzate ad altrettanti segmenti di mercato:

- **AdApta**, sviluppa e distribuisce prodotti *ERP* per la media impresa operando un approccio del tipo "prodotto + progetto";
- **ONE** dedicato all'implementazione e distribuzione di SAP Business One;
- **iProject**, forti competenze tecniche ed applicative che operano con approccio progettuale nella fornitura di soluzioni innovative e ad alto contenuto di personalizzazione sulle specificità del cliente.

Un altro prodotto aziendale è appunto **Egen**, un generatore di codice, che è il soggetto principale del progetto di Stage. Il progetto è nato dalla necessità di applicare le basi dello sviluppo condiviso allo sviluppo di software basato su Egen; prima di entrare nel dettaglio del progetto si darà una rapida descrizione del problema che è alla base dello stage.

1.2 Problematiche di contesto

Lo Stage si è svolto nel settore dell'azienda che si occupa dello sviluppo del software ed in particolare di soluzioni *ERP*, acronimo di *Enterprise Resource Planning* ("Pianificazione delle Risorse d'Impresa"). Esiste una grande varietà di attività specifiche per cui può essere vantaggioso l'utilizzo di una specifica soluzione *ERP*, come ad esempio controllo di inventari, tracciamento degli ordini, servizi per i clienti, finanza, risorse umane, ecc...

Uno dei problemi cardine della fornitura di sistemi informatici *ERP* è che ogni singola attività richiede una particolare struttura del programma di gestione, ma non solo: ogni particolare situazione aziendale ha requisiti particolari che variano a seconda delle dimensioni dell'azienda, dei settori in cui opera, delle esigenze specifiche di ogni singolo settore e di quali di questi il programma gestionale deve fare interagire, ed anche a seconda del grado di interazione tra i vari settori.

Esistono in commercio molti sistemi *ERP* che gestiscono situazioni più o meno particolari. Tuttavia un'azienda che vuole adottare un sistema *ERP* può incontrare delle difficoltà a trovare un programma che si adatti perfettamente alla propria situazione specifica: si deve adattare a soluzioni parziali del problema, o acquistare dei sistemi più complessi del necessario con maggiori costi e maggiori difficoltà di formazione del personale e di configurazione. Inoltre, ammesso che sia stato trovato il programma adatto, la situazione dell'azienda raramente è statica: nel corso del tempo le esigenze specifiche dell'azienda possono subire profondi cambiamenti, ad esempio in seguito ad un'espansione o ad un cambio di settore del mercato. A questo punto il sistema *ERP* adottato potrebbe risultare inadatto a far fronte alla nuova situazione aziendale. Questo problema diventa particolarmente rilevante per un'azienda in fase di sviluppo, con una situazione aziendale in continua evoluzione.

1.3 Strategia risolutiva

Viste le problematiche di contesto, appare evidente che una soluzione statica del problema, per quanto vasta e completa, non è la soluzione ottimale: infatti un singolo programma di gestione *ERP* sviluppato per comprendere tutte le possibili aree di gestione difficilmente può soddisfare nel dettaglio ogni singola esigenza, e non sarebbe comunque adatto a far fronte ad eventuali sviluppi di ogni area. Inoltre il costo di una soluzione così vasta non sarebbe competitivo con quello di soluzioni sviluppate ad hoc per una determinata area di gestione.Viceversa i programmi statici sviluppati ad hoc si rivelano inadatti qualora la situazione aziendale richieda l'interazione di aree gestionali normalmente distinte. Ad esempio un'azienda che fornisce prodotti e servizi (ad esempio, componenti elettronici e formazione professionale per l'uso dei propri prodotti) potrebbe volere un programma gestionale che sia in grado di far interagire il tracciamento degli ordini con i servizi per i clienti, oltre alle risorse umane dell'azienda impegnate per la parte di servizio al cliente di ogni ordine.

La conseguenza è che spesso le aziende necessitano di soluzioni gestionali specifiche, sviluppate appositamente per far fronte alle necessità particolari dell'azienda. Appare evidente però che un progetto sviluppato da zero per ogni soluzione particolare risulterà sensibilmente più costoso di una soluzione generica. Anche partendo da una base di partenza collaudata che preveda le funzioni comuni alle varie aree gestionali, o addirittura partendo da programmi precedentemente sviluppati per soluzioni simili, le modifiche alle funzionalità previste e le funzionalità aggiuntive richiedono modifiche ed aggiunte al codice onerose per mole di lavoro, anche qualora fossero differenze solo marginali rispetto ai progetti precedenti. Inoltre le successive modifiche nel tempo rischiano di minare la manutenibilità del software, e facilitano l'introduzione di errori logici e conflitti tra le varie parti del software.

Le problematiche suddette si acuiscono ulteriormente se si considera che le tecnologie informatiche sono varie e si evolvono rapidamente nel tempo. In ogni periodo convivono diversi linguaggi di programmazione che possono essere richiesti per necessità di contesto delle aziende clienti (ad esempio per facilitare l'integrazione o garantire la compatibilità con programmi in possesso dell'azienda, per interoperabilità). Anche la sola operazione di *porting* di un programma esistente ad un diverso linguaggio è estremamente onerosa in termini di lavoro.

1.3.1 La soluzione adottata

Per far fronte alle problematiche appena esposte l'azienda creato un software di sviluppo che semplifichi il riutilizzo del codice, l'aggiunta o la modifica di funzionalità preesistenti, il *porting* di un programma tra diversi linguaggi e la conservazione della manutenibilità del software.

2 Il progetto aziendale

Questo capitolo presenta il progetto sviluppato dall'azienda per far fronte alle problematiche di contesto sopra descritte, e che ha generato il progetto di Stage a cui si riferisce questa relazione.

2.1 Egen

Egen è il programma sulla cui espansione si basa il progetto di Stage: in questa sezione verrà presentato il programma e si mostrerà come questo soddisfi i requisiti di contesto dell'azienda.

L'esposizione della struttura e del funzionamento del programma semplificherà la comprensione delle problematiche incontrate nel progetto di stage e quindi delle soluzioni adottate.

2.1.1 Descrizione generale

Egen è un generatore di applicazioni software in ambito gestionale, il progetto che lo sviluppa ha come obiettivi:

- automazione nello sviluppo di sistemi informatici "gestionali" transazionali basati su database relazionali;
- definizione delle specifiche applicative attraverso la definizione di un apposito sistema di regole espresse in modo dichiarativo e/o tramite script controllabili e manipolabili dall'utente;
- generazione automatica del codice applicativo a partire dalle regole di cui sopra mediante l'utilizzo di opportuni *template*[1] ;
- utilizzo di modelli di generazione multi-language, capaci di generare codice sorgente in svariati linguaggi di programmazione e di aderire eventuali altre caratteristiche desiderate.

1. I *template* sono modelli che contengono porzioni di codice sorgente del linguaggio target, gestiti con un linguaggio proprietario di lettura ed interpretazione delle regole definite nel *Repository*.

Un sistema con queste caratteristiche otterrebbe di conseguenza alcuni benefici:

- maggiore produttività (il codice viene generato automaticamente ed una regola contenuta in una sola riga di testo sostituisce decine o centinaia di righe di codice);

- maggiore qualità (ancora: il codice viene generato automaticamente, esonerando lo sviluppatore dal ricordare quando una certa regola di programmazione deve scattare ed in quale sequenza all'interno di una particolare transazione);

- migliore manutenzione applicativa (poiché essa si sposta al livello delle regole e non più del codice scritto manualmente, e quindi per definizione auto-documentante e slegato dallo sviluppatore);

- immediato adeguamento alle novità tecnologiche, l'aggiornamento periodico dei modelli consente la rigenerazione totale del codice con nuove caratteristiche applicative (per esempio una nuova gestione degli accessi) o tecniche (per esempio un'interfaccia verso un nuovo DBMS).

Egen, quindi, consiste in un applicativo dal quale è possibile dichiarare entità e relazioni, specificare tramite opzioni come queste relazioni si devono comportare in casi specifici, scegliere tra varie possibilità la forma dell'applicazione che si sta generando ed anche il linguaggio nel quale verrà generata.

Tutto questo è possibile grazie alla programmazione dichiarativa e all'architettura di Egen.

Programmazione dichiarativa

La programmazione dichiarativa consiste nel definire affermazioni che devono essere considerate vere o che devono essere rese vere, descrivendo cosa rappresenta un determinato programma e non come lo deve rappresentare.

La generazione di codice realizzata da Egen è basata sulla programmazione dichiarativa, la quale consiste nel dichiarare oggetti e regole che descrivono gli oggetti stessi e le relazioni che sussistono tra di essi senza scrivere una sola riga di codice, limitando così al minimo gli errori che normalmente vengono inseriti in fase di codifica. Queste regole e relazioni sono quindi la parte fondamentale di

un *Repository* di Egen, e vengono chiamate *business rules.*

Egen prevede l'uso di *template* che "pilotano" la generazione del codice finale partendo da modelli base configurabili. Attraverso il *template* per un determinato linguaggio di programmazione è possibile descrivere come una determinata entità o relazione deve essere rappresentata in quel linguaggio di programmazione.

L'architettura generale di Egen

In figura 1 si può vedere lo schema a blocchi del sistema:

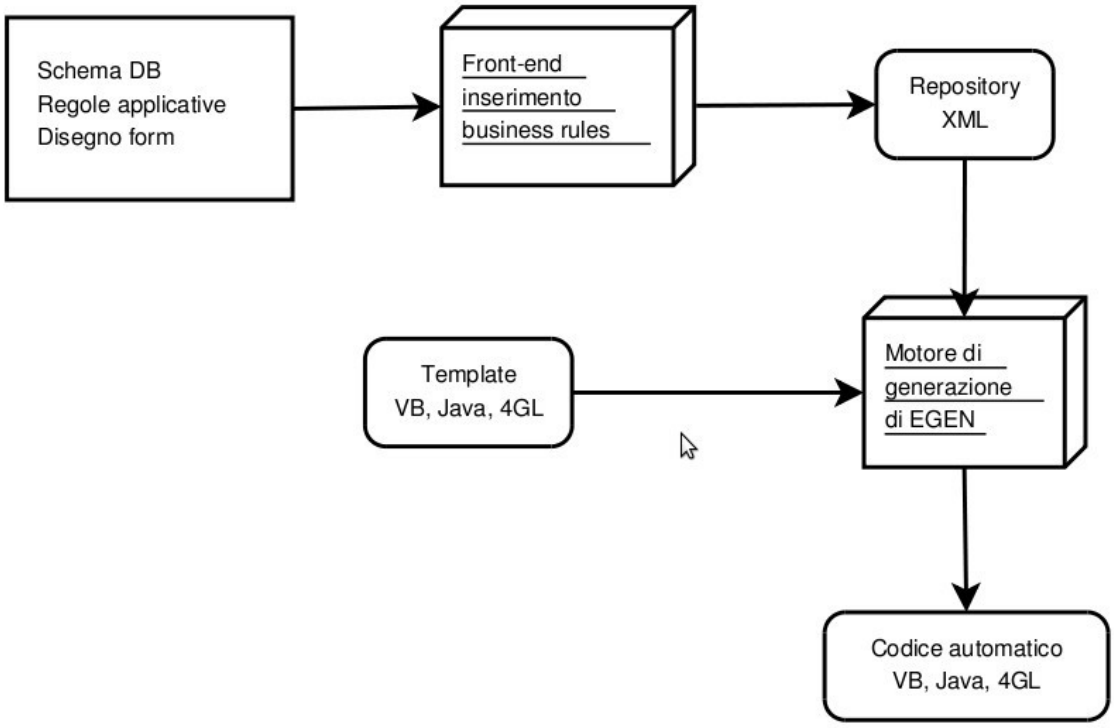

Figura 1: L'architettura di Egen.

Lo sviluppatore inserisce lo schema del database, alcune regole applicative e può decidere come devono essere visualizzate le *form*[1] del prodotto finale, attraverso un strumento messo a disposizione da Egen.

Egen successivamente elabora le informazioni inserite e crea un *Repository XML*, attualmente esteso per permettere la gestione della condivisione di regole su più livelli paralleli.

Da questo *Repository*, attraverso l'uso di un *template* relativo ad uno specifico

1 Le *form* sono le rappresentazioni grafiche degli oggetti del *Repository.*

linguaggio di programmazione viene generato il codice finale che, una volta compilato (o interpretato) esegue l'applicativo finale.

2.1.2 Come Egen risponde alle problematiche di contesto

Come si evince dalla descrizione del progetto, Egen è un programma particolarmente adatto al dominio operativo di un'azienda che sviluppi soluzioni *ERP*.

Sfruttare uno strumento che permetta di utilizzare la programmazione dichiarativa e che a partire dalle *business rules* inserite possa produrre il relativo codice in diversi linguaggi di programmazione comporta un considerevole risparmio in termini di tempo di progettazione, codifica e manutenzione. Questo implica una maggiore competitività dei programmi sviluppati tramite Egen, che hanno un costo minore rispetto ad un prodotto sviluppato con un ciclo di vita dedicato.

Inoltre anche in caso fosse richiesta una pesante modifica di un programma precedentemente sviluppato tramite Egen sarebbe sufficiente modificare le regole precedentemente definite, senza toccare il codice del programma e quindi riducendo il tempo ed il pericolo di errori.

2.1.3 La struttura interna di Egen

La definizione delle *business rules* tramite l'interfaccia grafica di Egen avviene concretamente con la creazione di file *XML* in un *Repository* di lavoro condiviso ai membri di un progetto. Un *Repository* Egen dopo la creazione è composto da:

- **LLObj**: Rappresenta la parte logica dell'applicazione e contiene tutti i file *XML* relativi agli oggetti logici creati in Egen ed elencati nell'indice;
- **PRES**: Rappresenta la componente di presentazione dell'applicazione e contiene tutti i file *XML* relativi alla rappresentazione grafica in *form* degli oggetti logici;
- **Impact.xml**: contiene tutte le dipendenze che legano gli oggetti definiti nell'indice, sia a livello logico che di presentazione;
- **NomeRepository.egpj**: ovvero l'indice dell'intero *Repository* sulla cui

lettura si basa la visualizzazione degli oggetti in Egen ed in cui tutti gli oggetti esistenti sono indicizzati, raccolti nelle seguenti tipologie:

- Data Objects (DO);
- Reps contenente le relazioni che intercorrono fra gli oggetti nella forma “R2_nomepadre-nomefiglio”;
- Query objects (QO);
- (Groups);

Per il corretto funzionamento di Egen sono di fondamentale importanza il file .egpj, definito come indice, ed il file Impact.xml.

A livello aziendale esistevano due diversi situazioni legate alla gestione dei *Repository* Egen. La maggioranza degli utenti utilizzava Egen v1, che non utilizza il file di Impact e si affida in termini di compilazione ad un compilatore *Java* proprietario; per la gestione del *Repository* di progetto esistono tanti *Repository* locali quanti sono i client Egen esistenti e la condivisione del lavoro di sviluppo avviene al termine di ogni giornata lavorativa tramite l’esecuzione automatica di un servizio di *merging*[1] che si occupa di aggiornare il *Repository* generale remoto unendo le modifiche effettuate. Pertanto sono presenti le problematiche tipiche relative ad ambienti di sviluppo non condivisi come lunghi intervalli di inconsistenza del *Repository*, rischi di sovrascrittura e di perdita di dati, rischi di *merging* errati, lunghi tempi di aggiornamento.

Un numero inferiore di utenti utilizzava invece Egen v2, che chiameremo semplicemente Egen, che non è vincolato ad alcun compilatore, e permette l’esportazione della logica tramite *template* di output; visto il numero limitato di utenti non esiste di fatto un gestione automatizzata di condivisione delle modifiche, ma questa risulterà necessaria nel momento in cui la nuova versione di Egen sarà rilasciata.

.

1 Per *merging* si intende l'accorpamento delle modifiche apportate a due *Repository* locali nel *Repository* generale.

Indice

Scritto in *XML*, rappresenta tramite appositi < *tag* > l'elenco degli oggetti presenti nel *Repository* di un progetto. Per ogni oggetto vengono indicati il campo "Nome" o "File" che identificano di fatto il percorso sul file system in cui è archiviato il file *XML* relativo all'oggetto; a seconda del tipo di oggetto i percorsi saranno NomeRepository/LLOBJ, per DataObjects, Rels e QueryObjects o NomeRepository/PRES per le Applications. La lettura del file di indice in Egen è il punto di partenza per la rappresentazione grafica dell'albero degli oggetti esistenti.

Impact

Scritto in *XML*, il file di Impact rappresenta una delle principali novità di Egen e permette l'inserimento e la rilevazione delle dipendenze logiche che intercorrono tra gli oggetti del progetto dal momento stesso della definizione della logica dell'applicazione. Questo assicura la creazione di una logica priva di errori nelle dipendenze e permette quindi una successiva corretta interpretazione al momento della compilazione da parte dei *template* di output. Nel file il nome di un oggetto viene usato come ID per indicare le dipendenze relative a quell'oggetto; la struttura del file è ricorsiva, ovvero, la presenza di una dipendenza tra due oggetti A e C, connessi da B, viene indicata come dipendenze separate A-B e B-C, pertanto, la rilevazione delle dipendenze di A avviene con una lettura delle dipendenze legate ad A e di quelle connesse agli oggetti dipendenti di A.

$$\mathbf{Dip(A) = Dip(A) + Dip(Dip(A))}$$

Gestione indice e file di Impact

Uno degli aspetti critici da considerare sarà pertanto la risoluzione delle problematiche di lettura ed aggiornamento di questi due elementi, essenziali per un corretto funzionamento di Egen.

La gestione di queste operazioni era fatta localmente da Egen v1 e poi risolta giornalmente tramite *merge* delle varie versioni dei client; all'interno di una ambiente di lavoro condiviso sarà invece necessario mettere sempre a disposizione dei client Egen la versione aggiornata di entrambi i file, garantendo così la consistenza generale dei dati.

2.1.4 Problematiche relative allo sviluppo condiviso

Le funzionalità di Egen lo rendono particolarmente efficace per lo sviluppo di applicazioni *ERP*, ma non essendo progettato dall'inizio per lo sviluppo condiviso presenta anche delle difficoltà di utilizzo che ne limitano il rendimento. Perché le potenzialità del programma siano sfruttate a pieno era quindi importante che venisse aggiunto ad Egen un componente che potesse garantire l'accesso condiviso al *Repository* mantenendo costantemente aggiornati tutti i file locali, propagando le modifiche sensibili alle macchine di tutti gli sviluppatori. In generale un software che fornisca supporto per lo sviluppo condiviso deve poter gestire le seguenti problematiche:

- ogni file può essere modificato da un solo utente per volta e deve essere possibile sapere quale utente sta modificando un dato file;
- deve essere possibile risalire all'autore delle modifiche su di un file, e devono poter essere ripristinate le versioni precedenti.

Source Safe è un programma che fornisce le funzionalità adatte alla gestione delle suddette problematiche, consentendo il versionamento dei file ed esponendo le funzioni per l'accesso mutuamente esclusivo ai file tramite queste azioni fondamentali:

Checkout (o ESTRAZIONE di un file): eseguire l'estrazione di un file significa prenotarlo (effettuare un *lock* sul file) in modo che l'utente che ha il file in estrazione sia l'unico che abbia i diritti per la modifica del file.

Checkin (o ARCHIVIAZIONE di un file): eseguire l'archiviazione di un file significa salvare le eventuali modifiche sul file e rilasciare il *lock* su di esso, in modo che altri utenti possano eventualmente effettuarne l'estrazione.

UndoCheckout (o RILASCIO di un file): eseguire il rilascio di un file significa scartare le eventuali modifiche effettuate e rilasciare il *lock* sul file.

Come si è visto nella sottosezione precedente, la struttura interna di Egen è piuttosto complessa. Per creare un ambiente di sviluppo condiviso efficace e fruibile non è sufficiente che la politica d'accesso condiviso sia effettuata sui singoli file: per le parti critiche di Egen, in particolare Indice e Impact, serve una propagazione più capillare delle modifiche effettuate.

Vista la struttura di Egen, una modifica ad un oggetto del *Repository* può coinvolgere a catena diversi altri oggetti collegati al primo tramite relazioni di vario genere. Questa situazione complica notevolmente la gestione dell'accesso condiviso in quanto ogni tipo di operazione che può essere svolta su un oggetto va trattata in modo opportuno a seconda delle modifiche che vengono fatte non solo all'oggetto stesso, ma potenzialmente agli altri oggetti a cui questo è collegato ed anche al *Repository*.

Nella successiva sezione verranno presentate le problematiche particolari risultanti dall'analisi svolta nella prima parte dello stage da Giulio Favotto, e nel terzo capitolo verrà mostrato come queste sono state affrontate, e quali conseguenze ha avuto la complessità di Egen sul progetto del Controller.

2.2 Vincoli e Obiettivi

Questa sezione descrive i vincoli del progetto relativi all'ambiente di sviluppo, derivanti da esigenze aziendali e/o da scelte progettuali effettuate nella prima parte dello stage.

2.2.1 Ambiente di sviluppo

Ambiente di programmazione

Il Controller sarà un modulo software il cui utilizzo finale avverrà all'interno di una rete di PC Microsoft su sistema operativo Windows XP. L'ambiente di esecuzione sarà quindi un intranet aziendale.

Il software di sviluppo aziendale è Microsoft Visual Studio 2005. Il linguaggio richiesto è VB.NET in quanto, sebbene Egen sia scritto in Visual Basic 6.0 che è non può leggere funzioni scritte in VB.NET senza una opportuna conversione, è in programma la conversione di Egen in linguaggio VB.NET. Creare il Controller in linguaggio VB.NET semplificherà quindi l'adattamento della prossima versione di Egen al software di controllo creato per l'attuale versione.

L'utilizzo di Visual Studio comporta anche alcuni vantaggi dal punto di vista della verifica del software, di cui si tratterà nell'apposita sezione.

Microsoft Visual Source Safe v8

Microsoft Visual Source Safe è un prodotto per il controllo delle versioni a livello di file che consente a svariati tipi di organizzazioni di utilizzare più versioni di un progetto contemporaneamente. Questa funzionalità risulta particolarmente utile negli ambienti di sviluppo software, in cui viene sfruttata per gestire versioni di codice parallele.

Source Safe è il programma che veniva già utilizzato per il versionamento e l'archiviazione dei file di Egen, ma tutte le operazioni erano eseguite manualmente dagli utenti. È invece possibile permettere ad un programma di interagire direttamente con Source Safe grazie alle librerie di interazione.

Web Service

Il paradigma computazionale che pone i servizi come elementi alla base dello sviluppo delle applicazioni si definisce *Service Oriented Computing*. In questo contesto, per servizi si intendono dei componenti aperti, autodescrittivi, indipendenti e accessibili tramite un'interfaccia la cui natura sarà definita dal particolare genere e tipo di servizio.

Questo modello generale si poggia su una struttura chiamata *Service Oriented Architecture*, che viene descritta nella figura 2.

Il servizio è incapsulato dietro all'interfaccia, quindi la sua implementazione è invisibile all'esterno. Le sue funzionalità sono note al cliente solo tramite l'interfaccia pubblica che rende disponibile la descrizione dei possibili modi di comunicazione accettabili dal servizio. Al cliente vengono date le sole informazioni di cui necessita per "dialogare" con il servizio, altre informazioni riguardo all'implementazione sono tenute nascoste.

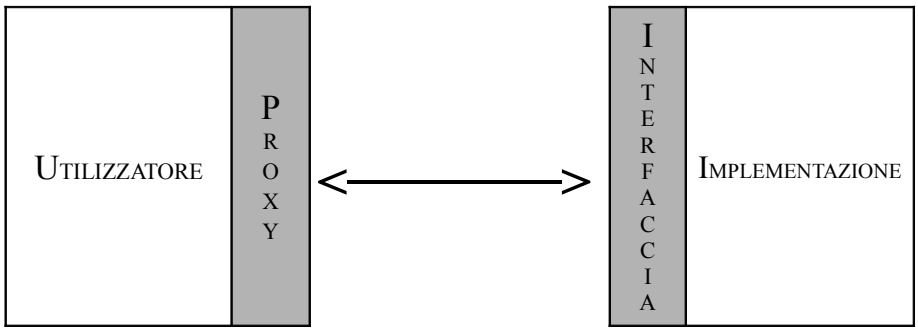

Figura 2: Service Oriented Architecture

Cliente e fornitore sono legati da un contratto che consiste nel formato dell'interfaccia, nei protocolli di comunicazione, nel formato dei messaggi e nelle modalità d'interazione che garantiscono che il servizio risponda con il comportamento desiderato.

Fino a qualche anno fa lo sviluppo del software è stato legato al tradizionale processo, codifica-compilazione-distribuzione-esecuzione. Lo sviluppo della rete, spinto dall'esplosione del Web, ha portato a una modifica di questo processo che però è ancora fortemente vincolato da diversi fattori: da un lato le applicazioni basate sugli strati bassi dei protocolli di rete sono rimaste afflitte da

problemi di interoperabilità tra diverse piattaforme e dalle difficoltà di sviluppo; dall'altro il Web che è riuscito a superare questi problemi ma è rimasto limitato dalla sua originaria natura di presentazione di documenti, non *general purpose* vincolato all'idea del browser.

I Web Service offrono oggi le possibilità per un nuovo processo di sviluppo dei servizi, mettendoli a disposizione in rete svincolando la logica applicativa dal lato di presentazione, rendendoli quindi più facilmente riutilizzabili rispetto alle applicazioni Web, permettendo comunicazione macchina-macchina ma mantenendo garanzia di interoperabilità. Il tutto basato su tecnologie già largamente diffuse come il protocollo *HTTP* (ma non vincolato ad esso) e la forza dell'estensibilità dei linguaggi derivati da *XML*.

Già al giorno d'oggi sono disponibili una buona quantità di strumenti che rendono la creazione di Web Service molto semplici per gli sviluppatori a partire non solo da piattaforme come *.NET* ma anche da linguaggi alternativi. Inoltre già in Visual Studio 2005 per il linguaggio VB.NET sono rese disponibili dei metodi per implementare i Web Service integrandoli con il codice dell'applicazione, i cui dettagli verranno illustrati nella sezione relativa allo svolgimento del progetto.

A seguito verranno presentati i vincoli del Controller risultanti dalla prima parte del progetto di Stage, svolto da Giulio Favotto.

2.2.2 Prima parte del progetto di stage

Interfacciamento con l'ambiente di lavoro

Il Controller dovrà fungere da interfaccia fra il software Egen installato nei client e lo strumento di archiviazione-versionamento adottato, MS Visual Source Safe, presente nel server; particolare attenzione sarà quindi riposta nella sua corretta integrazione in questa realtà; nel dettaglio:

- **Controller-Egen**: Egen mette già a disposizione un modulo per l'inserimento di possibili estensioni, di interfacciamento con l'ambiente di lavoro identificate come User Tools, di pulsanti e nuove voci dei menu senza dover accedere al codice sorgente del programma ma semplicemente effettuando degli inserimenti tramite la *GUI* a disposizione.

- **Controller-Source Safe**: Visual Source Safe v8 prevede un insieme di interfacce di automazione basate su COM fornite dal modello di automazione di Visual Source Safe (IVSS); Il modello IVSS fornisce funzionalità sia per l'automazione delle interazioni con un database di Visual Source Safe sia per la risposta agli eventi del database; l'automazione delle interazioni con un database di Visual Source Safe viene eseguita mediante il modello di automazione OLE IVSSDatabase. I dettagli implementativi vengono ampiamente descritti nella guida di Visual Source Safe, sezione "Modello di automazione di Visual Source Safe", con particolare riferimento ad implementazioni C# e VB;

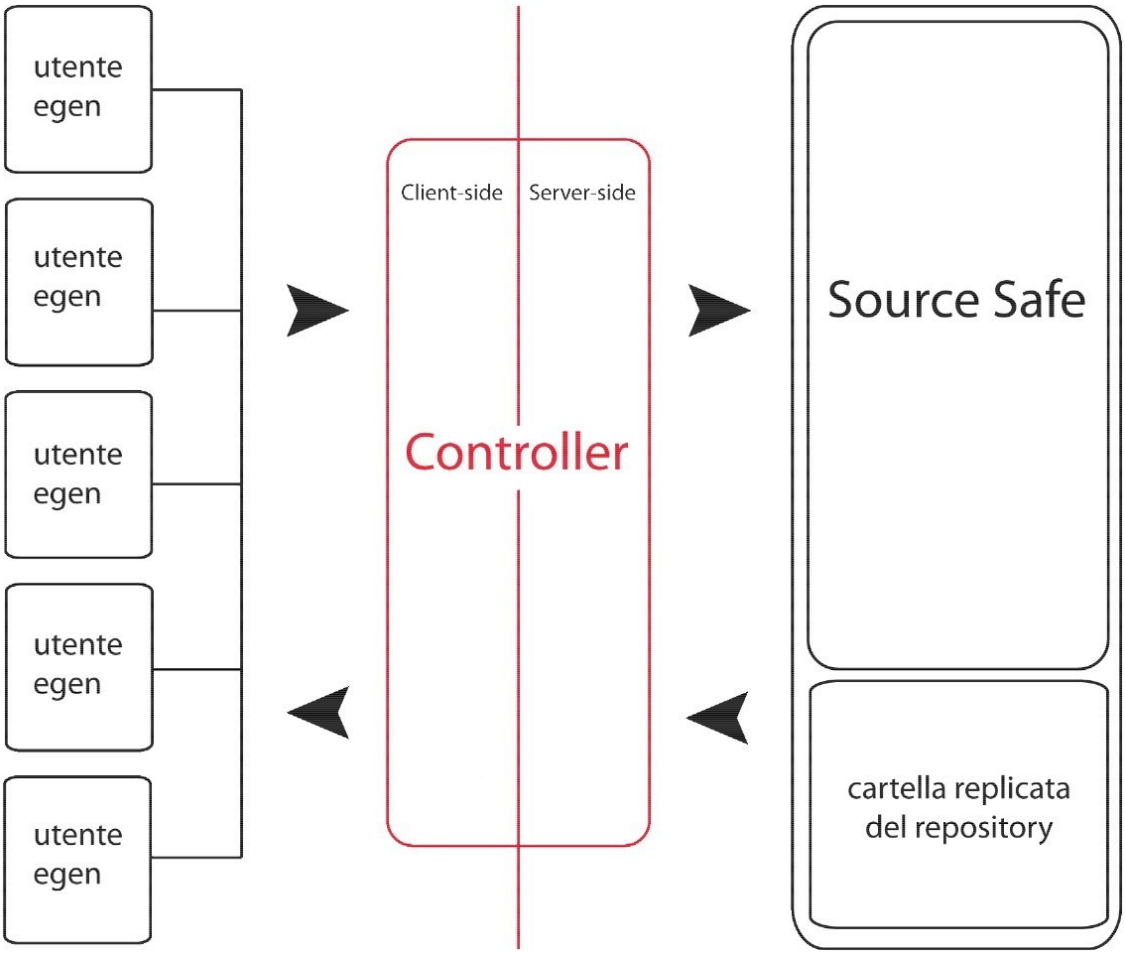

Figura 3: Integrazione del componente nel sistema.

Il Controller dovrà essere composto da due sotto-componenti che andranno a collocarsi in due ambiente differenti:

- un sotto-componente **server-side** installato come servizio nel server centrale che andrà ad estendere l'interfaccia fornita da Visual Source Safe v8 per la gestione del *Repository* generale;

- un sotto-componente **client-side** presente in ogni installazione Egen sotto forma di User Tool che si occuperà di dialogare con il componente sever-side.

In Figura 3 è rappresentata la collocazione logica del Controller, tra i vari client Egen e Source Safe; sfruttando uno strumento di Source Safe v8 si farà anche riferimento ad una cartella replicata del *Repository* di progetto disponibile in modalità di sola lettura. Tale struttura non sarà da intendere come unica nella rete aziendale; potranno infatti esistere diversi *Repository* dislocati in più server. Analizzata singolarmente però la struttura resterà la medesima, ovvero, in ogni server il Controller server-side dovrà gestire esclusivamente i *Repository* controllati da Source Safe v8 in quella stessa macchina.

Modalità d'uso del prodotto

A realizzazione ultimata il prodotto dovrà quindi soddisfare le esigenze espresse e permettere all'utente di Egen di sviluppare software in un ambiente di lavoro condiviso, sollevandolo però da impegni di gestione del *Repository* di progetto e garantendo le caratteristiche di consistenza logica del codice sviluppato.

Funzioni del prodotto

Il Controller client-side dovrà mettere a disposizione di Egen le seguenti tipologie di azioni base:

- **Read**: la lettura di un file locale o dell'ultima versione di uno o più file contenuti nel *Repository* (corrispondente al "get latest version" usuale);

- **GetLatestVersion**: il download nella cartella locale dell'ultima versione disponibile di uno o più file;

- **CheckIn**: l'archiviazione di uno o più file modificati nel *Repository*;
- **CheckOut**: l'estrazione di uno o più file dal *Repository* per modificarli;
- **UndoCheckOut**: l'annullamento dell'ultima estrazione effettuata;
- **Add**: aggiunta di un file creato localmente al *Repository* remoto;
- **Delete**: rimozione di un file dal *Repository* remoto.

Queste azioni base potranno diventare azioni più complesse in presenza di dipendenze. Dovrà infine consentire la realizzazione di verticalizzazioni personalizzate a partire dal *Repository* comune.

Il Controller server-side dovrà svolgere le seguenti funzioni:

- Inoltrare le richieste di **azioni base** provenienti dai client verso Source Safe.
- Gestire le **azioni complesse** effettuate dai client (azioni in cui sono coinvolte modifiche di più file in dipendenza) verificando possibili dipendenze segnalate dal client e risolvendole adeguatamente.
- Gestire l'**aggiornamento** del file di Indice e di Impact e la propagazione degli aggiornamenti ai client.
- Trasmettere le comunicazioni sugli stati dei file e su possibili errori o incongruenze.
- Fornire un interfaccia per la realizzazione di **verticalizzazioni**.

Caratteristiche degli utenti

Collocandosi all'interno di un ambiente operativo aziendale il componente non dovrà interagire con moduli di autenticazione dell'utente, si assume pertanto l'uso del componente sarà destinato solamente ad utenti autenticati.

Utenti previsti:

1. Utente Egen;

2. Amministratore.

Essendo l'utilizzo del componente strettamente vincolato all'uso di Egen, il principale utente sarà l'**utente Egen** inteso come utente che utilizza Egen in modalità client; un utente **amministratore** in questa prima versione del Controller è da considerarsi opzionale, ma potrebbe essere implementato per permettere operazioni di editing dei permessi utente, lock o unlock manuale di file, configurazione manuale delle verticalizzazioni... funzioni attualmente non richieste.

Architettura generale

In base agli elementi emersi in fase di analisi, Il Controller andrebbe a configurarsi come un componente distribuito caratterizzato quindi da un'architettura di base di tipo Client-Server. Le esigenze ed il contesto del componente rispecchiano quelle definite nel pattern architetturale *Three-tier distribution architecture* :

- front-end clients,
- domain servers (also known as application servers),
- and a storage and DB-server.

L'applicazione di tale pattern alla realtà descritta nell'analisi dei requisiti è visibile in Figura 4 .

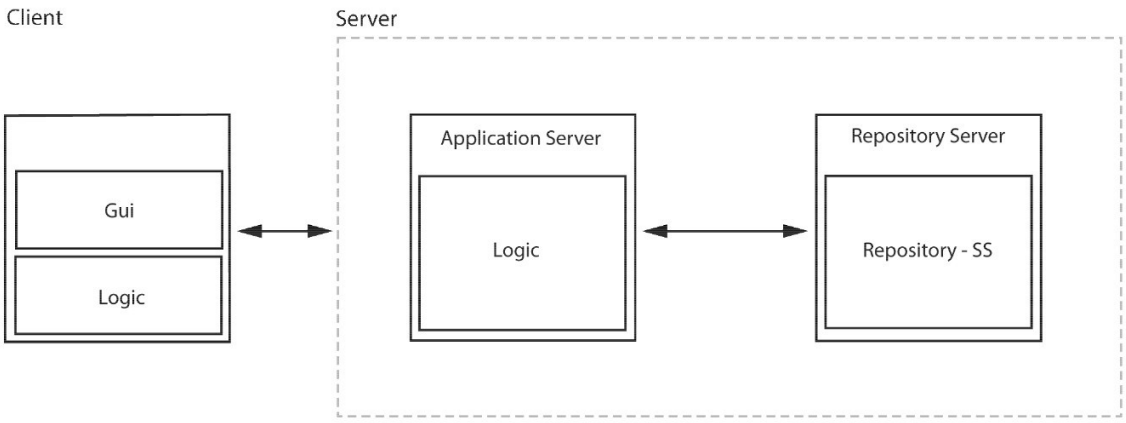

Figura 4: Architettura del componente.

Il sistema risulta di fatto composto da tre livelli:

- un livello **Client**, distribuito nei computer degli utenti Egen, caratterizzato da una sotto-componente di interfaccia grafica (***GUI***, per la gran parte integrata nella *GUI* di Egen) e da una sotto-componente logica locale (**Logic**);

- un macro-livello **Server** che raccoglie altri due livelli:

  - **Application Server**, livello contenente la logica del componente lato server, compresa l'interfaccia disponibile ai vari Client;

  - **Repository Server**, livello che connette l'Application Server al *Repository* in cui sono contenuti tutti i file di progetto (ed in particolare i file relativi agli indici e alle dipendenze, l'importanza dei quali è descritta nell'analisi dei requisiti) tramite l'apposita interfaccia che Source Safe mette a disposizione.

Nella realtà in esame i livelli di Application Server e Repository Server si troveranno fisicamente sulla stessa macchina, e l'Application Server non dovrà preoccuparsi di gestire le azioni base di dialogo con il databse del *Repository* ma utilizzerà l'interfaccia VB.Net messa a disposizione dalla libreria *Microsoft.VisualStudio.SourceSafe.Interop* di Source Safe.

Componente Client

Il componente Client del Controller dovrà essere un modulo software sviluppato in VB.Net, distribuito nei vari client Egen, la cui *GUI* sarà quasi interamente integrata in quella dei client; in fase di analisi la modalità di integrazione più opportuna era sembrata quella che utilizzava le "User Tools" messe a disposizione da Egen; è infatti possibile inserire comandi personalizzabili resi poi accessibili da appositi menu e pulsanti già predisposti, senza intervenire direttamente sul codice sorgente del programma. Già in fase di specifica però, tale modalità ha presentato alcuni limiti, il principale è che l'utilizzo degli User Tools è vincolato ad un'apposita chiamata fatta dall'utente e non sarebbe pertanto possibile avviare automaticamente in background altre azioni. Un adattamento del codice sorgente di Egen sarebbe pertanto preferibile. La componente grafica extra del Client sarà piuttosto minimale e potrà eventualmente consistere in *form* a pop-up per richieste o brevi messaggi.

L'elemento principale del Client sarà pertanto la componente logica che dovrà garantire l'esecuzione corretta e sicura delle esigenze manifestate nell'analisi, implementando:

- le azioni di Add, Delete, Read, Checkout, Checkin, UndoCheckout, GetLatestVersion;
- le operazioni di aggiornamento dell'indice dei *Repository* in uso e dei file delle dipendenze;
- un sistema di comunicazione e trasferimento file client-server con notifiche di errori, stati ed aggiornamenti.

In alcune circostanze in cui potrebbe risultare conveniente il Client potrebbe aprire una connessione diretta al *Repository*, evitando cosi di aggiungere lavoro inutile al Controller lato Server e sfruttando quindi le *DLL* che Source Safe mette a disposizione per l'effettivo trasferimento dei file al *Repository*. In alternativa si potrebbe implementare i Controller lato client e lato server in modo effettuino il trasferimento file tra loro e rimettere poi al Controller lato server la responsabilità di tutti gli accessi al *Repository*. Questo potrebbe però richiedere la scrittura di codice extra per gestire correttamente l'uso di protocolli di trasferimento.

Componente Server

Il componente Server del Controller dovrà essere un modulo software sviluppato in VB.Net attivo nel Server ospitante il *Repository* dei progetti Egen. La componente logica sarà divisa in più sotto-componenti:

- Application: implementerà la logica vera e propria del Controller lato server, mettendo a disposizione dei client l'interfaccia necessaria all'esecuzione dell'azioni richieste, facendo quindi da mediatore delle richieste al repository Source Safe;

- Repository: gestirà il *Repository* utilizzando la *DLL Microsoft.VisualStudio.SourceSafe.Interop* di Source Safe.

Dal punto di vista implementativo possono essere prese in considerazione le seguenti alternative, realizzare cioè il Controller lato Server come:

- servizio attivo nella macchina contenente il *Repository* gestito da Source Safe;

- web-service visibile sia nella *LAN* locale che esternamente ad essa utilizzando la libreria *System.Web.Services.WebService* di VB.Net.

Vincoli di progettazione

Da un'analisi combinata delle esigenze rilevate in fase di analisi e delle esigenze logiche di progettazione sono stati fissati i seguenti vincoli:

- non è possibile creare una dipendenza ad un file che è in modalità di modifica (*Lock*) da parte di un altro utente in quanto non si può essere certi che l'attributo a cui ci si vuole riferire resti invariato al termine delle eventuali modifiche apportate dal possessore del *lock*;

- ogni dipendenza aggiunta o eliminata in ambiente di lavoro (e non quindi solo salvata su file) va immediatamente condivisa con gli altri utenti Egen; questo perché, al fine di mantenere la consistenza a livello di progetto degli oggetti esistenti e delle loro dipendenze, è fondamentale il file di Impact locale sia sempre allineato con quello generale condiviso presente nel *Repository*; si riuscirebbe quindi a permettere un effettivo lavoro condiviso mantenendo però la logica esistente di Egen.

2.3 Obiettivi dello Stage

Date le premesse sulla struttura ed il funzionamento di Egen, e sulle risorse tecnologiche disponibili e in uso all'azienda, lo scopo dello stage è di sviluppare il software di controllo dello sviluppo condiviso che colleghi Egen al software di versionamento Source Safe, in modo che venga sempre mantenuta la consistenza dei dati in ogni momento durante l'utilizzo di Egen. Vanno risolte quindi le problematiche di integrazione tra Egen e Controller e tra Controller e Source Safe. Il Controller deve inoltre implementare una logica di gestione sicura, fornire un set di azioni completo ed essenziale ed evitar situazioni di inconsistenza dell'archivio di *business rules* e dei file critici.

Per il corretto funzionamento di Egen due tipologie di file assumono ruoli particolarmente importanti: il file di Indice e il file di Impact. Il primo contiene la gerarchia degli oggetti logici definiti e le relazioni che tra essi intercorrono, il secondo invece gli utilizzi incrociati di tali oggetti. Risultava pertanto necessario, al fine di non dover modificare il funzionamento di Egen, che tali file venissero sempre mantenuti aggiornati e che eventuali modifiche venissero adeguatamente propagate.

Vanno quindi affrontate e risolte le seguenti problematiche:

1. Il Controller deve integrarsi con uno strumento "*closed source*" per la gestione delle versioni e dello sviluppo concorrente attraverso l'uso delle API di MS Source Safe che tuttavia non forniscono una documentazione puntuale ed esaustiva.

2. Integrazione con il codice *legacy* di Egen, con problematiche di manutenzione e modifica del codice di uno strumento critico per l'azienda. Inoltre questa integrazione è tra piattaforme diverse: Egen in VB6 che espone oggetti *COM* e Controller scritto in VB.NET che espone componenti in codice gestito *.NET* e comunica tramite *Microsoft.VisualStudio.SourceSafe.Interop*.

3. Infine deve poter comunicare con client remoti, pertanto si aggiunge anche uno strato di comunicazione basato su Web Services per mettere a disposizione agli utenti remoti le funzionalità già disponibili su rete locale.

3 Attività di stage

3.1 Pianificazione

Le attività si è svolta nelle seguenti fasi:

- Formazione: studio del sistema esistente, degli strumenti di sviluppo utilizzati;
- Progettazione del Controller: progettazione nel dettaglio del Controller in base ai documenti prodotti da G. Favotto;
- Prototipazione e implementazione del Controller: realizzazione, test e messa in opera del Controller.

Il dettaglio e le tempistiche sono riportate nella tabella 1.

Essendo i componenti fortemente connessi, verrà redatto un unico Piano di Qualifica che verrà aggiornato durante lo sviluppo a partire dal documento di qualifica che sarà stato prodotto da G. Favotto. Lo sviluppo del PdQ è esteso a tutta la durata del progetto; l'impegno complessivo per il PdQ è stimato indicativamente a 5 giorni.

N°	*Fase ed attività*	*Preventivo ore*
1	Formazione	40
	Studio del sistema esistente, delle problematiche dello sviluppo condiviso, studio del codice legacy di Egen.	
2	Definizione di Prodotto	64
3	Prototipazione Controller	36
	Realizzazione di un applicativo che simula il comportamento di Egen.	
4	Implementazione del Controller	120
	Realizzazione e integrazione con Egen	
5	Piano di Qualifica	40
TOTALE		**300**

Tabella 1: dettaglio pianificazione delle attività

3.2 Formazione

La formazione per acquisire le competenze necessarie per svolgere il presente progetto di Stage ha riguardato soprattutto lo studio di Egen, del linguaggio di programmazione VB.NET e dello strumento di versionamento Microsoft Source Safe.

Source Safe

Per Source Safe erano necessari la ricerca e lo studio delle funzioni utili all'interfacciamento con il Controller per poter procedere nella Definizione di Prodotto con i dettagli dell'interazione tra Controller e Source Safe. In fase di codifica sono state inoltre testate e studiate le singole funzioni in modo da assicurare che il comportamento corrispondesse a quello documentato nel manuale di Source Safe. Questo perché essendo Source Safe uno strumento closed-source non si poteva verificare il comportamento del programma a partire dal codice stesso dell'applicazione, mentre era di estrema importanza che non si verificassero comportamenti imprevisti nelle chiamate alle funzioni di Source Safe per evitare errori che avrebbero compromesso l'integrità del *Repository*. Inoltre la documentazione di Source Safe non è esaustiva, fatto che non aiuta a raggiungere la necessaria sicurezza sul comportamento delle funzioni. Avendo comunque le funzioni di Source Safe dei tipi di ritorno omogenei ed essendo semplice il comportamento delle singole funzioni è stato possibile verificare in modo relativamente rapido il loro funzionamento già in fase di prototipazione.

Egen

Lo studio di Egen è stato fondamentale per la realizzazione del Controller, e già durante le fasi iniziali della progettazione i dettagli del codice di Egen hanno portato a modificare decisioni precedenti anche rilevanti, in un caso costringendo a rivedere uno dei requisiti definiti inizialmente (caso che verrà trattato nel dettaglio nella prossima sezione). Lo studio di Egen si è protratto praticamente per tutta la durata dello Stage, visto che per la sua complessità era impossibile studiare in anticipo tutti i dettagli che si sarebbero poi rivelati determinanti per l'interazione con il Controller, anche perché le sezioni delle azioni che andavano controllate si mescolavano al codice di azioni fuori dall'area di interesse del progetto che non potevano essere subordinate all'azione del Controller.

VB.NET

Per il linguaggio VB.NET dopo un'introduzione da parte del tutor aziendale Fabio Faieta è stata iniziata l'implementazione di un programma che simulasse il comportamento delle funzioni più semplici di Egen per avere una base su cui testare le prime funzioni del prototipo del Controller e nel contempo per poter prendere mano con il linguaggio di programmazione.

Il codice prodotto, oltre ad esser stato sottoposto a verifica per il Piano di Qualifica, è stato controllato e valutato per tutto l'arco dello stage dal Dott. Faieta, il quale, pur lasciando piena libertà nelle scelte progettuali e di codifica, ha fornito consigli per migliorare la qualità del codice, quindi la sua manutenibilità e la sua verificabilità.

3.3 Progettazione e prototipazione

3.3.1 Standard di progettazione architetturale

Il sistema si basa su un architettura a 3 livelli distribuita tra Client e Server le cui componenti principali risultano essere:

- Client: componente incluso nell'interfaccia grafica di Egen; gestisce localmente le chiamate ad azioni remote, trasferimenti ed aggiornamenti dei file, notifiche varie;

- Application Server: web-service attivo nel server ospitante il *Repository* Egen che fornisce l'interfaccia ai Client;

- Repository Server: modulo che implementa l'interfaccia per la gestione del *Repository* tramite Source Safe.

Il pattern permette di ottenere un buon grado di disaccoppiamento tra componenti distribuite, logica del servizio server e modalità di accesso al *Repository* di progetto.

Il sistema è assimilabile ad un *pattern Facade*: Egen si interfaccia solamente alle funzioni del componente Client, che gestisce le chiamate a tutte le altre componenti del Controller (Server, impactWriter, indexWriter, queue). Questo semplifica la comunicazione tra Egen e Controller e minimizza la dipendenza e l'accoppiamento tra le componenti.

Per via dell'interazione tra i vari Client con il server, il sistema riporta ad un *pattern Observer*: il *Repository* di Egen nel Data Base di Source Safe viene modificato in modo concorrente dai vari utenti tramite i rispettivi Client. Ogni modifica viene quindi propagata e diviene visibile a tutti i Client (e quindi agli utenti di Egen).

3.3.2 Procedura seguita nella Progettazione

Nonostante i requisiti iniziali fossero stabili e ben definiti, ed anche dopo un iniziale periodo di formazione, la progettazione del Controller ha presentato diverse difficoltà.

Inizialmente è mancata la conoscenza dei linguaggi VB.NET e VB6 in cui sarebbe stato scritto il Controller ed in cui è scritto Egen rispettivamente. La formazione iniziale non ha potuto comunque fornire la conoscenza e l'esperienza sufficienti per decidere con efficacia come progettare componenti in linguaggi diversi e di tipi diversi (Egen in *COM*, Controller in *.NET*, Web Service) che prevedono una forte interazione, e che saranno poi effettivamente utilizzati in ambito aziendale.

Inoltre la scrittura della Specifica Tecnica non era completa nelle fasi iniziali e si è provveduto quindi alla definizione delle funzionalità critiche del Controller.

Si è quindi deciso di procedere alternando la definizione di alcune parti base del Controller e la relativa realizzazione prototipale, raffinando la definizione delle componenti a seconda delle soluzioni tecniche individuate come più consone ed a seconda dei test effettuati, e modificando il codice di conseguenza, fino a raggiungere la versione finale della definizione e del codice.

All'inizio si è seguita questa procedura per la definizione e la realizzazione di alcune funzioni base descritte nella Specifica Tecnica, per ottenere poi un prototipo funzionante che realizzi in ogni aspetto le suddette funzioni, compresa l'integrazione delle funzioni del Controller in Egen e l'inserimento del *Repository* utilizzato in Source Safe.

3.3.3 Conseguenze nel modello di ciclo di vita

Questa procedura riporta ad un Modello Incrementale del Ciclo di Vita. Inoltre il fatto di aver iniziato la progettazione nel dettaglio e la realizzazione durante le fasi finali della stesura della Specifica Tecnica ha permesso di apportare delle correzioni alla Specifica stessa per includere casi inizialmente non previsti e messi in luce dall'analisi nel dettaglio dei vari problemi o da questioni sorte durante l'implementazione; questo include in piccola parte la progettazione architetturale al ciclo, il che riporta per la fase iniziale ad un Modello Evolutivo del Ciclo di Vita.

3.3.4 Definizione delle componenti

Verranno ora descritti le caratteristiche e i ruoli delle varie componenti all'interno del Controller, evidenziando le eventuali particolarità che hanno influenzato la loro progettazione.

Visual Source Safe

Source Safe mette a disposizione un insieme di interfacce incluse nello spazio dei nomi *Microsoft.VisualStudio.SourceSafe.Interop* (implementate nelle relative classi del medesimo spazio dei nomi) che vengono esposte agli utenti.

Le componenti principali sono:

- L'interfaccia IVSSDatabase che rappresenta il database in cui verrà salvato il *Repository*
- L'interfaccia IVSSItem che permette la gestione degli oggetti appartenenti al progetto aperto, e che permette di operare sul progetto con operazioni analoghe a quelle disponibili direttamente da Source Safe
- Le funzioni sul database (Open, Close) e sugli oggetti (Checkin, Checkout, UndoCheckout, …), per svolgere le opportune operazioni sui file relativi agli oggetti
- Le proprietà relative agli oggetti (isCheckedout, Name, …) che permettono di ottenere informazioni sui file relativi agli oggetti e sul loro stato

Un fattore importante da considerare è che l'installazione di Source Safe sulle macchine di ogni Client permette la connessione remota al database di Source Safe. Visto che Souce Safe va comunque installato nelle macchine degli sviluppatori per permettere il versionamento locale del *Repository*, è stato possibile far eseguire direttamente al Controller Client le operazioni di base sugli oggetti del database, esclusi Indice, Impact e code di cui va presa in considerazione la sola versione aggiornata sul server.

InstructionsQueue

Le code di istruzioni sono state create per diversi scopi: innanzitutto sono essenziali per garantire la propagazione delle modifiche ad Indice ed Impact.

Per quest'ultimo in particolare non può essere sufficiente inviare l'intero file al database di Source Safe ad ogni modifica e scaricarlo per ottenere gli aggiornamenti, come viene invece fatto per i vari altri file del *Repository* (Indice a parte). Infatti ogni minima modifica agli oggetti e alle relazioni in Egen comporta una modifica all'Impact che va immediatamente propagata. Visti il numero di operazioni che vengono effettuate, e visto che una singola modifica può innescare diverse istruzioni sull'Impact, si è ritenuto più opportuno trasferire le singole istruzioni piuttosto che l'intero file di Impact. Inoltre il file di Impact è il più pesante del *Repository* (normalmente un oggetto di Egen in formato *XML* occupa pochi kB), arrivando ad occupare facilmente centinaia di kB.

Soprattutto nell'ottica di un futuro utilizzo di Egen su client remoti connessi al database tramite Internet, un gruppo di sviluppatori che lavora contemporaneamente sullo stesso *Repository* centrale e genera potenzialmente decine di operazioni sull'Impact arriverebbe a generare un traffico notevole se ad ogni operazione dovesse essere scaricato l'intero file di Impact, provocando rallentamenti del programma e minando la fruibilità dello stesso.

Gli altri scopi delle code di istruzioni sono la tracciabilità e la possibilità di annullare le operazioni fatte. Le code contengono i dettagli di tutte le operazioni svolte, inclusi nome utente e *timestamp*, e sono mantenute nel server centrale. In questo modo è possibile risalire all'autore di ogni modifica.

Ci sono inoltre casi in cui le operazioni vengono propagate prima che ne venga effettuato il salvataggio: per garantire che lo sviluppatore possa eseguire una serie di modifiche ad un file prima di renderle effettive è stata prevista la possibilità di annullare le modifiche, in modo da non rendere l'utilizzo del programma poco pratico. Tuttavia ogni volta che un file viene modificato è necessario fare in modo che le condizioni che rendono possibile la modifica non vengano meno prima dell'effettivo salvataggio delle modifiche.

Perché questo sia possibile il Controller effettua per conto dell'utente il *Checkout* di tutti i file coinvolti dalla modifica dell'oggetto corrente (che è stato precedentemente estratto dall'utente). In questo modo quando si eseguirà il

salvataggio delle modifiche non ci sarà il rischio che i file collegati all'oggetto corrente siano stati modificati o estratti da altri sviluppatori.

Infine è stata prevista la possibilità che un utente modifichi un oggetto da lui precedentemente estratto, salvi localmente le modifiche ma decida in seguito di non volerle propagare e di voler tornare alla versione precedente del file, eseguendo un'operazione di *UndoCheckout*.

Tutte le code contengono variabili di tipo *DataSet*, che è una classe di Visual Studio per la gestione del contenuto *XML*. Utilizzando i *DataSet* è possibile trattare i file *XML* contenenti informazioni racchiuse in tag come fossero oggetti di una gerarchia idealmente simile ad un file system. Stabilita la gerarchia e la struttura del file *XML*, si possono inserire direttamente i dati negli oggetti di tipo opportuno, lasciando che la classe *DataSet* si occupi di aggiungere i tag come precedentemente impostati.

Le code di Indice e Impact hanno strutture e funzioni diverse, quindi è stato necessario creare due classi distinte ognuna con una propria struttura di *DataSet* e delle specifiche funzioni. Transient e Saved Queue accettano istruzioni analoghe a quelle della coda dell'Impact, ma hanno funzioni particolari. Inoltre le istruzioni di queste code vanno distinte da quelle su Index e Impact perché sono code locali e contengono istruzioni non definitive e quindi potenzialmente da annullare. Le code di Indice e Impact sono invece situate solo sul server.

Le istruzioni delle code di Indice e Impact contengono ciascuna un ID univoco incrementale che permette ai Client di recuperare le sole istruzioni aggiornate, non dovendo quindi scaricare ogni volta tutte le operazioni. Una volta ottenute le istruzioni aggiornate il Controller Client provvede semplicemente ad applicarle agendo tramite le classi indexWriter e indexWriter, descritte nella prossima sezione.

Data questa premessa, le funzioni delle singole code sono le seguenti:

- indexQueue: classe che raccoglie le istruzioni svolte sull'Indice del *Repository*, alla creazione o alla cancellazione di un file, o nel caso in cui un file sia rinominato o spostato in un altro punto del Repository. Di questa classe esiste una sola istanza situata nel Controller Server.

- impactQueue: classe che raccoglie le istruzioni svolte sull'Impact. Di questa classe esiste una sola istanza situata nel Controller Server. Le istruzioni vengono inviate alla impactQueue anche quando sono relative ad istruzioni temporanee, in modo che siano propagate a tutti gli utenti.

- transientQueue: classe che raccoglie le istruzioni sull'Impact relative a modifiche non salvate localmente ma già propagate all'Impact lato server. Le istruzioni vengono mantenute finché le modifiche non sono salvate, nel qual caso le istruzioni sono trasferite alla savedQueue, oppure finché le modifiche non sono annullate, nel qual caso vengono eseguite le istruzioni inverse a quelle precedentemente inserite.

- savedQueue: classe che raccoglie le istruzioni sull'Impact relative a modifiche salvate localmente e sul server, ma relative ad oggetti che non sono ancora stati archiviati. Se l'oggetto modificato viene archiviato le relative modifiche vengono eliminate dalla coda; se invece viene eseguito l'UndoCheckout sull'oggetto vengono eseguite le istruzioni inverse a quelle provocate precedentemente per ripristinare lo stato precedente.

E' possibile visualizzare il funzionamento delle code in relazione alle azioni dell'utente in figura 5, che rappresenta un'istruzione rivolta all'Impact, ed un esempio di definizione di una funzione di impactQueue nell'Esempio 1:

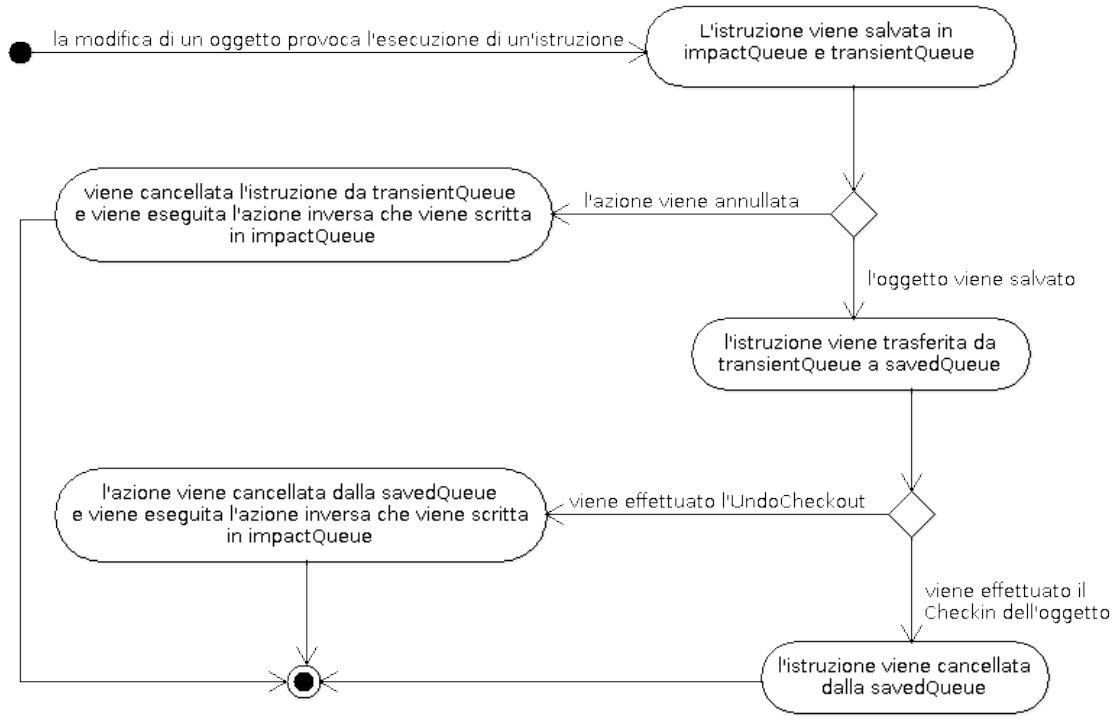

Figura 5: transito di un'istruzione sull'Impact nelle varie code.

GetQueueSince

Funzione che ritorna un dataset contenente tutte le istruzioni applicate all'impact lato server per conto di utenti diversi dall'utente chiamante e successive alla versione attuale dell'Impact locale. L'Id dell'utente e la versione dell'Impact (che corrisponde all'Id dell'ultima istruzione eseguita localmente) sono passati come parametro (rispettivamente user_id e instruction_id).

La funzione crea un array di righe di tipo DataRow chiamando la funzione Select di DataSet.Tables su dsQ. La Select cerca le istruzioni con Id maggiore di instruction_id inviate da utenti diversi da user_id.

In seguito la funzione crea un DataTable con struttura analoga a dsQ tramite Clone di DataSet.Tables ed inserisce una copia tutte le righe trovate con la Select tramite Add di DataTables.Rows. Infine la funzione crea un nuovo DataSet a cui viene assegnato il DataTable precedente assegnato.

La funzione ritorna il DataSet contenente le righe relative alle istruzioni ottenute tramite la Select.

Esempio 1 – Definizione della funzione GetQueueSince di impactQueue

Impact e Index Writer

Le classi Writer sono state create per effettuare modifiche sui file di Impact e Indice.

Un'ipotesi alternativa sarebbe stata quella di usare le stesse librerie di Egen per eseguire operazioni di modifica sui suoi file. Per realizzarla però il Controller avrebbe dovuto avere un riferimento ad un istanza di Egen, mentre Egen ha già un riferimento al Controller. Visto che Egen è scritto in VB6 non è possibile effettuare riferimenti circolari (cosa invece permessa in VB.NET).

Si è valutata quindi l'ipotesi di riscrivere le librerie di Egen in VB.NET, cosa che è comunque prevista per quando Egen verrà portato interamente in VB.NET. Tuttavia il *porting* delle librerie sarebbe stato eccessivamente oneroso rispetto alle tempistiche dello Stage, quindi l'idea è stata abbandonata. Si è preferito invece scrivere delle classi apposite utilizzando anche in questo caso la classe *DataSet* per manipolare il contenuto *XML*, analogamente a quanto si è fatto per le code di istruzioni.

Per comprendere il funzionamento di queste classi all'interno del Controller va premesso che di Impact e Indice esistono sia una copia locale (necessaria per il funzionamento di Egen) sia una copia remota sul server (che rappresenta la copia aggiornata di riferimento per tutti i Client). La copia locale è aggiornata da Egen stesso per quanto riguarda le modifiche effettuate dallo sviluppatore che lavora sulla stessa macchina locale. Le classi Writer entrano invece in gioco nella scrittura delle modifiche dell'Impact e dell'Indice lato server (dove non è presente una copia di Egen), e nella scrittura delle istruzioni eseguite sempre su Impact e Indice dagli altri sviluppatori.

Sia l’Impact locale che quella generale sul Server condividono la stessa struttura. Questa classe viene utilizzata dal Controller Server per propagare le modifiche effettuate dall’utente e dal Controller Client per scrivere sull’Impact locale le istruzioni generate da altri utenti. L'Indice ha una struttura diversa da quella dell'Impact, ma per il resto il caso dell'Indice è perfettamente analogo.

E' possibile leggere un esempio di definizione di una funzione di impactWriterClass nell'Esempio 2.

AddUse

Funzione che aggiunge la dipendenza richiesta alla Impact lato Server.

La funzione controlla la presenza di una riga Irw corrispondente all'oName in input con una Select di DataSet.Tables. Se la riga è assente, aggiungo creo una nuova riga con NewRow di DataSet.Tables inserendovi l'oName e l'oType in input e la aggiungo con Add di DataSet.Tables.Rows su ds. In seguito salvo le modifiche su disco con WriteXml di DataSet utilizzando la path della Impact passata in input.

Successivamente controllo se esiste già la Use corrispondente alla dipendenza di obName da oName con tutti i parametri di obName utilizzando Select di DataSet.Tables. Se la Use non esiste, creo una nuova riga use con NewRow di DataSet.Tables inserendovi i parametri di obName in input. Chiamo quindi writeUse per aggiungere la use alla Impact.

La funzione ritorna un array di stringhe con il risultato generale della funzione ("1" se ha avuto successo, "0" altrimenti), una stringa informativa relativa al risultato della funzione ed il risultato delle procedure eseguite al suo interno.

***Esempio 2** – Definizione della funzione AddUse di impactWriterClass*

Controller Server: Server_Logic

La parte del Controller che risiede nel server verrà chiamata Controller Server. La classe che implementa la logica del Controller Server si chiama invece Server_Logic.

Il Controller Server sarà installato nella stessa macchina del *Repository* generale di Egen. In questo modo interagirà direttamente con il progetto Source Safe principale. Il Controller Server fornirà i metodi necessari alle funzioni del Controller Client tramite Web Service, grazie a metodi ereditati da *System.Web.Services.WebService*. In questo modo il Controller Server si occuperà di mantenere aggiornati Impact ed Indice generali, fornendo anche ai Client i metodi per ottenere gli aggiornamenti di Impact e Indice. Come già descritto in precedenza, tutte le istruzioni eseguite su Impact ed Indice saranno infatti salvate su apposite code indicizzate: impactQueue ed indexQueue.

Il componente Controller Server è stato progettato per dialogare con Source Safe, quindi contiene funzioni per effettuare la connessione con il database contenente il *Repository* generale e per eseguire le funzioni classiche dello sviluppo condiviso. Queste funzioni vengono poi esposte tramite Web Services utilizzando i costrutti appositi di Visual Studio, i cui dettagli verranno trattati nel capitolo relativo alla Codifica. Inoltre il Controller Server contiene le istruzioni per aggiungere e rimuovere file dal *Repository*, ed eseguire istruzioni su Impact e Indice. Le funzioni sono strutturate in modo che ogni singola fase del processo sia portata a termine solo se tutte le operazioni hanno buon fine; in caso si verifichi un errore, l'intero processo è annullato, la situazione precedente è ripristinata e viene comunicata la natura dell'errore.

Va notato che le funzioni di Server_Logic che corrispondono alle varie azioni tipiche del controllo degli accessi (*Checkin*, *Checkout*, ecc...) non sono delle semplici chiamate alle rispettive funzioni di Source Safe, ma contengono controlli sui formati dei parametri in input e provvedono alla gestione delle eccezioni. Le funzioni hanno tipi di ritorno strutturati in modo da fornire le informazioni relative all'esito della funzione, con opportuna documentazione per spiegare il significato di ogni output possibile.

E' possibile leggere un esempio di definizione di una funzione di Server_Logic nell'Esempio 3.

SCheckout

Funzione che esegue il checkout del file se il file non è estratto da altri

utenti.

La funzione crea un oggetto VSSItem che si riferisce all'oggetto da archiviare. In seguito controlla il risultato di isCheckedOut di IVSSItem sull'oggetto. Se il file non è estratto viene eseguito Checkout di IVSSItem. Se il file è già estratto dall'utente viene segnalato lo stato d'estrazione. Se il file non è estratto, viene segnalato il problema indicando qual è l'utente che ha estratto il file.

Esempio 3 *– Definizione della funzione SCheckout di Server_Logic*

Controller Client: Client_Logic

La parte del Controller che risiede nelle macchine di ogni sviluppatore verrà chiamata Controller Client. La classe che implementa la logica del Controller Client si chiama invece Client_Logic.

Il Controller Client viene installato nelle macchine di ogni utente di Egen. Va impostata la connessione tramite *LAN* con il database di Source Safe nel server che mantiene il *Repository* generale.

Il Controller Client viene chiamato direttamente da Egen ogni volta che viene eseguita una delle funzioni definite nella Specifica Tecnica. Per ogni azione svolta dall'utente di Egen il Controller Client si occuperà di mantenere coerente ed aggiornato il *Repository*, in modo che ogni utente abbia accesso all'ultima versione di ogni file e che nessun utente possa modificare un file già estratto da un altro utente.

Il Controller Client si appoggia al Controller Server per le azioni su Impact e Indice, mentre per le altre azioni si connette direttamente al database Source Safe via *LAN* tramite la libreria *Microsoft.VisualStudio.SourceSafe.Interop*.

La classe Client_Logic contiene quindi dei riferimenti al database di Source Safe (per le operazioni dirette sui file), alle classi transientQueue e savedQueue (per memorizzare localmente le istruzioni propagate ma non definitive) ed al Controller Server (per utilizzarne i relativi Web Services).

Client_Logic include funzioni per la connessione al database di Source Safe, per le funzionalità di controllo dell'accesso ai file, per la propagazione delle modifiche ad Impact ed Indice e per il loro aggiornamento di versione.

Analogamente al caso del Controller Server, le funzioni per l'accesso condiviso non sono semplici invocazioni alle funzioni di Source Safe, ma includono controlli degli input, gestione delle eccezioni e ritorno del risultato.

E' possibile leggere un esempio di definizione di una funzione di Client_Logic nell'Esempio 4.

CUpdate

Funzione che esegue l'update locale dell'Impact e dell'Indice richiedendo al server le istruzioni applicate da altri utenti ed applicandole localmente. La versione dell'Impact e dell'Indice vengono aggiornate di conseguenza.

La funzione legge la versione attuale dell'Impact con CReadImpactVersion e CReadIndexVersion e successivamente le usa con CModifyImpact per aggiornare l'Impact e l'Indice ed ottenere il valore aggiornato della rispettive versioni. Infine chiama CwriteImpactVersion e CWriteIndexVersion per aggiornare effettivamente i valori delle versioni di Impact e Indice, e ritorna il risultato dell'operazione.

Esempio 4 *– Definizione della funzione CUpdate di Client_Logic*

Egen: Integrazione con il Controller

Il Controller viene integrato in Egen per intercettare le azioni dell'utente che implicano modifiche all'Indice e/o all'Impact, e relative modifiche al database.

Nella Definizione di Prodotto il comportamento del Controller viene descritto nel dettaglio per ogni azione tramite diagrammi di sequenza con relative descrizioni formali. Di seguito verranno presentate tre azioni di crescente complessità, ognuna presentata con l'immagine del diagramma di sequenza in figura a la relativa descrizione formale:

Cancellazione di una dipendenza:

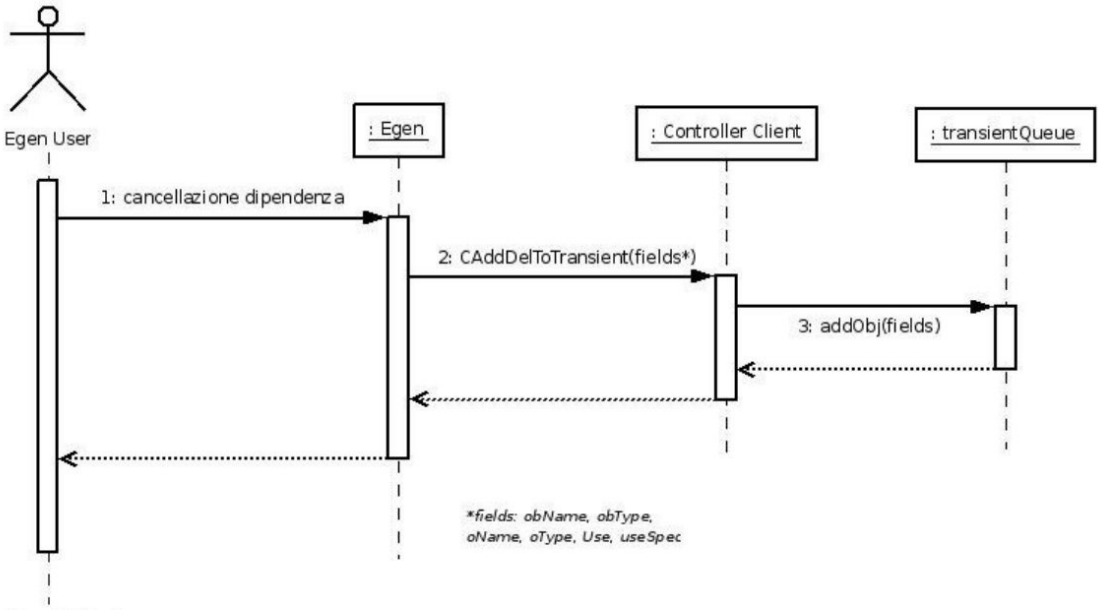

Figura 6: diagramma relativo all'azione "Cancellazione di una dipendenza"

Precondizioni: l'utente ha in checkout l'oggetto che intende modificare.

Sequenza:

1. *L'utente esegue una modifica che comporta la cancellazione di una dipendenza dall'Imact.*
2. *L'azione viene intercettata e viene invocata la funzione CaddDelToTransient del Controller Client.*
3. *Il client aggiunge quindi l'istruzione alla coda Transient tramite la funzione AddObj di transientQueue.*

Postcondizioni: l'istruzione è stata aggiunta alla coda transient.

Nota: l'istruzione verrà effettivamente eseguita al salvataggio dell'oggetto. Il motivo di questo comportamento è che in caso di uscita senza salvataggio potrebbe non essere possibile annullare la cancellazione della dipendenza, perchè l'oggetto padre potrebbe non avere più dipendenze ed essere cancellato, o potrebbe semplicemente essere stato cancellato l'attributo oggetto della dipendenza. Invece finché non viene cancellata la dipendenza né l'oggetto padre, né l'attributo oggetto della dipendenza possono essere cancellati. Questa è una modifica rispetto alla Specifica Tecnica, resasi necessaria per ottenere il comportamento desiderato senza compromettere la fruibilità del sistema.

Salvataggio di un oggetto:

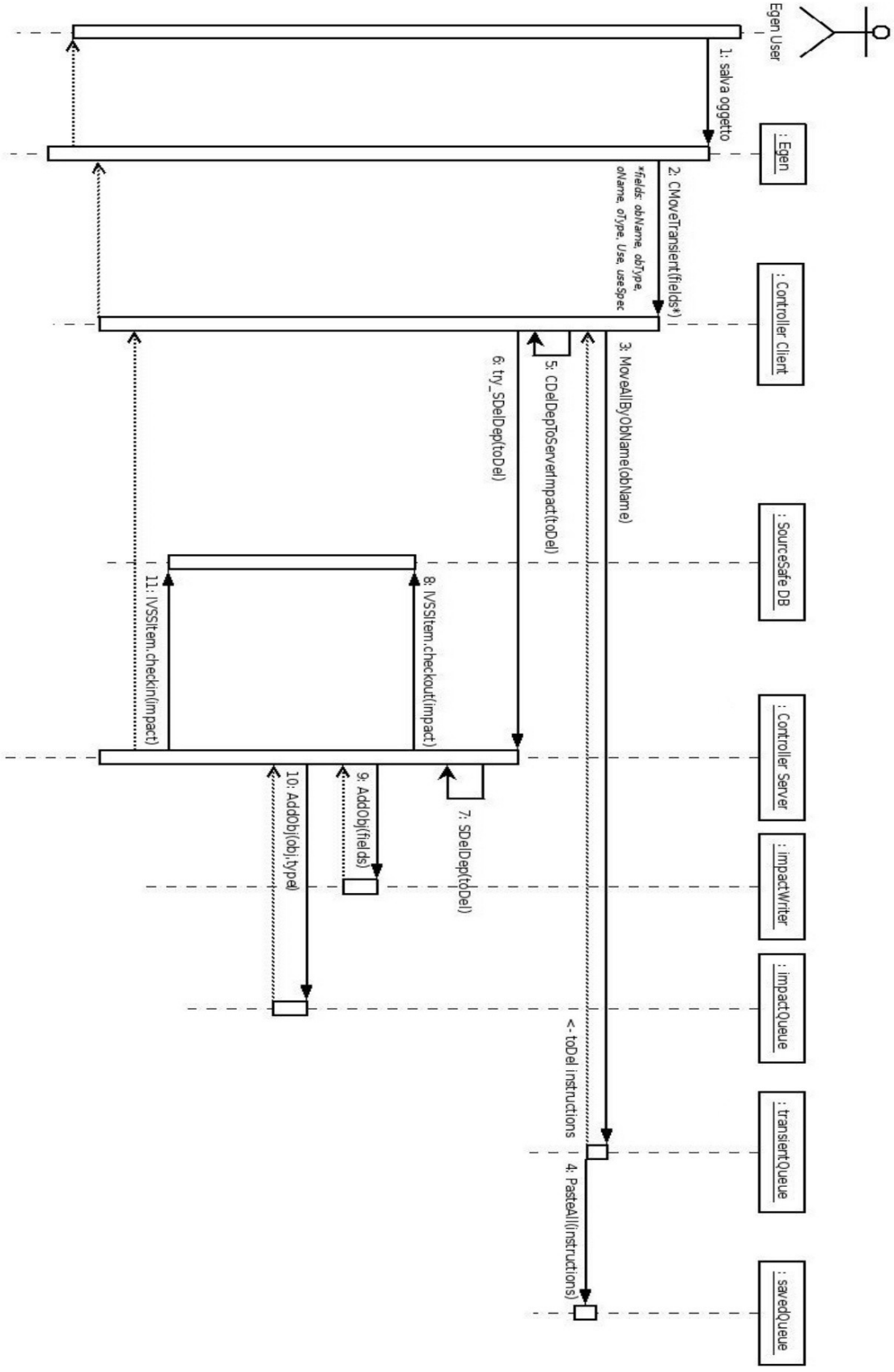

Figura 7: diagramma relativo all'azione "Salvataggio di un oggetto"

Precondizioni: l'utente ha in estrazione l'oggetto da salvare.

Sequenza:

1. *L'utente richiede il salvataggio dell'oggetto.*
2. *Egen chiama la funzione CmoveTransient e ne attende l'esito per eseguire il salvataggio in locale.*
3. *Il Controller Client chiama la funzione MoveAllByObName di transientQueue.*
4. *transientQueue chiama a sua volta PasteAll di savedQueue, che trasferisce le istruzioni relative ad obName alla savedQueue, e ritorna al client le istruzioni Del da eseguire.*
5. *Il client chiama la funzione CdelDepToServerImpact, passando in input la collezione delle istruzioni da eliminare.*
6. *La suddetta funzione provoca la chiamata al web service try_SDelDep del Controller Server.*
7. *Il server chiama quindi SdelDep per eseguire l'eliminazione delle dipendenze.*
8. *Il server esegue il checkout nel database della Impact.*
9. *Per ogni istruzione in input, il server chiama la funzione AddObj di impactWriter per cancellare la dipendenza.*
10. *Per ogni istruzione in input, il server chiama AddObj di impactQueue per aggiungere l'istruzione alla coda delle modifiche.*
11. *Infine il server esegue il checkin della Impact nel database.*

Postcondizioni: le istruzioni relative ad obName nella transientQueue sono state trasferite alla savedQueue e le istruzioni Del relative ad obName sono state applicate.

Update:

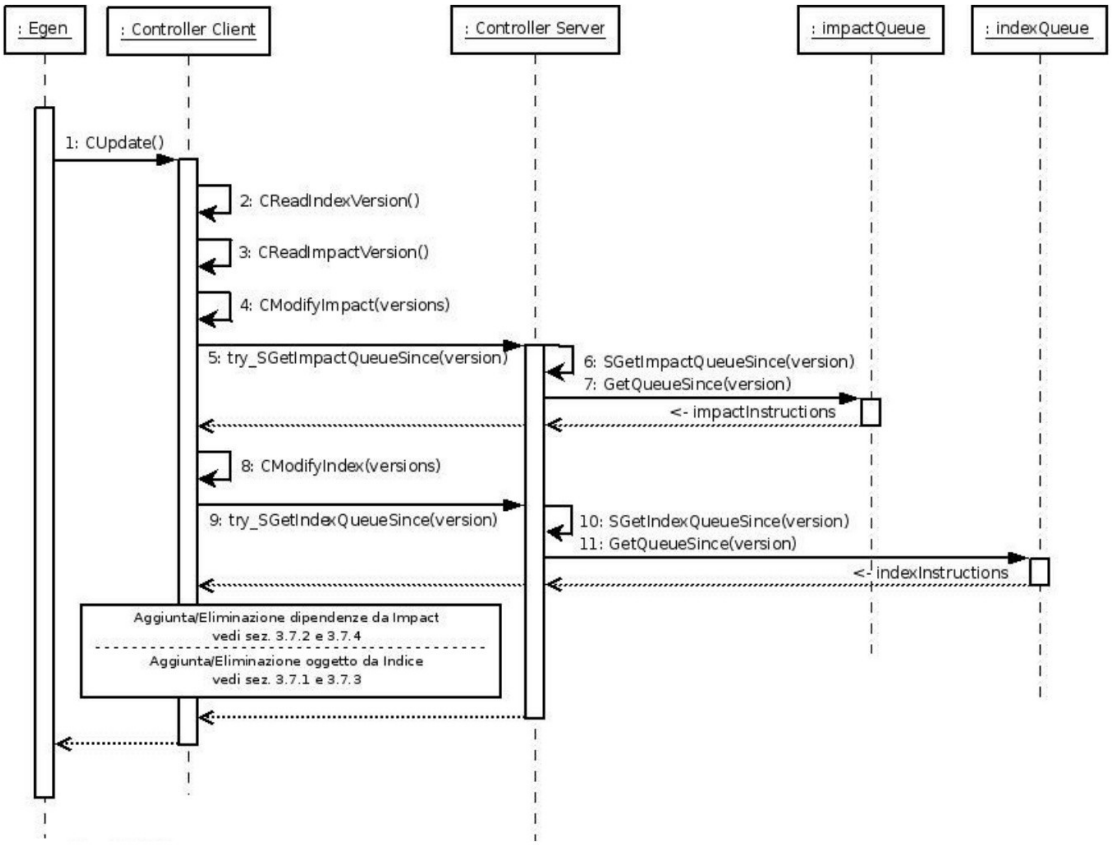

Figura 8: diagramma relativo all'azione "Update"

Precondizioni:

L'utente ha aperto un repository.

Sequenza:

1. *Egen chiama la funzione CUpdate del Controller Client.*
2. *Il client chiama CReadIndexVersion per ottenere il numero di versione dell'Indice.*
3. *Il client chiama CreadImpactVersion per ottenere il numero di versione dell'Impact.*
4. *Il client chiama CModifyImpact per eseguire l'aggiornamento dell'Impact.*
5. *Il client chiama quindi il web service try_SGetImpactQueueSince per ottenere la lista delle istruzioni per aggiornare l'Impact.*
6. *Il web service provoca la chiamata alla relativa funzione SGetImpactQueueSince.*

7. *Il server chiama GetQueueSince di impactQueue, che ritorna la lista delle istruzioni per aggiornare l'Impact.*

8. *Il client chiama CModifyIndex per eseguire l'aggiornamento dell'Indice.*

9. *Il client chiama il web service try_SGetIndexQueueSince per ottenere la lista delle istruzioni per aggiornare l'Indice.*

10. *Il web service provoca la chiamata a SgetIndexQueueSince.*

11. *Il server chiama GetQueueSince di indexQueue, che ritorna la lista delle istruzioni per aggiornare l'Indice.*

A questo punto per ogni istruzione ricevuta il client la applica eseguendo la relativa istruzione. Per le operazioni sull'Indice ed Impact vedere le relative sezioni.

3.4 Codifica e integrazione

Per la codifica è stato utilizzato l'ambiente di sviluppo Visual Studio 2005 , che offre le strumentazioni per la documentazione XML del codice, strumenti per la modellazione delle classi progettate, librerie di classi (tra cui quelle già citate come necessarie nella Specifica Tecnica), e debug del codice di librerie e applicazioni server. Le fasi di formazione e codifica sono state quindi semplificate dall'ambiente di lavoro, anche se alcune delle scelte progettuali sono state conseguenti proprio all'uso di questo strumento; ad esempio per i Web Service è stata utilizzata una classe fornita da Visual Studio e non il linguaggio originale WSDL, limitando la portabilità del codice di uno strumento nato proprio per fornire la massima interoperabilità, appunto i Web Service.

Nel contesto del progetto di stage l'uso di Visual Studio è stato valutato nel suo complesso positivo, in quanto ha permesso di ridurre i tempi di codifica e verifica in un contesto in cui il tempo a disposizione era una risorsa limitata.

3.4.1 Interazione COM / .NET / Web Service / Source Safe

Una particolarità del progetto di Stage è stata quella di richiedere l'interfacciamento di diversi programmi che utilizzano codici ed ambienti diversi in modo da cooperare in tempo reale evitando di intralciare il lavoro degli sviluppatori.

Egen è scritto in VB6 ed opera in ambiente *COM* (*Component Object Mode*), mentre il Controller è scritto in VB.NET ed utilizza il *framework .NET*. I due componenti del Controller (lato Client e lato Server) comunicano tramite Web Service. Infine, il Controller utilizza l'interfaccia di Source Safe per accedere al database contenente il *Repository* generale.

L'interazione tra *COM* e *.NET* è possibile esponendo esplicitamente la classe .NET che deve essere visibile al programma che lavora in COM. In questo caso è il Controller Client che si rende visibile ad Egen, con il tag che notifica che la classe Client_Logic è da rendere visibile alle classi in ambiente COM: *Microsoft.VisualBasic.ComClass()*

Per le classi queue e *Writer bastano degli import, in quanto sono classi interne al progetto del Controller nell'ambiente di sviluppo. Invece per potersi riferire correttamente al Controller Server, il Controller Client deve importare il Service esposto dal Server, in questo caso impostato come *Controller_Client_NS.Controller_Server.Service* .

Infine per il collegamento a Source Safe viene importata la relativa interfaccia: *Microsoft.VisualStudio.SourceSafe.Interop* .

Ecco l'intestazione della classe Client_Logic, che mostra tutte le suddette chiamate:

```
Imports Microsoft.VisualStudio.SourceSafe.Interop

Imports Controller_Client_NS.Controller_Server.Service

Imports queue

Imports impactWriter

Imports indexWriter



<Microsoft.VisualBasic.ComClass()> Public Class Client_Logic

...

End Class
```

Web Service

L'esposizione dei Web Service da parte del Controller server avviene tramite un'apposita sezione a fine classe annunciata dal tag:

```
<WebService(  Namespace:="http://...",  Description:="...")>
```

In seguito per ogni metodo che deve essere esposto viene creato un service che esegue la chiamata al metodo e ne ritorna il risultato. Ad esempio per esporre la funzione per il *Checkout* di un file il metodo è il seguente:

```
<WebMethod()> _
    Public Function try_SCheckout(ByVal fileProjPath As String) As String()
        Return s.SCheckout(fileProjPath)
    End Function
```

Esempio di codifica di un'azione: Checkin

Seguiremo ora parte della serie di chiamate che provoca un azione di *Checkin*, ovvero l'archiviazione di un oggetto nel *Repository* da parte di un utente. Questa azione viene provocata esplicitamente dall'utente per dopo averne effettuato l'estrazione e solo nel caso in cui voglia rendere persistenti le modifiche effettuate..

In blu sono indicati i commenti di documentazione della funzione, in verde sono indicati i commenti comuni.

Nel Controller Client la funzione chiamata per eseguire l'azione è la seguente:

```
    ''' <summary>
    ''' Esegue il checkin del file se il file è presente ed è checkedout dall'utente.
    ''' Se il file non esiste lo aggiunge al DB Source Safe e all'Indice lato Server.
    ''' </summary>
    ''' <param name="localPath">E' la path del file nella cartella di lavoro, incluso il nome del
    ''' file.</param>
    ''' <param name="projFolder">E' la cartella nel DB Source Safe in cui si trova _
    ''' il file che si vuole estrarre.</param>
    ''' <param name="file">E' il nome del file da estrarre.</param>
    ''' <param name="nameAnd_OrRelativeFolder">Oggetto da aggiungere all'Indice.
```

```
''' Questo parametro serve per aggiungere l'istruzione alla indexQueue solo se il file
''' non è presente nel DB (caso in cui viene aggiunto).</param>
''' <param name="objType">Tipo dell'oggetto di cui si vuol fare checkin.
''' Questo parametro serve per aggiungere l'istruzione alla indexQueue solo se il file
''' non è presente nel DB (caso in cui viene aggiunto).</param>
''' <returns>Ritorna un array di stringhe:
''' ret(0) = 0 se il checkin fallisce, 1 se ha successo, 2 se il file è stato aggiunto.
''' ret(1) = contiene il messaggio informativo relativo al risultato della funzione.
''' </returns>
''' <remarks></remarks>
Public Function CCheckin(ByVal localPath As String, ByVal projFolder As String, _
    ByVal file As String, ByVal nameAnd_OrRelativeFolder As String, _
    ByVal objType As String) As String()
    Dim tmp As String = ""
    ' creo la coppia di stringhe che contiene i dati per il return
    Dim ret(0 To 1) As String
    Try
        'Creo un oggetto che si riferisce al file che voglio archiviare
        Dim item As IVSSItem = CCdatabase.VSSItem(Replace(projFolder & file, _
        "\", "/"), False)
        Try
            'Se il file esiste non vengono lanciate eccezioni ed eseguo il checkin
            item.Checkin()
        Catch e As System.Runtime.InteropServices.COMException
            ' In caso di errore imprevisto segnalo il fallimento dell'operazione.
            ' Nota: il Checkin è abilitato se e solo se è stato precedentemente effettuato
            ' il Checkout del file da parte dell'utente!
            ret(0) = "0"
            ret(1) = "Checkin fallito"
            Return ret
        End Try
    Catch e As System.Runtime.InteropServices.COMException
        'Se il file non esiste, tento di aggiungerlo ad database
```

```
            tmp = CAdd(localPath, projFolder)
            'Quindi aggiungo il file all'Indice lato server
            server.try_SAddObjToIndex(userName, _
                serverIndexFile, _
                serverMainFolder, _
                sourceSafeProjMainFolder, _
                nameAnd_OrRelativeFolder, objType)
            ret(0) = "2"
            ret(1) = "File non trovato: " & tmp
            Return ret
        End Try
        ' Nessun errore, comunico l'esito positivo.
        ret(0) = "1"
        ret(1) = "Checkin riuscito"
        Return ret
    End Function
```

Seguiamo il caso in cui il file non era presente nel database, cioè è un file che è stato appena creato dall'utente, che lo sta archiviando nel database.

Questa chiamate provocano una modifica dell'Indice, che va propagata. Per questo viene chiamata il Web Service evidenziato in grassetto nel codice, ***server.try_SAddObjToIndex()***, che provoca la chiamata della funzione *SaddObjToIndex()* nel Controller Server (come spiegato precedentemente):

```
    ''' <summary>
    ''' Funzione che aggiunge un oggetto all'Indice lato Server.
    ''' </summary>
    ''' <param name="userName">Nome dell'utente che richiede l'aggiunta.</param>
    ''' <param name="serverIndexFile">Nome del file Indice.</param>
    ''' <param name="serverIndexFolder">Cartella di lavoro nel server per il file Indice.
    ''' Non include il nome dell'Indice.</param>
    ''' <param name="sourceSafeProjIndexFolder">Path per eseguire il checkout ed il checkin
    ''' dell'Indice nel DB di Source Safe.
```

```
    ''' Inizia sempre con $/
    ''' Non include il nome dell'Indice.</param>
    ''' <param name="nameAnd_OrRelativeFolder">Nome dell'oggetto e/o cartella
    ''' relativa del file
    ''' da aggiungere all'Indice.
    ''' Note: per specifiche consultare il commento della funzione
    ''' indexWriterClass.AddObj</param>
    ''' <param name="objType">Tipo dell'oggetto da aggiungere all'Indice.</param>
    ''' <returns>Ritorna una stringa informativa relativa al risultato delle funzioni di indexWriter
e queue chiamate.
    ''' </returns>
    ''' <remarks></remarks>
    Friend Function SAddObjToIndex(ByVal userName As String, _
    ByVal serverIndexFile As String, _
    ByVal serverIndexFolder As String, _
    ByVal sourceSafeProjIndexFolder As String, _
    ByVal nameAnd_OrRelativeFolder As String, _
    ByVal objType As String) As String

        'stringa di ritorno
        Dim ret As String = ""

        'checkout index
        System.Diagnostics.Debug.Print("Impact: " & _
          (SCheckout(sourceSafeProjIndexFolder & serverIndexFile))(1))

        'Aggiungo l'oggetto creando un'istanza di indexWriterClass e chiamandovi AddObj
        Dim inW As New indexWriterClass(serverIndexFolder & serverIndexFile)
        ret = inW.AddObj(nameAnd_OrRelativeFolder, objType)

        'checkin index
        System.Diagnostics.Debug.Print("Impact: " & (SCheckin(serverIndexFolder & _
          serverIndexFile, sourceSafeProjIndexFolder, serverIndexFile))(1))
```

```
    'Aggiungo l'istruzione alla coda indexQueue
    ret = ret & ": " & inQ.AddObj(userName, "Add", objType, nameAnd_OrRelativeFolder)(1)

    Return ret
End Function
```

Questa funzione provoca le chiamate alle funzioni di impactWriter e queue preposte all'aggiunta rispettivamente di riferimento al nuovo oggetto nell'Indice e di un'istruzione nella coda delle istruzioni indexQueue.

Riportiamo quindi il codice di ***AddObj()*** di indexQueue per fornire un esempio di una funzione che utilizzi i *DataSet* per la manipolazione del contenuto *XML*:

```
''' <summary>
''' Questa funzione aggiunge alla coda IndexQueue una istruzione richiesta dall'utente.
''' </summary>
''' <param name="uName">Nome dell'utente che richiede l'aggiunta</param>
''' <param name="iType">Tipo dell'istruzione da aggiungere alla coda.</param>
''' <param name="objType">Tipo dell'oggetto a cui si riferisce l'istruzione.</param>
''' <param name="nameOrFile">Valore dell'attributo File (per le Rels) o Name (per gli altri
''' oggetti).</param>
''' <returns>Ritorna un array di stringhe
''' ret(0) = 1 se la funzione ha successo, 0 altrimenti.
''' ret(1) = messaggio informativo relativo all'esito della funzione.
''' </returns>
''' <remarks></remarks>
Public Function AddObj(ByVal uName As String, _
ByVal iType As String, _
ByVal objeOrFile As String) As String()

    'array di stringhe per il return
    Dim ret(0 To 1) As String
    ret(0) = "0"
    ret(1) = "errore"
```

```
        'creo la riga da aggiungere alla coda
        Dim roA As DataRow = dsQ.Tables("instructions").NewRow()

        'riempio i campi della riga
        roA.Item("user_id") = uName
        roA.Item("iType") = iType
        roA.Item("objType") = objType
        roA.Item("nameOrFile") = nameOrFile
        roA.Item("timeStamp") = System.DateTime.Now

        'aggiungo la riga a salvo le modifiche
        dsQ.Tables("instructions").Rows.Add(roA)
        dsQ.AcceptChanges()

        'scrivo le modifiche sul file su disco
        dsQ.WriteXml(My.Resources.indexQueuePath, XmlWriteMode.WriteSchema)

        ret(0) = "1"
        ret(1) = "Istruzione aggiunta alla indexQueue"
        Return ret

    End Function
```

3.5 Verifica e validazione

Durante l'attività di Stage è stato seguito il Piano di Qualifica redatto nella prima parte dello Stage da Giulio Favotto, sia per la verifica della documentazione che per la verifica del codice.

E' stata posta particolare attenzione al tracciamento dei requisiti, in modo da rendere semplice la verifica dell'adempimento degli obiettivi iniziali. A partire dall'Analisi dei Requisiti, ogni requisito è identificato univocamente. In ogni documento successivo viene associata ogni componente al rispettivo requisito, in modo sempre più capillare.

Nella Definizione di Prodotto le singole funzioni vengono associate al requisito per cui sono state create. In questo modo è possibile verificare sia che tutti i requisiti sono rispettati, sia che non ci siano funzioni inutili o ridondanti che non sono effettivamente utili per il funzionamento del programma.

Durante la Progettazione di Dettaglio e contestuale codifica si è adottato un ciclo PDCA, nel seguente modo:

- **Pianificazione**: a partire dalle una delle componenti della Specifica Tecnica veniva definito il dettaglio delle relative funzioni nella Definizione di Prodotto, in prima istanza in base alle conoscenze acquisite durante la formazione, ed in iterazioni successive del ciclo in base ai risultati ottenuti ed alle soluzioni individuate come più adeguate.

- **Esecuzione**: veniva creato il codice relativo alla componente appena definita, o modificato in base al risultato delle iterazioni precedenti.

- **Controllo**: veniva controllata l'aderenza delle definizioni di dettaglio ai requisiti, e veniva esaminata la correttezza e la qualità del codice prodotto. E' stata controllato che il nuovo codice o le nuove componenti interagissero correttamente con quanto precedentemente prodotto.

- **Azione**: in caso fossero rilevati degli errori, o individuate soluzioni migliori, si è intervenuti per risolvere il problema, migliorando la definizione di dettaglio del componente e/o correggendo il codice, reiterando quindi il ciclo sul componente.

Il ciclo suddetto si riferisce in particolare al metodo generale adottato nel progetto, che riporta al Modello Incrementale di ciclo di vita. Le procedure seguite per Progettazione e Codifica riportano a loro volta ognuna un ciclo analogo PDCA, in scala ridotta proporzionalmente all'entità d'azione svolta.

Vista la complessità del progetto, una procedura disorganizzata non avrebbe potuto produrre risultati soddisfacenti in tempo utile, minando la buona riuscita del progetto di Stage.

Per il controllo del codice, oltre agli strumenti messi a disposizione da Visual Studio, è stato utilizzato il software NUnit per l'esecuzione di test dinamici di progetti .NET.

3.6 Valore Aggiunto dal Controller ad Egen

Il Controller, come già spiegato nel corso della relazione, è un componente fondamentale per garantire la fruibilità di Egen. La sua realizzazione ha permesso di aumentare le potenzialità di Egen, eliminando i vincoli che affliggevano la vecchia versione di Egen che rendevano necessari accorpamenti giornalieri delle modifiche effettuate dai vari sviluppatori.

Ora ogni sviluppatore può operare sul *Repository* con la garanzia che le modifiche da lui eseguite non entreranno in conflitto con quelle degli altri sviluppatori. Ogni modifica sarà visibile a tutti gli sviluppatori, che potranno aggiornare la propria versione locale del *Repository* in ogni momento. Le modifiche potranno essere annullate, ripristinando la situazione precedente grazie al programma di versionamento Source Safe ed alle funzioni di archiviazione ed esecuzione delle istruzioni del Controller.

Il Controller rende inoltre possibile l'utilizzo di Egen in rete, permettendo l'accesso remoto al *Repository* e quindi aumentando ulteriormente la flessibilità di Egen. Ad esempio gli sviluppatori presenti nelle diverse sedi dell'azienda potranno lavorare sullo stesso *Repository* ottenendo le modifiche in tempo reale.

4 Conclusioni

4.1 Consuntivo

I piani iniziali per l'attività di Stage si sono rivelati abbastanza realistici, ma è stato leggermente sottostimato il carico di lavoro necessario alla Progettazione ed alla Codifica, come si vede dal dettaglio in Tabella 2.

N°	*Fase ed attività*	*Preventivo ore*	*Consuntivo ore*
1	Formazione	40	40
	Studio del sistema esistente, delle problematiche dello sviluppo condiviso, studio del codice legacy di Egen.		
2	Definizione di Prodotto	64	72
3	Prototipazione Controller	36	40
	Realizzazione di un applicativo che simula il comportamento di Egen.		
4	Implementazione del Controller	120	136
	Realizzazione e integrazione con Egen		
5	Piano di Qualifica	40	40
TOTALE		**300**	**328**

Tabella 2:dettaglio consuntivo Piano di Lavoro

4.2 Raggiungimento degli obiettivi

Gli obiettivi iniziali posti per il progetto di Stage, e quindi i requisiti risultanti dall'analisi svolta nella prima del progetto, sono stati nel complesso rispettati.

Il Controller risponde si integra in Egen ed alla sua interfaccia, garantendo la corretta gestione del contenuto dei file *XML* sensibili (Impact e Indice) ed il rispetto delle procedure proprie dello sviluppo condiviso.

Le varie azioni previste nel Dettaglio di Prodotto implicano azioni esplicite, comandate cioè dall'utente in determinate occasioni, ed automatiche, cioè

provocate da altre azioni. Per quanto riguarda le azioni automatiche, queste vengono intercettate ed eseguite in modo trasparente in caso di esito positivo, o comunicando gli errori rilevati con informazioni dettagliate in caso di esito negativo. Le azioni esplicite forniscono sempre informazioni sull'esito della richiesta. In ogni caso vengono fornite informazioni utili alla risoluzione del problema eventualmente rilevato.

Va notato comunque che uno dei requisiti iniziali ha subito una modifica nel corso dell'evoluzione del progetto: inizialmente il Controller non avrebbe dovuto integrarsi in Egen, ma sfruttare i moduli di interfacciamento di Egen, chiamati “User Tools”. Questi moduli tuttavia consentono solamente delle chiamate esplicite, e non possono intercettare altre funzioni di Egen. Non potevano quindi essere utilizzati per gestire le azioni più complesse, fondamentali soprattutto per garantire l'integrità dei file critici e per il mantenimento delle condizioni che permettono ogni modifica ad oggetti del *Repository*. Potevano invece essere eventualmente utilizzati per le azioni esplicite, ma senza integrazione del Controller in Egen non si sarebbe potuto controllare l'abilitazione delle varie funzioni, che permette di creare un percorso obbligato per la modifica dei file senza minare la fruibilità del software. Senza integrazione si sarebbe invece dovuto lasciare all'utente l'onere di procedere sempre con la corretta serie di azioni, peggiorando la fruibilità di Egen e rischiando di minare la consistenza del database.

Si è deciso quindi, per questo solo caso, di modificare il requisito iniziale, documentando i motivi che hanno indotto a questa scelta. Per il resto è stato possibile verificare ogni requisito grazie al tracciamento eseguito in ogni fase del progetto, dai bisogni iniziali alle funzioni del codice del Controller.

Sebbene i requisiti obbligatori e desiderabili siano stati soddisfatti, non è stato possibile per problemi di tempo raffinare l'interfaccia ed inserire alcune funzionalità che avrebbero reso più comodo l'utilizzo di Egen. L'idea di inserire alcune di queste funzionalità è nata durante la codifica e l'integrazione del Controller in Egen, o a seguito dei test d'uso del prodotto. Una di queste funzionalità era però prevista in uno dei requisiti opzionali, che non è quindi stato rispettato.

4.3 Competenze e nozioni acquisite nel progetto di Stage

Il progetto di Stage ha richiesto di affrontare diverse tecnologie, molte delle quali non erano state presentate durante il corso di studi.

Sono stati analizzati in maniera approfondita gli strumenti con cui il software da sviluppare doveva integrarsi (Source Safe ed Egen), nonché linguaggi e strumenti da utilizzare per la successiva realizzazione (VB6, VB.NET, Visual Studio 2005). Questo ha permesso di studiare i dettagli del codice di un programma particolare, Egen, creato dall'azienda per un utilizzo professionale, potendo trarre quindi importanti spunti per quanto riguarda la progettazione, la codifica ed i meccanismi interni di un programma di elevata complessità. Inoltre i linguaggi di programmazione Visual Basic e l'ambiente di sviluppo Microsoft Visual Studio sono molto diffusi in ambito professionale, e l'averli studiati ed utilizzati per il progetto di Stage rappresenta un vantaggio per lo sviluppo personale nell'ottica dell'inserimento nel mondo del lavoro.

Le problematiche di integrazione verso il software *legacy* di Egen hanno richiesto lo studio dei sistemi di integrazione tra tecnologie COM e .NET, che saranno utili per assolvere a richieste di riutilizzo della applicazioni aziendali meno recenti o di espansione o integrazione delle suddette applicazioni con tecnologie più moderne.

Per consentire lo sviluppo condiviso ad utenti remoti è stata utilizzata la tecnologia dei Web Service, che fornendo un'interfaccia universale ai servizi Web sono particolarmente utili a gestire le applicazioni che possano richiedere buona interoperabilità tra elaboratori diversi connessi in rete.

Infine tutta la persistenza è gestita con tecnologie *XML*, su cui in generale si basano molte delle tecnologie Web moderne, inclusi tra l'altro i Web Service.

Il progetto di Stage ha quindi permesso di acquisire nozioni particolarmente utili in quanto riguardanti tecnologie molto diffuse in ambito professionale. Inoltre ha permesso comprendere cosa comporti lavorare in un team di sviluppo all'interno di un azienda, dovendo quindi confrontarsi con esigenze specifiche in un contesto diverso da quelle di un corso di studi universitario.

4.3.1 Valutazione delle conoscenze pregresse

Le competenze necessarie per affrontare le suddette tecnologie non erano state presentate nel corso di studi, o comunque non in modo da consentire di affrontarle senza affrontare prima un ulteriore studio che permettesse di risolvere in maniera adeguata i problemi propri dei casi particolari incontrati. Tuttavia le nozioni acquisite negli esami di Programmazione e Linguaggi di Programmazione, di Sicurezza nei Sistemi di Calcolo, di Ingegneria del Software e di molti altri esami affrontati nel corso di studi hanno permesso di minimizzare i tempi di formazione, e di individuare soluzioni efficaci ai problemi che si sono presentati nel corso del progetto. Ad esempio, nonostante i linguaggi proprietari Microsoft Visual Basic 6 e .NET non fossero stati trattati nel corso di studi, una volta assimilati i dettagli di sintassi dei linguaggi VB si sono potute applicare le nozioni di Programmazione e Ingegneria del Software che hanno permesso di ottenere codice affidabile, efficace, manutenibile e documentato.

Per la modellazione dei componenti è stato utilizzato UML, anch'esso studiato nel corso di Ingegneria del Software. In generale, proprio quest'ultimo ha fornito la preparazione e la metodologia appropriate per affrontare un progetto di questa complessità. Senza nozioni su processi software e relativi standard di processo, su qualità del software, su progettazione e documentazione, e verifica e validazione non sarebbe stato possibile ottenere un software che incontrasse le esigenze di funzionalità, affidabilità e manutenibilità di un'azienda di sviluppo software.

In definitiva ritengo che il corso di studi abbia fornito una buona preparazione per affrontare progetti software anche complessi, dato che anche nei corsi che affrontavano temi specifici alle nozioni particolari è stato affiancato lo studio di concetti e pratiche generali utili.

4.3.2 In conclusione

La valutazione dell'esperienza di Stage nel suo complesso è sicuramente positiva, essendosi svolta in un ambiente stimolante e costruttivo ma comunque disteso, ed avendo permesso di acquisire abilità e conoscenze utili ad un futuro ingresso nel mondo del lavoro.

Glossario

COM: il Component Object Model è un'interfaccia per componenti software introdotta da Microsoft nel 1993. *COM* permette la comunicazione tra processi e creazione dinamica di oggetti con qualsiasi linguaggio di programmazione che supporta questa tecnologia. Per poter interagire con il più recente *.NET* è necessario che quest'ultimo esponga esplicitamente le funzionalità *.NET* in *COM*.

Database: archivio di dati riguardanti lo stesso argomento o più argomenti tra loro correlati, strutturato in modo tale da consentire la gestione dei dati stessi da parte di applicazioni software.

Documentazione XML: si tratta del metodo usato in Visual Basic per la generazione automatica di documentazione del codice a partire da particolari commenti scritti in forma apposita dall'utente all'inizio delle funzioni. E' possibile fare un analogia con lo strumento *JavaDoc* per il linguaggio Java visto nel corso di studi.

ERP: *Enterprise Resource Planning* (Pianificazione delle risorse d'Impresa). L'*ERP* è un sistema informatico che integra tutti gli aspetti del business e i suoi cicli, inclusa la pianificazione, la realizzazione del prodotto, le vendite, gli approvvigionamenti, gli acquisti, la logistica, il magazzino ed il marketing.

Framework .NET: è l'ambiente per la creazione, la distribuzione e l'esecuzione di tutti gli applicativi che supportano *.NET* siano essi Servizi Web o altre applicazioni.

Legacy : un sistema *legacy* è un sistema informatico esistente o un'applicazione che continua ad essere usata poiché l'utente non vuole o non può rimpiazzarla. In questo caso Egen è scritto in un linguaggio precedente a quello in cui è scritto il Controller, e non può essere

.NET: è una piattaforma di sviluppo Microsoft indipendente dalla versione del sistema operativo Microsoft installato, proposta come evoluzione del modello *COM*. *.NET* contiene infrastrutture per la compatibilità con qualunque linguaggio orientato agli oggetti, anche se la *suite* fornisce solo alcuni linguaggi proprietari Microsoft (VB.NET, C#, J#, Managed C++). E' strutturalmente comparabile con

J2EE, il corrispettivo Java prodotto dalla Sun Microsystems.

Porting: consiste nell'*adattamento* o *modifica* di un componente software al fine di consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

Repository: archivio. Per il presente documento si farà riferimento al *Repository* come all'archivio contenente tutte i file relativi ad un singolo progetto creato da Egen.

Timestamp: è una sequenza di caratteri che denota la data e/o l'ora in cui si è verificato un evento.

XML: acronimo di eXtensible Markup Language, è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati.

Bibliografia

Documentazione prodotta durante il progetto di Stage:

Piano di Lavoro, Definizione di Prodotto, Piano di Qualifica
Analisi dei Requisiti (G. Favotto), Specifica Tecnica (G. Favotto)

Documentazione tecnica per la programmazione e guida per i sistemi di sviluppo Microsoft, MSDN Library:

http://msdn.microsoft.com/it-it/library/default.aspx

La guida alla Software Engineering Body of Knowledge (oltre alle pagine del corso di Ingegneria del Software):

http://www.swebok.org/

Pagina principale di Wikipedia:

http://www.wikipedia.org/