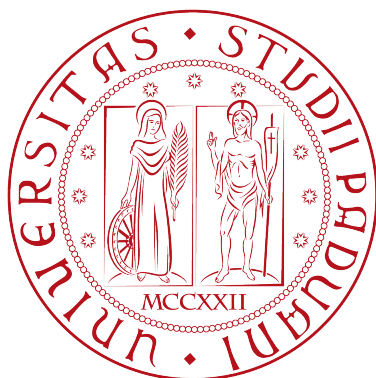


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



**Sviluppo di un sistema completo di update
e installazione per prodotti CAD/CAM**

Tesi di laurea triennale

Relatore

Prof. Massimo Marchiori

Laureando

Eduard Bicego

ANNO ACCADEMICO 2015-2016

A Elena

Sommario

Il presente documento descrive il progetto sviluppato durante il periodo di stage del laureando Eduard Bicego presso l'azienda Salvagnini S.p.A. sotto supervisione del tutor aziendale Alberto Conz.

Il progetto richiedeva lo sviluppo di un sistema semplificato per la distribuzione ai clienti dei prodotti software sviluppati all'interno dell'azienda. Gli obiettivi erano quindi lo sviluppo di un'applicazione installabile lato client e un servizio REpresentational State Transfer (REST).

Ringraziamenti

Desidero ringraziare tutti coloro che in questo percorso universitario hanno condiviso una parte del cammino verso il mondo informatico insieme a me. In particolare ringrazio il Prof. Massimo Marchiori, relatore della mia tesi, per la sua immensa passione nell'insegnamento e il gruppo Leaf: Andrea, Davide, Cristian, Federico, Marco e Oscar per la fatica e la gioia condivisa quest'anno.

Ringrazio con affetto i miei genitori: Gianpaolo e Rossella per l'enorme sostegno e l'immenso aiuto nel conseguire questo obiettivo.

Ringrazio i miei amici e la stupenda famiglia dell'A²CMMS che in questi anni ha consentito la mia crescita personale in un vortice di intense esperienze ed emozioni.

Ringrazio poi il mio tutor aziendale Alberto Conz e il suo collega Davide Fasolo per i numerosi insegnamenti e spunti ricevuti durante tutto il periodo di stage.

Infine desidero ringraziare chiunque, anche se solo per un momento, mi ha accompagnato fin qui e chi mi accompagnerà nel prosieguo.

Settembre 2016

Eduard Bicego

Indice

Indice	viii
Elenco delle figure	ix
1 Introduzione	1
1.1 Organizzazione del documento	2
2 Analisi	3
2.1 Requisiti di vincolo	3
2.2 Requisiti funzionali	4
2.2.1 Lato Client	4
2.2.2 Lato Server	7
2.3 Requisiti di qualità	8
2.3.1 Soddisfacimento	8
3 Tecnologie adottate	9
3.1 Parte Client	9
3.1.1 Linguaggi	9
3.1.2 Tecnologie esterne	9
3.1.3 Tecnologie interne	12
3.2 Parte Server	13
3.2.1 Linguaggi	13
3.2.2 Tecnologie esterne	13
4 Architettura	19
4.1 Introduzione	19
4.2 Architettura generale	20
4.3 Macro componenti	23
4.3.1 Updater	23
4.3.2 Updater User Interface Controls	26
4.3.3 Updater Client Library	29
4.3.4 Updater Serialization	33
4.3.5 Updater Server	34
4.3.6 Updater Database Entity Framework (DBEF)	37

4.4	Relazioni esterne tra componenti	38
4.4.1	Relazioni Updater	38
4.4.2	Relazioni Updater UIControls	39
4.4.3	Relazioni Updater Client Library	40
4.4.4	Relazioni Updater Server	41
5	Dettagli componenti	43
5.1	Updater - Parte client	44
5.1.1	Interfaccia grafica	44
5.1.2	Flusso d'esecuzione operazioni principali	46
5.2	Updater - Parte server	50
5.2.1	Sito web amministrativo	50
5.2.2	Web Api - Documentazione	55
5.3	Concorrenza	57
5.3.1	Pattern	57
5.3.2	Componenti coinvolte	57
5.3.3	Criticità e miglioramenti	58
5.3.4	Approfondimenti	58
5.4	PowerShell script	59
5.4.1	Ambito applicativo dello script	59
5.4.2	Implementazione script powershell	59
6	Conclusioni	61
6.1	Futuri sviluppi	61
6.2	Future estensioni	62
	Bibliografia	63
	Glossario	65

Elenco delle figure

4.1	Dipendenze tra macro componenti del sistema Updater	21
4.2	Esempio generico di comunicazione tra parte client e parte server	22
4.3	Struttura macro componente Updater	25
4.4	Diagramma di sequenza per la costruzione degli elementi grafici UIControls	27
4.5	Struttura macro componente Updater UIControls	28
4.6	Struttura macro componente Updater Client Library	32
4.7	Struttura macro componente Updater Server	36
4.8	Relazioni componenti di Updater con le componenti di UIControls e Client Library	38
4.9	Relazioni componenti di Updater UIControls con le componenti di Client Library	39
4.10	Relazioni componenti di Updater Client Library con le componenti di Updater Serialization	40
4.11	Relazioni componenti di Updater Server con le componenti di Updater Serialization e UpdaterEBEF	41
5.1	Componenti grafiche della finestra principale	44
5.2	Updater - Finestra principale	45
5.3	Updater - Schermata delle impostazioni	46
5.4	Diagramma di attività - Download	47
5.5	Diagramma di attività - Installazione	48
5.6	Diagramma di attività - Aggiornamento	49
5.7	Pagina web - Products.html	51
5.8	Pagina web - Prerequisites.html	52
5.9	Pagina web - Dependencies.html	53

Capitolo 1

Introduzione

Il progetto richiesto getta le basi per la creazione di un forte sistema di *continuous delivery* per i prodotti software sviluppati dall'azienda.

Lo sviluppo di programmi software CAD/CAM richiede la gestione di un ambiente di sviluppo molto complesso e in continua evoluzione per bug-fix e per aggiunte di nuove funzionalità richieste esplicitamente dai clienti aziendali nonché utenti e fruitori di tali programmi. Il rilascio di software gestito manualmente presenta sempre numerosi svantaggi tanto da essere addirittura definito un antipattern¹. Gli svantaggi sono:

- Un ciclo di vita, ossia il tempo che intercorre tra la decisione di un cambiamento nel software e la consegna di esso all'utente finale, molto lungo;
- Costi molto elevati per il rilascio del prodotto software;
- Una complessa gestione manuale di un insieme di processi con un alto tasso di errore, soprattutto se questi non vengono seguiti pedissequamente o peggio se non sono definiti;
- Un dispendioso utilizzo di risorse e tempo.

Per un'azienda che distribuisce software quindi risulta necessario un meccanismo di *continuous delivery*, una distribuzione dei propri software in modo **completamente automatizzato** rendendo così il rilascio un processo veloce e ripetibile.

Il sistema che verrà presentato nel presente documento forma quella che diventerà la base per la distribuzione dei prodotti aziendali. In particolare offrirà:

¹ [8] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, cap. 1, pag. 5.

- Un modo semplice per usufruire dei prodotti aziendali e dei loro aggiornamenti, garantendo che l'utente finale possa sempre restare al passo con i cambiamenti dei programmi installati;
- Le basi per un'automazione della distribuzione del software, garantendo **risparmio sul personale** e **velocizzazione dei tempi di rilascio**;
- Un controllo più veloce e completo per la distribuzione del proprio software e delle licenze d'uso.

Il sistema nasce già suddiviso in due parti, una parte client ed una server. La parte client sarà responsabile di offrire le operazioni all'utente grazie ai dati distribuiti dalla parte server la quale gestirà i dati dei prodotti software e sarà collegata al sistema di *continuous integration* aziendale. In questo modo il processo di distribuzione sarà il più possibile automatizzato. Il sistema di *continuous integration* garantirà l'output di un programma installabile ed eseguibile, questo verrà caricato nel database gestito dalla parte server e in automatico verrà distribuito a tutti gli utenti che usufruiscono della parte client.

1.1 Organizzazione del documento

Il documento si prefigge principalmente di essere una guida per la costruzione di tale sistema, adotta quindi un approccio abbastanza generico. Il contenuto principale presentato è l'architettura ad alto livello la quale è svincolata dalla presentazione di tutti i dettagli implementativi. Di questi infatti se ne occupa una parte secondaria di approfondimento che presenta quelli più significativi e quelli ritenuti più interessanti.

- Il capitolo 2 mette in evidenza le funzionalità da implementare attraverso l'elenco dei requisiti;
- Il capitolo 3 presenta l'elenco delle tecnologie coinvolte nello sviluppo del sistema;
- Il capitolo 4 presenta l'architettura ad alto livello che lega ogni macro componente che va a formare il sistema prefiggendosi l'obiettivo di aiutare il lettore a comprendere il sistema già creato o a implementarne uno nuovo;
- Il capitolo 5 invece racchiude una serie di sezioni di approfondimento per entrare, dove ritenuto più utile, nei dettagli dell'implementazione del sistema.

Capitolo 2

Analisi

La prima parte del progetto di stage è stata basata sulla definizione dei requisiti software che il prodotto dovrà soddisfare. Alcuni requisiti hanno subito diverse variazioni. Qui di seguito sono elencati solamenti i requisiti nella loro versione finale.

I requisiti a seguire sono stati classificati per le seguenti tipologie:

- Di vincolo;
- Funzionali;
- Di qualità.

I requisiti di tipo funzionale data la loro quantità elevata sono stati suddivisi in due sezioni:

- Requisiti funzionali lato client;
- Requisiti funzionali lato server.

Inoltre sono stati classificati con la seguente codifica: **R[X][Y]** per facilitarne la consultazione. La codifica deve essere interpretata nel modo seguente:

1. **X** indica la priorità del requisito e può assumere uno dei seguenti valori:
 - **Obb** per requisito obbligatorio;
 - **Opz** per requisito opzionale.
2. **Y** indica il codice gerarchico che identifica un requisito e che è unico.

2.1 Requisiti di vincolo

Gli unici requisiti di vincolo presenti sono:

1. Sviluppare il sistema in un ambiente Microsoft Windows;

2. Sviluppare il programma lato client compatibile con i sistemi operativi Windows XP, Windows Vista, Windows 8 e Windows 10;
3. Sviluppare utilizzando tecnologie Open Source con licenza MIT o Microsoft (licenza acquisita dall'azienda).

2.2 Requisiti funzionali

2.2.1 Lato Client

RObb1 Il programma deve consentire la visualizzazione dei prodotti software disponibili;

RObb1.1 Il programma deve consentire la visualizzazione del nome dei prodotti software;

RObb1.2 Il programma deve consentire la visualizzazione di una breve descrizione per ogni prodotto software;

RObb1.3 Il programma deve consentire la visualizzazione di un'indicazione se è disponibile un aggiornamento per il prodotto software disponibile;

RObb1.4 Il programma deve consentire la visualizzazione filtrata dei prodotti software disponibili;

RObb1.4.1 deve consentire di visualizzare tutti i prodotti software;

RObb1.4.2 deve consentire di visualizzare i prodotti software installabili;

RObb1.4.3 deve consentire di visualizzare i prodotti software con aggiornamento disponibile;

RObb1.4.4 deve consentire di visualizzare i prodotti software con il nome più vicino alla stringa inserita dall'utente in un'apposita casella di ricerca;

RObb2 Il programma deve fornire indicazioni più dettagliate per ogni prodotto software;

RObb2.1 Il programma deve fornire il nome del prodotto;

RObb2.2 Il programma deve fornire una descrizione esaustiva del prodotto;

RObb2.3 Il programma deve fornire la data di rilascio del prodotto;

RObb2.4 Il programma deve fornire un'indicazione se il prodotto è già installato nella macchina;

RObb3 Il programma deve permettere all'utente di eseguire operazioni per ogni prodotto software;

- RObb3.1** Il programma deve permettere di scaricare il file di installazione del prodotto software nella macchina;
- RObb3.2** Il programma deve permettere di annullare il download in corso di un prodotto software;
- RObb3.3** Il programma deve permettere di installare il prodotto software nella macchina;
- RObb3.4** Il programma deve permettere di scaricare e installare il prodotto software nella macchina;
- RObb3.5** Il programma deve permettere di reinstallare il prodotto software precedentemente installato;
- RObb3.6** Il programma deve permettere di scaricare e installare gli aggiornamenti di un prodotto software installato precedentemente;
- RObb4** Il programma deve fornire all'utente delle preferenze modificabili dall'utente;
 - RObb4.1** deve fornire all'utente la possibilità di attivare gli avvisi per gli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.2** deve fornire all'utente la possibilità di disattivare gli avvisi per gli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.3** deve fornire all'utente la possibilità di attivare il download automatico degli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.4** deve fornire all'utente la possibilità di disattivare il download automatico degli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.5** deve fornire all'utente la possibilità di attivare il download e l'installazione automatica degli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.6** deve fornire all'utente la possibilità di disattivare il download e l'installazione automatica degli aggiornamenti disponibili per i prodotti software installati;
 - RObb4.7** deve fornire all'utente la possibilità di impostare un'ora in cui si effettuerà il download e l'installazione automatica degli aggiornamenti disponibili per i prodotti software installati;
- RObb5** Il programma deve segnalare con un avviso nel caso sia disponibile un aggiornamento o più aggiornamenti per i prodotti software installati;

- RObb6** Il programma deve segnalare con un avviso nel caso sia disponibile un aggiornamento o più aggiornamenti per i prodotti software installati e questi siano stati scaricati;
- RObb7** Il programma deve segnalare con un avviso nel caso sia stato scaricato e installato automaticamente un aggiornamento disponibile o più aggiornamenti disponibili per i prodotti software installati;
- RObb8** Il programma deve segnalare con un avviso nel caso non sia stato possibile reperire informazioni sugli aggiornamenti disponibili o nel caso il download automatico di un aggiornamento non sia riuscito o nel caso l'installazione automatica di un aggiornamento non sia riuscita;
- RObb9** Il programma deve consentire di poter aggiornare tutti i prodotti software con l'aggiornamento disponibile;
- RObb10** Il programma deve consentire all'utente di aggiornare la lista di prodotti software per verificare nuovi aggiornamenti disponibili;
- RObb11** Il programma deve segnalare all'utente se il server che offre il servizio online è raggiungibile;
- RObb12** Il sistema deve fornire una libreria software che operi in ambiente client usufruita dal programma;
- RObb13** La libreria deve fornire diverse funzionalità;
- RObb13.1** La libreria deve permettere di effettuare operazioni ad un database remoto tramite servizi web;
 - RObb13.1.1** La libreria deve reperire informazioni sui programmi Salvagnini disponibili online;
 - RObb13.1.2** La libreria deve reperire informazioni su un singolo prodotto software disponibile;
 - RObb13.1.2** La libreria deve reperire informazioni su un insieme di prerequisiti disponibili online;
 - RObb13.1.3** La libreria deve reperire informazioni su un singolo prerequisito disponibile online;
 - RObb13.2** La libreria deve permettere di effettuare operazioni sui prodotti software;
 - RObb13.2.1** La libreria deve permettere di scaricare un prodotto software;
 - RObb13.2.2** La libreria deve permettere di installare un prodotto software;
 - RObb13.2.3** La libreria deve permettere di aggiornare un prodotto software;

RObb13.2.4 La libreria deve permettere di scaricare e installare prerequisiti se l'installazione di un prodotto software lo richiede;

RObb13.3 La libreria deve fornire la possibilità di riavviare il programma ove ritenuto necessario;

RObb13.4 La libreria deve fornire la possibilità di gestire aggiornamenti in modo automatico;

RObb13.4.1 La libreria deve permettere il controllo automatico di aggiornamenti disponibili online;

RObb13.4.2 La libreria deve permettere il download automatico di aggiornamenti disponibili online;

RObb13.4.3 La libreria deve permettere l'installazione automatica di aggiornamenti disponibili online;

2.2.2 Lato Server

RObb14 Il server deve offrire un servizio di tipo REST

RObb14.1 Il servizio deve richiedere l'autenticazione tramite username e password;

RObb14.2 Il servizio deve fornire le operazioni di GET, POST, PUT e DELETE per i prodotti software interni al database;

RObb14.3 Il servizio deve fornire le operazioni di GET, POST, PUT e DELETE per i prerequisiti software interni al database;

RObb15 Il server deve offrire un sito web amministrativo;

RObb15.1 Il sito web deve offrire una pagina in cui effettuare il login;

RObb15.2 Il sito web deve offrire una pagina in cui sia possibile effettuare operazioni sui prodotti software all'interno del database;

RObb15.2.1 La pagina deve fornire una lista di tutti i prodotti all'interno del database e relative informazioni;

RObb15.2.2 La pagina deve fornire la possibilità di aggiungere un prodotto software nel database;

RObb15.2.3 La pagina deve fornire la possibilità di eliminare un prodotto software nel database;

RObb15.2.4 La pagina deve fornire la possibilità di modificare un prodotto software nel database;

RObb15.3 Il sito web deve offrire una pagina in cui sia possibile effettuare operazioni sui prerequisiti software all'interno del database;

RObb15.3.1 La pagina deve fornire una lista di tutti i prerequisiti all'interno del database e relative informazioni;

RObb15.3.2 La pagina deve fornire la possibilità di aggiungere un prerequisiti software nel database;

RObb15.3.3 La pagina deve fornire la possibilità di eliminare un prerequisiti software nel database;

RObb15.3.4 La pagina deve fornire la possibilità di modificare un prerequisiti software nel database;

RObb15.4 Il sito deve offrire una pagina in cui sia possibile gestire la relazione tra prodotti software e prerequisiti;

RObb15.4.1 La pagina deve fornire la possibilità di selezionare un prodotto software;

RObb15.4.2 La pagina deve fornire la possibilità di associare i prerequisiti disponibili nel database al prodotto software selezionato;

RObb15.4.3 La pagina deve fornire la possibilità di dissociare i prerequisiti disponibili nel database al prodotto software selezionato;

ROpz16 Il sistema di countnuos integration dell'azienda dovrà essere collegato al servizio REST consentendo la PUT automatica dei prodotti software presenti nel database;

2.3 Requisiti di qualità

Per quanto concerne la qualità è stato espressamente richiesto di sviluppare il sistema nel modo più modulare possibile così da poter favorire il riuso delle componenti. Da ciò si è deciso di sviluppare una libreria di classi incaricate di effettuare la comunicazione con il servizio REST e delle componenti grafiche indipendenti dal programma client.

2.3.1 Soddisfacimento

Per soddisfare i requisiti di qualità il progetto è scomposto in due parti suddivise a sua volta in piccoli progetti: macro componenti. Inoltre anche se non richiesto, in vista del riuso, molto probabile, del codice da parte di sviluppatori esterni allo sviluppo, l'intero sistema è stato sviluppato cercando di seguire le linee guida riportate sui seguenti libri di testo:

- *Clean Code* di Robert C. Martin [9];
- *Implementation Patterns* di Kent Beck [2].

Capitolo 3

Tecnologie adottate

3.1 Parte Client

3.1.1 Linguaggi

Il linguaggio usato per realizzare la parte client del sistema Updater è il C#. Il C# è un linguaggio di programmazione orientato agli oggetti puro, sviluppato da Microsoft all'interno dell'iniziativa .NET. Nel 2001 è divenuto standard ECMA e successivamente, nel 2003, anche standard ISO (ISO/IEC 23270).

L'utilizzo di questo linguaggio è stato espressamente richiesto dal committente, l'azienda è infatti in possesso delle licenze per l'utilizzo di tutte le tecnologie Microsoft per scopi commerciali.

3.1.2 Tecnologie esterne

3.1.2.1 .NET Framework

.NET è il framework che accompagna il linguaggio C# e altri linguaggi. Gestisce l'ambiente di esecuzione, il Common Language Runtime (CLR), in cui vengono eseguiti i programmi precompilati in un linguaggio intermedio. Inoltre mantiene una libreria di classi utilizzabili nelle proprie applicazioni e numerose altre tecnologie.

Versione utilizzata: 4.5.1

3.1.2.2 Windows Presentation Foundation - WPF

Windows Presentation Foundation è una tecnologia all'interno del .NET Framework. Essa offre la base operativa per l'esecuzione di applicazioni Windows con interfaccia grafica e riesce a rendere l'interfaccia grafica il più possibile indipendente dal codice logico. Per costruire un'interfaccia grafica infatti si utilizza un linguaggio ad hoc: l'Extensible Application Markup

Language (XAML) un linguaggio di markup basato su XML che permette la costruzione di interfacce grafiche senza la conoscenza di un linguaggio di programmazione.

Versione utilizzata

4.5.1

Applicazione

Viene utilizzato per la creazione del programma Updater e per le componenti grafiche;

Vantaggi

Rende facile la divisione tra parte grafica e parte logica grazie all'uso del meccanismo di *data-binding*;

Svantaggi

Richiede l'apprendimento di un nuovo linguaggio con una curva di apprendimento bassa;

3.1.2.3 Catel

Piattaforma di sviluppo di applicazioni che vogliono seguire il pattern Model View ViewModel (MVVM) e Model View Controller (MVC). Essa racchiude una serie di classi e funzionalità che facilitano l'implementazione dei pattern architetturali.

Versione utilizzata

4.4.0

Applicazione

Sono sfruttate alcune classi base di Catel e attributi speciali per implementare il pattern MVVM nel programma Updater e per utilizzare la serializzazione su file XML eseguita dalle componenti della Libreria Updater;

Vantaggi

Riduzione del codice scritto con conseguente riduzione di possibili errori;

Svantaggi

Aumento della complessità dell'intero sistema.

3.1.2.4 Command Line Parser

Libreria semplice ed efficace per la manipolazione di argomenti per comandi da terminale e le relative attività. Essa offre un parser di comandi da terminale integrato.

Versione utilizzata

1.9.71.2

Applicazione

La libreria garantisce all'avvio dell'applicazione di leggere e interpretare eventuali operazioni pendenti dopo la sua chiusura. In particolare è utilizzata per gestire l'installazione di prodotti software che necessitano i permessi di amministratore e quindi il riavvio del programma;

Vantaggi

Semplice da usare ed efficace;

Svantaggi

Poca documentazione disponibile.

3.1.2.5 MahApps.Metro

MahApps.Metro è un insieme di componenti grafiche per WPF che seguono il design *modern UI*, conosciuto anche come *Metro*, creato da Microsoft.

Versione utilizzata

1.2.4

Applicazione

Viene utilizzato per far apparire la finestra principale del programma con il design moderno di oggi;

Vantaggi

Fornisce un insieme di componenti grafici dal design moderno già creati;

3.1.2.6 WPF NotifyIcon

Implementazione di una icona nella taskbar del sistema operativo windows per WPF che fornisce la possibilità di mostrare avvisi *balloon*, popup e context menu.

Versione utilizzata

1.0.8

Applicazione

Creare la notifyicon del programma Updater consentendo al programma di comunicare all'utente anche se in background;

Vantaggi

Fornisce componenti grafici già creati e facili da riutilizzare;

3.1.3 Tecnologie interne

In questa sezione si elencano le tecnologie offerte all'interno dell'azienda, ossia librerie di componenti comuni.

3.1.3.1 Salvagnini UI Catel Controls**Applicazione**

Le componenti offerte hanno facilitato l'implementazione del pattern Model View Viewmodel tramite l'uso dell'ereditarietà nelle nuove componenti definite;

Vantaggi

Riuso di componenti già definite e risparmio di codice;

Svantaggi

Nuova dipendenza del codice sviluppato.

3.1.3.2 Salvagnini UI Controls**Applicazione**

Uso di alcune componenti grafiche già definite in XAML per la costruzione della finestra principale e della schermata delle impostazioni;

Vantaggi

Riuso di componenti già definite;

Svantaggi

Nuova dipendenza del codice sviluppato.

3.2 Parte Server

3.2.1 Linguaggi

Per la creazione del sistema REST si sono utilizzati ancora il linguaggio C# e l'ambiente .NET. Mentre per la creazione del sito web si sono utilizzati i linguaggi affermati nello sviluppo di applicazioni web quali HTML5, CSS e Javascript.

3.2.2 Tecnologie esterne

3.2.2.1 ASP.NET Web API

ASP.NET è un web framework open source per costruire applicazioni web e servizi web. Esso permette in modo rapido e semplice la costruzione di un servizio REST.

Versione utilizzata

2.2

Applicazione

Creare il servizio REST per comunicare i dati attraverso la rete Internet;

Vantaggi

Framework appartenente all'insieme .NET che permette in modo veloce e semplice di costruirsi una propria Web API basandosi sul pattern MVC;

Svantaggi

Troppo vincolato al framework .NET e alle tecnologie Microsoft.

3.2.2.2 Entity Framework

Framework Object relational mapping (ORM) all'interno delle tecnologie di .NET. Esso consente di creare una comunicazione automatica tra una base di dati relazionale e un programma orientato agli oggetti.

Versione utilizzata

5

Applicazione

Creazione delle componenti che rappresentano sotto forma di oggetti le informazioni all'interno della base di dati;

Vantaggi

Modello già creato per la comunicazione con la base di dati. Recupero delle informazioni nella base di dati attraverso l'uso di programmazione funzionale usando Linq invece di SQL;

Svantaggi

Inefficiente per basi di dati enormi.

3.2.2.3 Devart Entity Developer

Designer automatico per la creazione di modelli Entity Framework ORM a partire dalla definizione della base di dati in SQL o viceversa.

Versione utilizzata

6.0.11

Applicazione

Costruzione automatica delle componenti Entity Framework;

Vantaggi

Automatizzazione per la codifica di numerose classi;

Svantaggi

Strumento a pagamento.

3.2.2.4 Internet Information Server

L'Internet Information Server è un insieme di servizi server internet per sistemi operativi Windows, creato dalla Microsoft

Versione utilizzata

8.0

Applicazione

Ospita il database, l'esecuzione della Web API e il sito web;

Vantaggi

Completamente integrato nella piattaforma di sviluppo Visual Studio e facilmente configurabile;

Svantaggi

Sistema a pagamento supportato solo su sistemi operativi Windows.

3.2.2.5 PostgreSQL

PostgreSQL è un database management system (DBMS) ossia un sistema software progettato per la creazione e manipolazione e l'interrogazione di una base di dati di tipo relazionale basato sul linguaggio SQL.

Versione utilizzata

9.5.4

Applicazione

Utilizzato per la creazione e gestione della base di dati necessaria per l'intero sistema Updater;

Vantaggi

Potente DBMS che supporta quasi la totalità dei costrutti SQL ed è Open Source;

3.2.2.6 PowerShell

Linguaggio di scripting dell'omonimo terminale disponibile nei sistemi operativi Windows, sviluppato da Microsoft. Basato sulla programmazione ad oggetti e sul framework .NET.

Versione utilizzata

5

Applicazione

Creazione dello script che mette in relazione il sistema di Continuous Integration aziendale con il sistema Updater;

Vantaggi Linguaggio di scripting che può usufruire di tutte le potenti librerie offerte da .NET;

Svantaggi Opera solo in ambiente con sistema operativo Windows.

3.2.2.7 Bootstrap

Framework per lo sviluppo di siti web responsive e mobile-first. Esso offre un insieme di componenti per il web create con HTML, CSS e Javascript.

Versione utilizzata

3.3.7

Applicazione

Costruzione del sito web amministrativo della base di dati;

Vantaggi

Permette la creazione di un sito web responsive dal design moderno in pochissimo tempo;

Svantaggi

Appesantisce il sito web, molte componenti non utilizzate sono comunque comprese.

3.2.2.8 JQuery

Libreria Javascript che ne estende le funzionalità. Essa fornisce una facile manipolazione di una pagina HTML, fornisce la possibilità di creare animazioni, inoltre fornisce potenti strumenti per utilizzare la tecnica Asynchronous JavaScript and XML (AJAX).

Versione utilizzata

3.0.0

Applicazione

Utilizzata per eseguire le richieste HTTP con la tecnica AJAX e manipolare gli elementi delle pagine web;

Vantaggi

Semplifica l'invio di richieste HTTP e la manipolazione delle pagine web usando Javascript. Documentazione eccellente;

Svantaggi

Molto spesso alcune funzioni non sono supportate in tutti i Browser.

3.2.2.9 KnockoutJs

Libreria Javascript che semplifica l'implementazione del pattern MVVM in una pagina web.

Versione utilizzata

3.4.0

Applicazione

Utilizzata per l'implementazione del data binding tra parte grafica di una pagina web e la sua parte logica;

Vantaggi

Libreria open source che permette facilmente il data binding tra elementi HTML (parte grafica) e oggetti Javascript (parte logica);

Svantaggi

La libreria non forza gli sviluppatori a seguire una struttura ben organizzata del codice Javascript prodotto.

Capitolo 4

Architettura

4.1 Introduzione

Per poter gestire meglio lo sviluppo, l'intero sistema è stato suddiviso in due parti visibili concettualmente e concretizzate in *solution* (soluzioni), configurazioni definite dall'ambiente di sviluppo Visual Studio. Esse sono:

- Updater Client;
- Updater Server.

Ad esse sono stati associati i vari progetti definiti all'interno dei seguenti namespace:

Updater: racchiude le componenti principali del programma che fornisce le funzionalità di installazione e aggiornamento dei prodotti;

UIControls: raggruppa le componenti grafiche che seguono il pattern MV-VM (quindi per ogni componente grafica, view, abbiamo un corrispondente ViewModel e Model);

UpdaterClientLibrary: contiene le componenti che incapsulano una corretta comunicazione con il servizio REST e effettuano le operazioni per scaricare, installare e aggiornare un prodotto software;

UpdaterSerialization: contiene le componenti che racchiudono i dati trasmessi tra server e client, quindi serializzati.

UpdaterServer: contiene le componenti necessarie per il funzionamento del servizio RESTful;

UpdaterDBEF: (DBEF = DataBase Entity Framework) contiene le componenti che implementano la tecnica ORM la quale consente una facile relazione tra le componenti orientate agli oggetti e il database relazionale.

Di seguito l'associazione delle varie macro componenti con le *solution*:

Updater Client composto dalle macro componenti:

- Updater;
- UpdaterUIControls;
- UpdaterClientLibrary;
- UpdaterSerialization.

Updater Server composto dalle macro componenti:

- UpdaterSerialization;
- UpdaterServer;
- UpdaterDBEF.

Come è possibile notare, le componenti in UpdaterSerialization sono condivise in entrambe le *solution* ciò perché esse permettono lo scambio di informazioni tra client e server.

4.2 Architettura generale

L'intero sistema segue una struttura a livelli (figura 4.1) in cui la dipendenza tra le macro componenti segue un flusso verso il basso.

Comunicazione tra componenti

La comunicazione avviene principalmente solo tra macro componenti vicine. In particolare la macro componente Updater Client Library rende indipendente le altre componenti client dalla comunicazione attraverso la rete e l'uso delle richieste HyperText Transfer Protocol (HTTP) (figura 4.2).

Architettura - Parte client

Per la parte client si è seguito il pattern Model View ViewModel (MVVM) incoraggiato dalla tecnologia WPF e rafforzato dall'uso del framework di supporto Catel. Quest'ultimo mette a disposizione tag, attributi e classi che permettono una gestione più semplice delle tre parti: View, ViewModel e Model. Ogni componente grafico avrà il suo corrispettivo ViewModel (logica e dati utili al componente View) e se necessario anche il Model (se in possesso di dati persistenti, per esempio prelevati dal database).

Architettura - Parte server

Per la parte server invece si utilizza il framework ASP.NET il quale attraverso la definizione di classi **controller** permette la costruzione di un servizio RESTful. La comunicazione tra Web API e database avviene grazie ad Entity Framework il quale implementa la tecnica di programmazione Object-Relational Mapping (ORM).

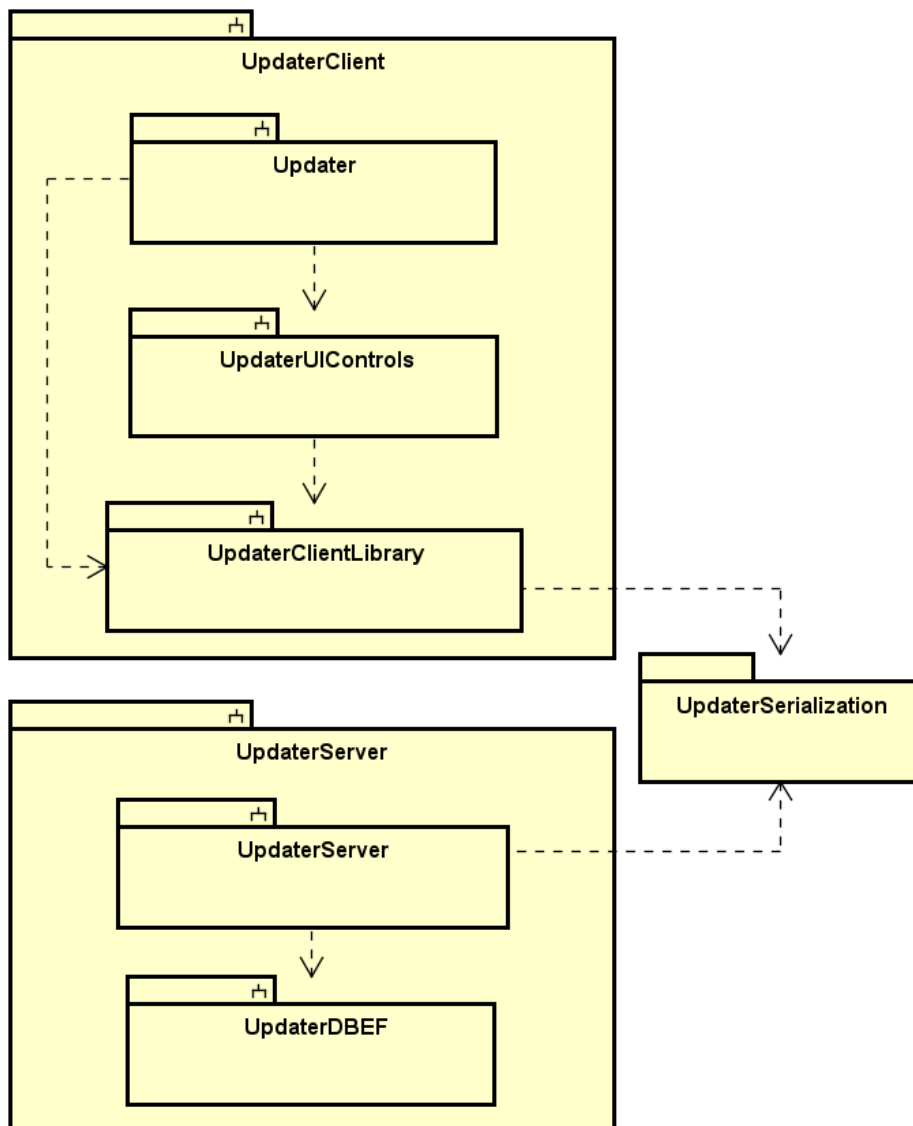


Figura 4.1: Dipendenze tra macro componenti del sistema Updater

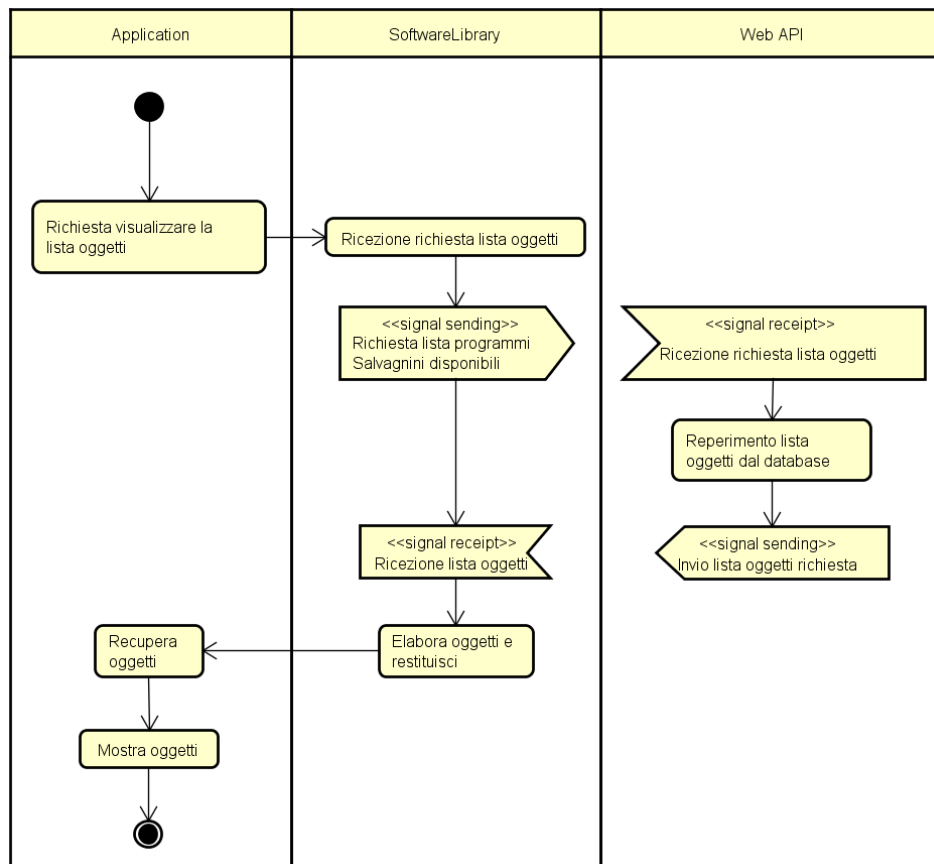


Figura 4.2: Esempio generico di comunicazione tra parte client e parte server

4.3 Macro componenti

Nella presente sezione vengono descritte in dettaglio le macro componenti riportando per ciascuna una descrizione esauriente e la struttura con le componenti interne e le relazioni tra esse. Dove necessario vengono discusse le scelte e le possibili criticità cercando di proporre soluzioni migliori e migliori che in futuro potrebbero essere implementate.

4.3.1 Updater

4.3.1.1 Descrizione

Identifica un programma eseguibile lato client composto principalmente da un'interfaccia grafica per agevolare l'utente nel download, nell'installazione o nell'aggiornamento dei prodotti software e relativi prerequisiti.

4.3.1.2 Responsabilità

- Gestire l'avvio dell'applicazione WPF;
- Gestire eventuali parametri all'avvio dell'applicazione;
- Gestire la finestra principale dell'applicazione;
- Gestire la finestra delle impostazioni dell'applicazione;
- Gestire l'icona di notifica nella *TaskBar*.

4.3.1.3 Implementazione

Dipendenze esterne

- WPF;
- Catel;
- MahApps;
- Command Line Parser.

Dipendenze interne

- Salvagnini.UICatelControls;
- Salvagnini.UIControls.

Il programma si basa sul framework Windows Presentation Foundation il quale mette a disposizione la classe **Application** singleton e primo oggetto istanziato all'esecuzione del programma. **Application** è quindi stata ereditata dalla classe **App** la quale è responsabile di:

- creare la finestra principale e la schermata secondaria delle impostazioni utente;
- creare la Notify Icon che comparirà nella barra dello start (*TaskBar*) e della comparsa;
- creare avvisi (*balloon*) in risposta al completamento corretto o in errore delle operazioni automatiche.

La finestra principale è identificata dalla classe **MainWindows**, estende la classe **MetroWindow** distribuita dall'UI toolkit *MahApp* il quale applica alla finestra lo stile delle moderne applicazioni windows aderenti al design *Metro*. Una finestra secondaria è rappresentata dalla classe **SettingsView** la quale si occupa di gestire le componenti grafiche per le preferenze utente dell'intera applicazione, tali preferenze (o impostazioni) impatteranno nel funzionamento della **TaskBarIcon**. In se il progetto è costituito solo dalla finestra principale e dalla gestione dell'icona nella taskbar del sistema. Esso contiene prettamente componenti grafiche e la logica di base di esse e dell'avvio dell'applicazione (implicito grazie all'uso di **Application**).

4.3.1.4 Funzionamento

La struttura segue il pattern MVVM come si vede in figura 4.3.1.5. **App** è una classe singleton e la prima ad essere istanziata dal framework WPF, da questa il programma inizializza la **MainWindow** e tutte le componenti grafiche (View) coinvolte insieme ai rispettivi ViewModel e Model. Le classi interne al namespace **Views** sono costituite da una parte descritta in XAML (descrive l'aspetto prettamente grafico) e una in C# (il *code behind*). La classe **Options** usufruisce della classe **IParserState** della libreria *Command Line Parser* consentendo all'applicazione di essere eseguita con alcuni parametri utili per gli eventuali riavvi necessari nell'installazione di qualche prodotto software.

4.3.1.5 Criticità e possibili miglioramenti

Il pattern architetturale MVVM garantisce una certa separazione tra le componenti e l'uso del framework Catel rende le comunicazioni tra View e ViewModel molto chiare e semplici, anche tra diversi ViewModel grazie all'attributo **[InterestedIn]**. L'uso della *TaskBarIcon* costringe all'inserimento di codice di logica (e quindi non riguardante la grafica) nel *code behind* della classe **App** ciò oltre a non essere buona pratica dà una nuova responsabilità

di più basso livello (creazione dei *balloon*) rispetto alla creazione delle finestre della classe `App`¹.

Nel caso in cui ci sia la necessità di aggiungere nuove responsabilità per la classe `App` è necessario un redesign per spostare il codice che coinvolge la classe `TaskBarIcon` ad una nuova classe.

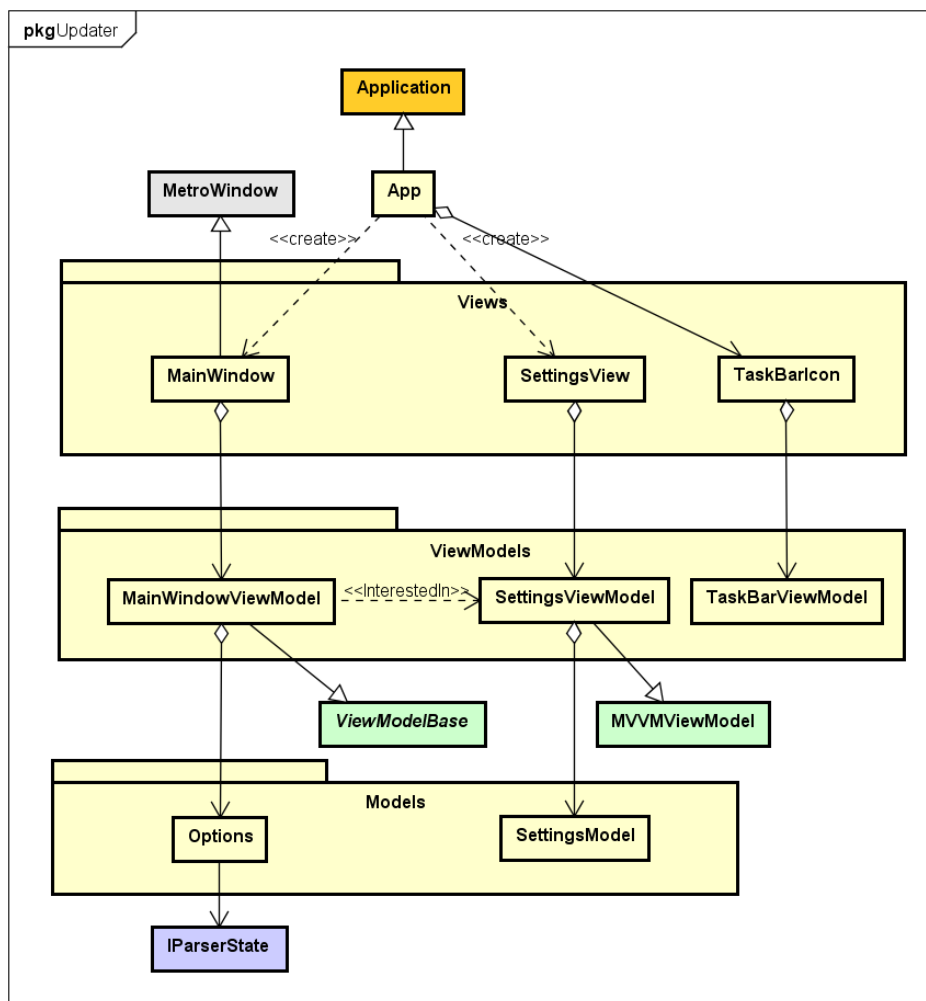


Figura 4.3: Struttura macro componente Updater

¹[2] Responsabilità di diverso livello rompono il *principio di simmetria* individuato da Kent Beck in *Implementation Pattern*, cap. 3, pag. 15.

4.3.2 Updater User Interface Controls

4.3.2.1 Descrizione

Identifica un insieme di componenti grafici utilizzabili da software esterno per costruire un'interfaccia grafica. Tali componenti sono utilizzate da Updater e potranno essere utilizzate da altri prodotti software.

4.3.2.2 Responsabilità

- Fornire le componenti grafiche principali che consentono all'utente di visualizzare, scaricare, installare, reinstallare e aggiornare un prodotto software.

4.3.2.3 Implementazione

Dipendenze esterne

- WPF;
- Catel;
- MahApps.

Dipendenze interne

- Salvagnini.UICatelControls;
- Salvagnini.UIControls.

La macro componente è costituita da tre elementi grafici principali:

1. ProductsPanel;
2. ProductControl;
3. ProductInfo.

Il primo rappresenta una lista degli elementi grafici `ProductControl` e offre vari filtri da applicare ad essa compresa una ricerca per nome. Il secondo rappresenta un singolo prodotto software e fornisce i pulsanti per effettuare le operazioni di installazione, reinstallazione, download e aggiornamento. Il terzo è legato al primo con l'attributo offerto da Catel `[InterestedIn]`, in base alla selezione di un elemento della lista questo mostra informazioni più dettagliate su di esso. Tutti gli elementi implementano il pattern MVVM pertanto sono scomposti in tre parti. La parte prettamente visiva (*View*) è costruita in XAML e C# mentre le altre in puro C#.

4.3.2.4 Funzionamento

WPF istanzia le classi View. `ProductInfoView` istanzia il proprio Model, ossia `ProductInfoViewModel`.

Per `ProductsPanelView` invece il comportamento è diverso (figura 4.3.2.4). Se il proprio `ProductsPanelViewModel` non è già istanziato prima costruisce `ProductsPanelModel`, poi `ProductsPanelViewModel`. Dopo la creazione di `ProductsPanelModel`, `ProductsPanelViewModel` incarica `ProductsPanelModel` di eseguire una richiesta HTTP per recuperare le informazioni sui prodotti software.

Viene quindi istanziato un `ProductControlModel` per ogni prodotto software. Terminata questa operazione, `ProductsPanelViewModel` può istanziare per ogni `ProductControlModel` un rispettivo `ProductControlViewModel`. Infine grazie al meccanismo del `DataContext` di WPF la classe `ProductControlView` può recuperare il rispettivo `ProductControlViewModel`.

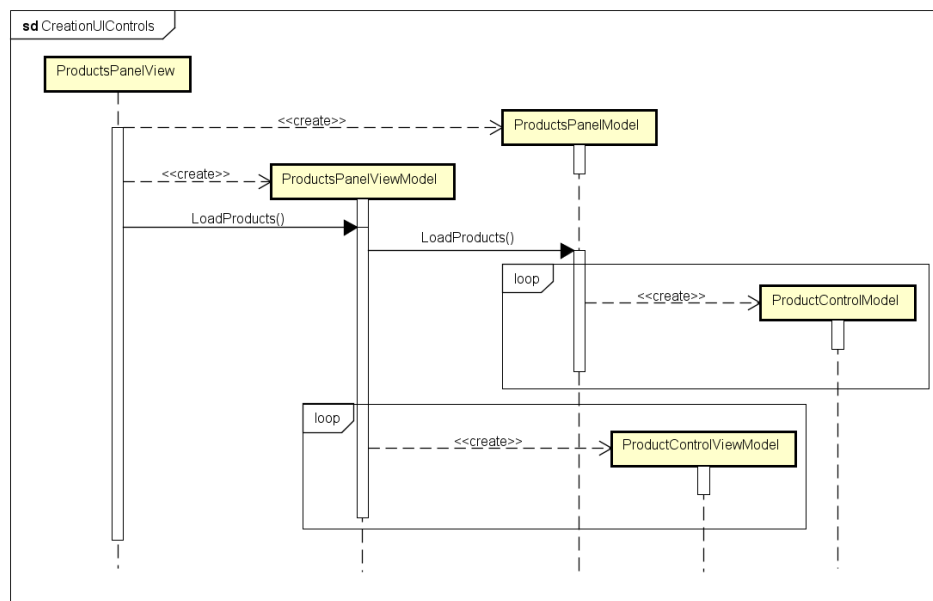


Figura 4.4: Diagramma di sequenza per la costruzione degli elementi grafici UIControls

4.3.2.5 Criticità e possibili miglioramenti

Il macro componente presenta confusione nell'uso di `DataContext`, inoltre le responsabilità sono confuse con il risultato di un necessario redesign. Infatti prima vengono costruiti i vari `ProductControlModel` e in seguito i `ProductControlViewModel` mancando di coesione e incrementando la com-

plexità della struttura. Tutto ciò rende ancor più difficile la comprensione delle componenti e quindi la modifica futura.

La classe `ProductControlModel` racchiude in sé logica delle componenti di *Updater Client Library* poiché all'interno di queste operazioni sono necessari dei feedback che impattano nell'interfaccia grafica, inoltre è possibile che durante l'operazione sia richiesto il riavvio dell'app. Un'alternativa forse più valida sposterebbe la logica nella libreria definendo la possibilità che essa lanci degli eventi indicanti lo stato delle operazioni e a questi `ProductControlModel` reagisca opportunamente. Questa alternativa non è stata implementata per evitare la complessità nella gestione della concorrenza. Procedendo così infatti si verrebbe a creare una situazione in cui due differenti thread devono comunicare tra di loro.

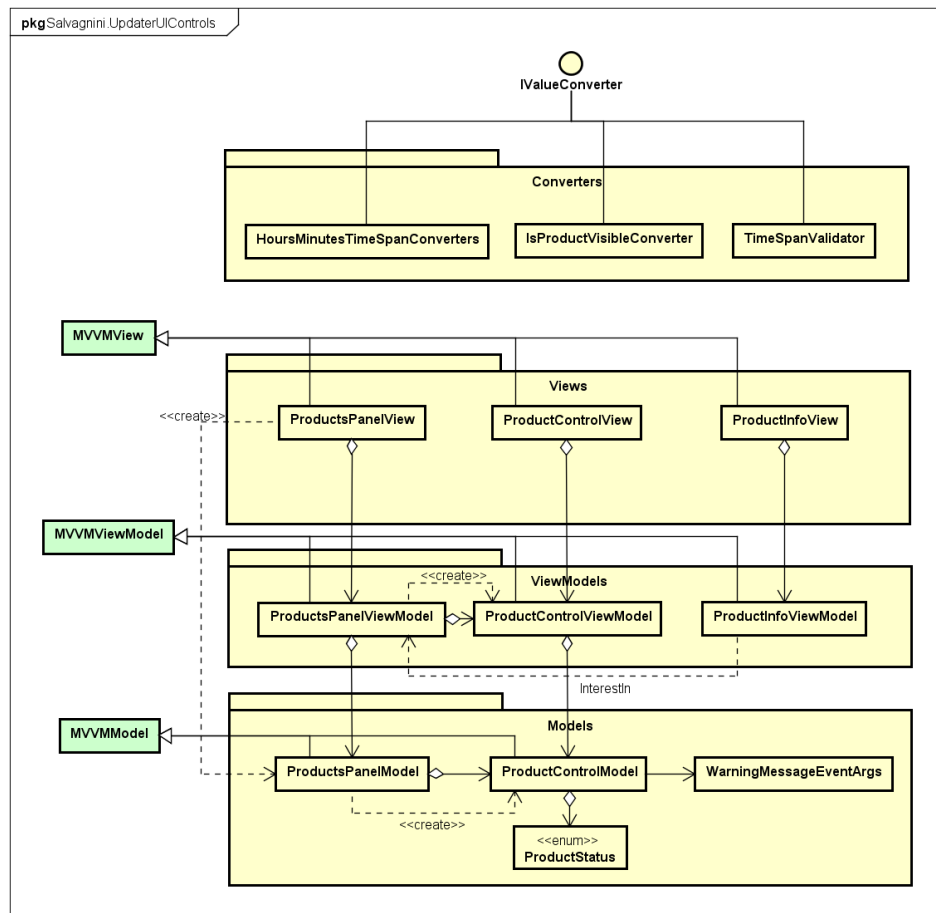


Figura 4.5: Struttura macro componente Updater UIControls

4.3.3 Updater Client Library

4.3.3.1 Descrizione

Identifica un insieme di componenti software incaricate di effettuare delle operazioni di controllo locali (per es. controllare la presenza dei programmi Salvagnini) e di gestire le comunicazioni con la Web API tramite servizio REST e il formato dati JSON. Tali componenti sono utilizzate dalle macro componenti *Updater* e *UpdaterUIControls*. Le componenti sono state rese indipendenti con lo scopo di poterle riutilizzare in altri prodotti software.

4.3.3.2 Responsabilità

- Incapsulare la comunicazione con il servizio REST;
- Consentire di effettuare su un prodotto software le seguenti operazioni:
 - Download;
 - Installazione;
 - Reinstallazione;
 - Aggiornamento;
- Consentire l'esecuzione automatica delle operazioni elencate precedentemente;
- Fornire gli strumenti per il corretto riavvio dell'applicazione;
- Fornire gli strumenti per il salvataggio su file XML delle operazioni effettuate (per es. salvare la data di installazione di un prodotto).

4.3.3.3 Implementazione

Dipendenze esterne

- Catel.

Dipendenze interne

- Updater Serialization.

La libreria è costituita principalmente da tre parti:

1. Dalle componenti che si occupano di comunicare con il servizio REST e di effettuare le operazioni sui prodotti software;
2. Dalle componenti necessarie al salvataggio di dati in formato XML;
3. Dalle componenti che si occupano di automatizzare le operazioni;

La prima parte è costituita da `ProductWrapper` e `PrerequisiteWrapper` le quali rappresentano un'incapsulazione dei dati deserializzati ricevuti dal servizio REST grazie alla classe `UpdaterService`. Le componenti `Wrapper` incapsulano tutte le operazioni che è possibile effettuare su un prodotto software e sui prerequisiti software. La seconda parte è costituita dalle classi all'interno del namespace `Models`. La classe singleton `ObjectManager` è incaricata di gestire il caricamento e il salvataggio dei dati resi persistenti in un file XML. Questi sono rappresentati dalle classi `Set` che racchiudono una struttura dati contenente oggetti contenenti i dati effettivi da salvare.

La terza parte è costituita da una gerarchia, le classi concrete di tale gerarchia sono incaricate di lanciare le corrette operazioni automatiche di download, installazione o aggiornamento. Questo sulla base delle preferenze impostate dall'utente e attraverso l'uso di timer. La classe `AppRestarter` racchiude la logica per il corretto riavvio dell'applicazione nel caso un'operazione automatica lo richiedesse.

4.3.3.4 Funzionamento

La libreria mette a disposizione delle classi involucro, `ProductWrapper` e `PrerequisiteWrapper`, che consentono di effettuare le operazioni sui prodotti software. Queste stesse classi `Wrapper` si occupano di rendere persistenti i dati delle operazioni svolte attraverso l'uso delle componenti interne al namespace `Models`. Oltre alle classi `Wrapper` la libreria mette a disposizione le classi `AutomaticUpdates`, esse basano il proprio funzionamento in base ad un timer interno, alla loro creazione il timer parte e allo scadere la classe effettuerà l'operazione per cui è incaricata in modo completamente automatico (da qui la dipendenza tra `AutomaticUpdatesController` e `ProductWrapper`), il timer quindi riparte. Ogni qual volta il timer scade le classi `AutomaticUpdates` lanciano un evento `ProductToUpdateEventArgs` in questo modo sfruttando il pattern Observer² insito nel linguaggio C# è possibile reagire allo scadere del timer anche in componenti esterne.

4.3.3.5 Criticità e possibili miglioramenti

Dipendenza circolare con `ProductWrapper` e `UpdateService`, a causa della scelta della *lazy initialization* per i `PrerequisiteWrapper`. `UpdaterService` potrebbe contenere metodi statici per risolvere oppure la *lazy initialization* potrebbe essere eliminata per rimuovere la dipendenza. Manca qualche namespace che organizzerebbe meglio la libreria, ad esempio `AutomaticUpdates` potrebbe essere un nuovo namespace per la gerarchia di classi strettamente connesse. `ProductWrapper` contiene codice duplicato con `AppRestarter` come conseguenza di un refactoring, oltre ad assumersi un numero elevato

²[7] Gang of Four. Design Patterns: Elements of Reusable Object-Oriented Software, cap. 5, pag. 326.

di responsabilità che suggerisce di spezzare la classe in più classi. La scelta di contenere tutti in una è data dal fatto che le operazioni necessarie erano ben definite (download, install e update) e si suppone che in futuro non ne servano altre. Nel caso contrario si potrebbe pensare a rappresentare ogni operazione come una classe implementando così il Command pattern³.

Le componenti incaricate della deserializzazione e serializzazione dal file XML sono spesso ridondanti, l'uso di classi generiche consentirebbe un'ulteriore flessibilità al sistema evitando di dover ad ogni aggiunta di dati da rendere persistenti la codifica di ulteriori classi praticamente identiche. L'uso di classi generiche consentirebbe anche al singleton `ObjectsManager` di acquisire maggior flessibilità. Attualmente questa operazione non è stata fatta per limiti tecnologici e conoscitivi del framework Catel, si è sperimentato infatti che non è possibile rendere generica una classe che estende `ModelBase`, probabilmente il limite deriva dal fatto che i tipi parametrici contano solo nella precompilazione, di fatto poi tutto è tipizzato con la classe `Object` di C# in questo modo il meccanismo di serializzazione non può più distinguere una classe generica istanziata con un tipo da un'altra istanziata con tipo diverso.

³[7] Gang of Four. Design Patterns: Elements of Reusable Object-Oriented Software, cap. 5, pag. 263.

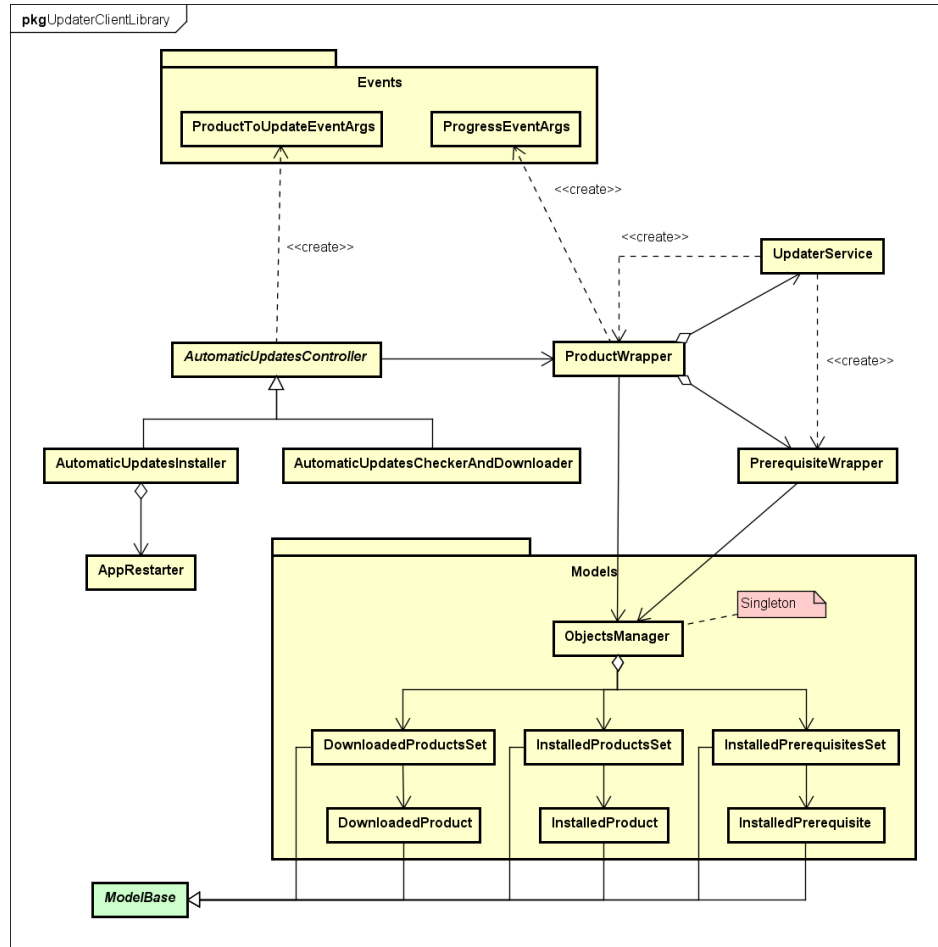


Figura 4.6: Struttura macro componente Updater Client Library

4.3.4 Updater Serialization

4.3.4.1 Descrizione

Identifica l'insieme di definizioni di oggetti che permettono la trasmissione di dati in una rete tra server e client tramite serializzazione.

4.3.4.2 Responsabilità

- Definire oggetti serializzabili contenenti dati.

4.3.4.3 Implementazione

Dipendenze esterne

Nessuna.

Dipendenze interne

Nessuna.

Le classi che terminano con il suffisso `Object` sono definizioni di oggetti che contengono solo campi dati e costruttori. I campi dati sono etichettati con l'attributo `[DataMember]` in tal modo il framework .NET è capace di trasformarle in formato JSON così da poter trasmettere informazioni attraverso una rete. La classe `Serializer` si occupa di facilitare tale serializzazione attraverso la definizione di metodi statici per la lettura e scrittura di dati serializzati.

4.3.4.4 Funzionamento

L'unica classe del namespace che racchiude logica è `Serializer`, la quale semplicemente semplifica la serializzazione e la deserializzazione definendo due metodi statici. Le altre componenti non racchiudono logica al loro interno. Nessuna componente ha dipendenze esterne per questo motivo si omette il diagramma della struttura del macro componente.

4.3.4.5 Criticità e possibili miglioramenti

L'unico punto critico sono i costruttori delle classi `Object` serializzabili. Nel caso infatti si dovesse aggiungere un campo dati si è costretti a definire un nuovo costruttore e definire un valore di default per il campo dati aggiunto. Nel caso questo non possa avvenire, tutte le dipendenze verso la classe necessiterebbero di essere modificate. Per far sì che questo sforzo di modifiche sia limitato le dipendenze verso questo macro componente sono **esclusivamente** delle macro componenti *Updater Client Library* e *Updater Server* e così dovrebbe essere mantenuto in future estensioni.

4.3.5 Updater Server

4.3.5.1 Descrizione

Identifica il server con sistema IIS su cui opera la web API supportata da un database relazionale che implementerà un servizio REST col quale comunicheranno il macro componente *Updater Client Library* e le pagine web amministrative del database).

4.3.5.2 Responsabilità

- Fornire un servizio REST per trasmettere i dati dei prodotti software e dei relativi prerequisiti software prelevati da un database;
- Fornire delle pagine web amministrative per facilitare le modifiche ai dati contenuti nel database.

4.3.5.3 Implementazione

Dipendenze esterne

- ASP.NET;
- JQuery;
- Bootstrap;
- KnockoutJS.

Dipendenze interne

- Updater Serialization;
- Updater Entity Framework Database (EFDB).

Per la parte server del servizio REST è stata impostata la configurazione del servizio nelle componenti del namespace **App_Start** preconfigurate dal framework e per richiedere l'autenticazione alle richieste HTTP è stata creata la classe **BasicAuthenticationHandler**. Le componenti principali sviluppate sono le classi **Controller** ossia classi i cui metodi vengono invocati in base alla richiesta HTTP ricevuta. Le gerarchie **Putter** e **Poster** sono state sviluppate con lo scopo di supportare tali metodi. Il namespace **SerializableFactory** racchiude classi **Factory**⁴ le quali racchiudono metodi statici per la creazione di oggetti **Object** definiti in *Updater Serialization* a partire dagli oggetti distribuiti da *UpdaterDBEF*. Per quanto riguarda la

⁴[7] Gang of Four. Design Patterns: Elements of Reusable Object-Oriented Software, cap. 5, pag. 326.

parte web (omessa nel diagramma in figura 4.3.5.5) sono state create quattro pagine web in HTML5 decorate con l'aiuto del framework Bootstrap. Le pagine web interagiscono con il servizio REST tramite l'uso della tecnica Asynchronous JavaScript and XML (AJAX) resa disponibile dalla libreria JQuery (per approfondimento si veda la sezione 5.2.1).

4.3.5.4 Funzionamento

Una volta avviata la web api vengono caricate le dovute impostazioni codificate nelle classi all'interno di **App_Start**. A questo punto ad ogni richiesta http inviata al server agisce la classe **BasicAuthenticationHandler** che verifica la presenza di nome utente e password corretti nella richiesta. Una volta accertata la richiesta il flusso del programma è passato alle classi **Controller**, qui viene eseguito il codice del metodo che corrisponde all'uri della richiesta.

4.3.5.5 Criticità e possibili miglioramenti

Attualmente molti dati e informazioni di configurazione sono codificati direttamente mentre sarebbe molto più opportuno leggerli da file esterni garantendo così una maggiore flessibilità dell'intero sistema.

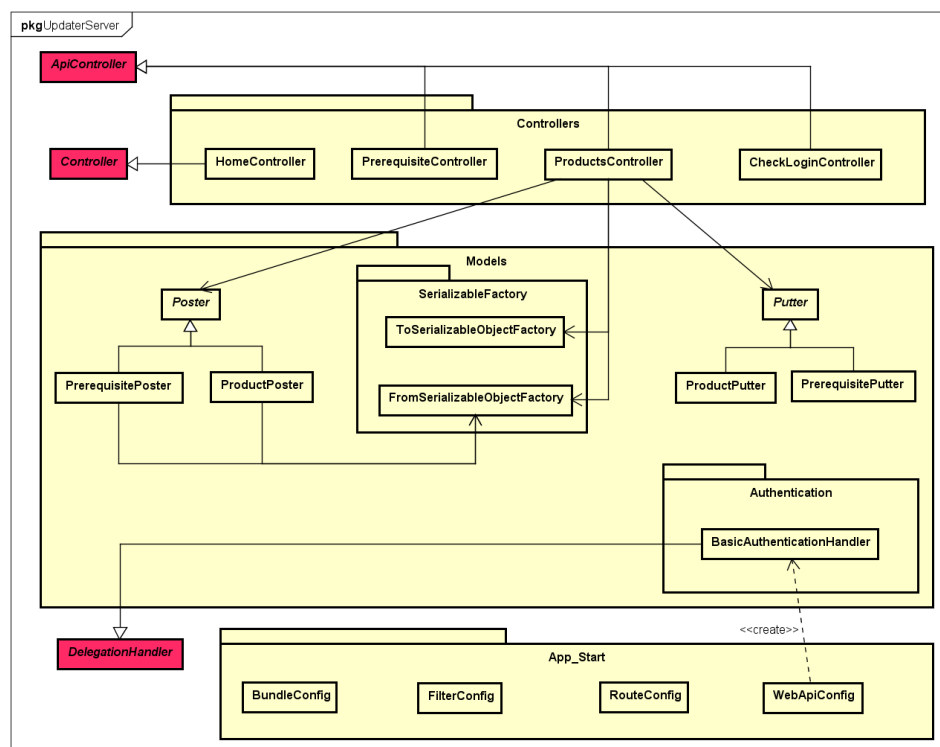


Figura 4.7: Struttura macro componente Updater Server

4.3.6 Updater Database Entity Framework (DBEF)

4.3.6.1 Descrizione

Identifica un'insieme di componenti incaricate di mettere in relazione il paradigma della programmazione orientata agli oggetti e la base di dati relazionale.

4.3.6.2 Responsabilità

- Fornire un accesso facile e semplice alla base di dati relazionale.

4.3.6.3 Implementazione

Dipendenze esterne

- Devart;
- Entity Framework.

Dipendenze interne

Nessuna.

L'utilizzo della tecnologia *Devart* ha permesso di costruire le componenti strutturate su Entity Framework in modo completamente automatico a partire dal modello SQL della base di dati.

4.3.6.4 Funzionamento

Le componenti create dalla tecnologia *Devart* consentono di usare la tecnologia Linq di .NET per poter usufruire attraverso programmazione funzionale di tutto il contenuto del database attraverso un'automatica conversione in oggetti.

4.3.6.5 Criticità e possibili miglioramenti

Le criticità che potrebbero emergere sono principalmente le problematiche derivanti dall'uso della tecnica ORM, colpevolizzata di non rispettare i principi della programmazione ad oggetti⁵. I vantaggi però del suo uso sono stati evidenti durante lo sviluppo. La tecnologia *Devart* permetteva l'aggiornamento in pochissimi passi a qualsiasi modifica nel modello della base di dati. L'uso di Entity Framework permette l'uso di Linq e quindi di accedere con una facilità disarmante ai dati nel database attraverso la programmazione funzionale e quindi codificando spesso una singola linea di codice.

⁵<http://www.yegor256.com/2014/12/01/orm-offensive-anti-pattern.html>
http://selldo.com/weblog/2011/08/11/orm_is_an_antipattern.

4.4 Relazioni esterne tra componenti

Nella presente sezione vengono mostrate le relazioni esterne tra le componenti delle macro componenti attraverso l'uso di diagrammi delle classi.

4.4.1 Relazioni Updater

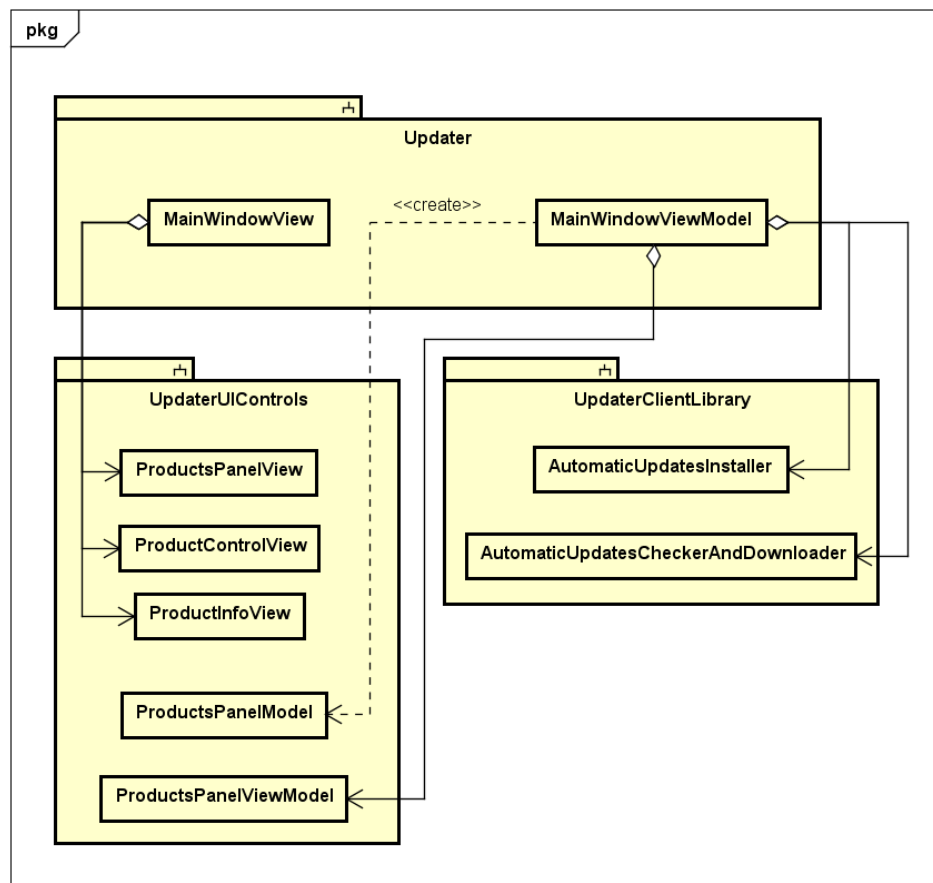


Figura 4.8: Relazioni componenti di Updater con le componenti di UIControls e Client Library

4.4.2 Relazioni Updater UIControls

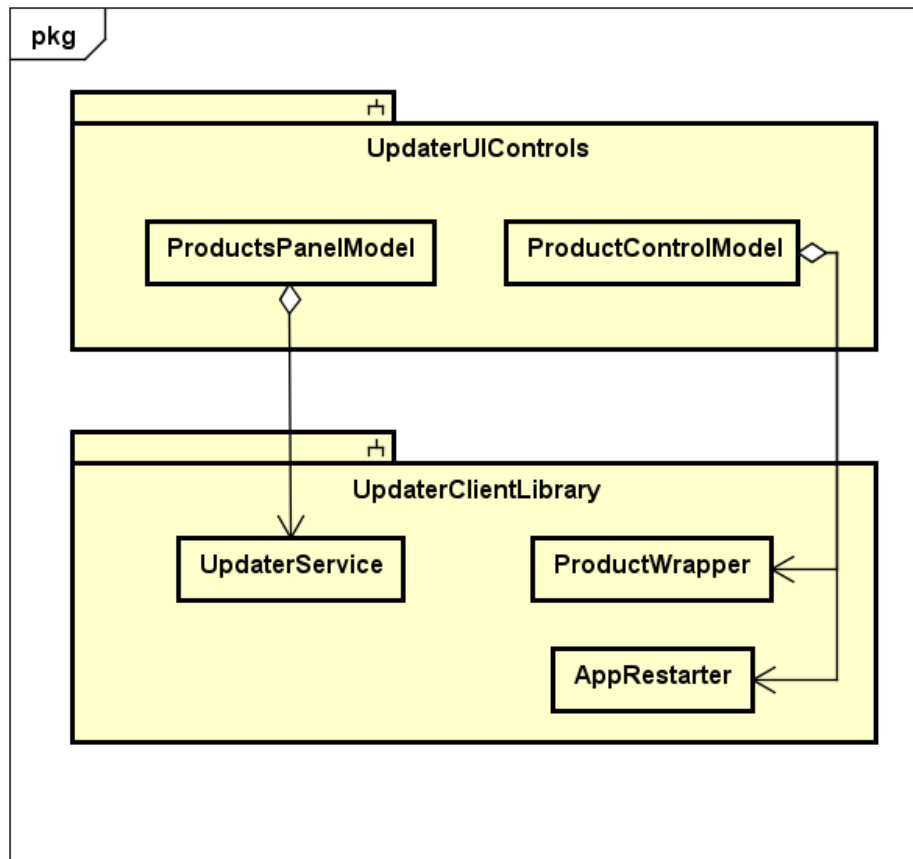


Figura 4.9: Relazioni componenti di Updater UIControls con le componenti di Client Library

4.4.3 Relazioni Updater Client Library

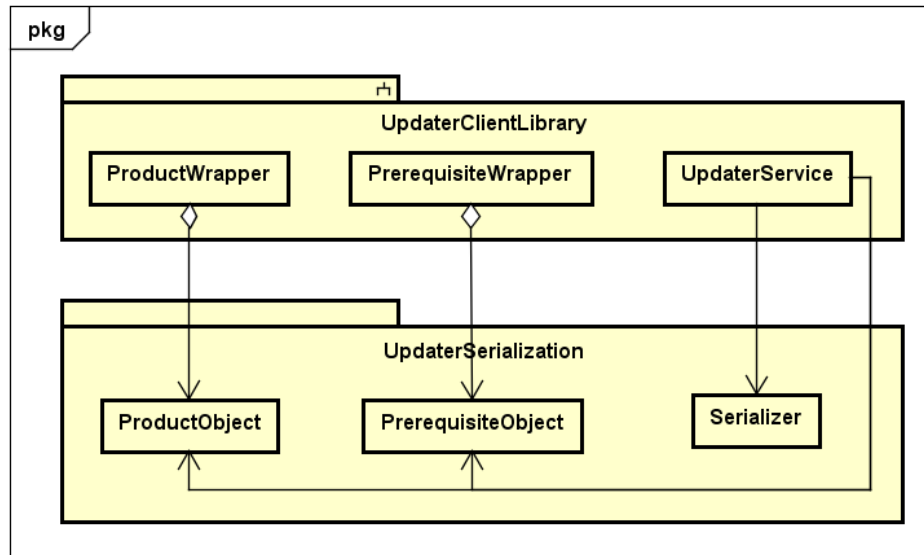


Figura 4.10: Relazioni componenti di Updater Client Library con le componenti di Updater Serialization

4.4.4 Relazioni Updater Server

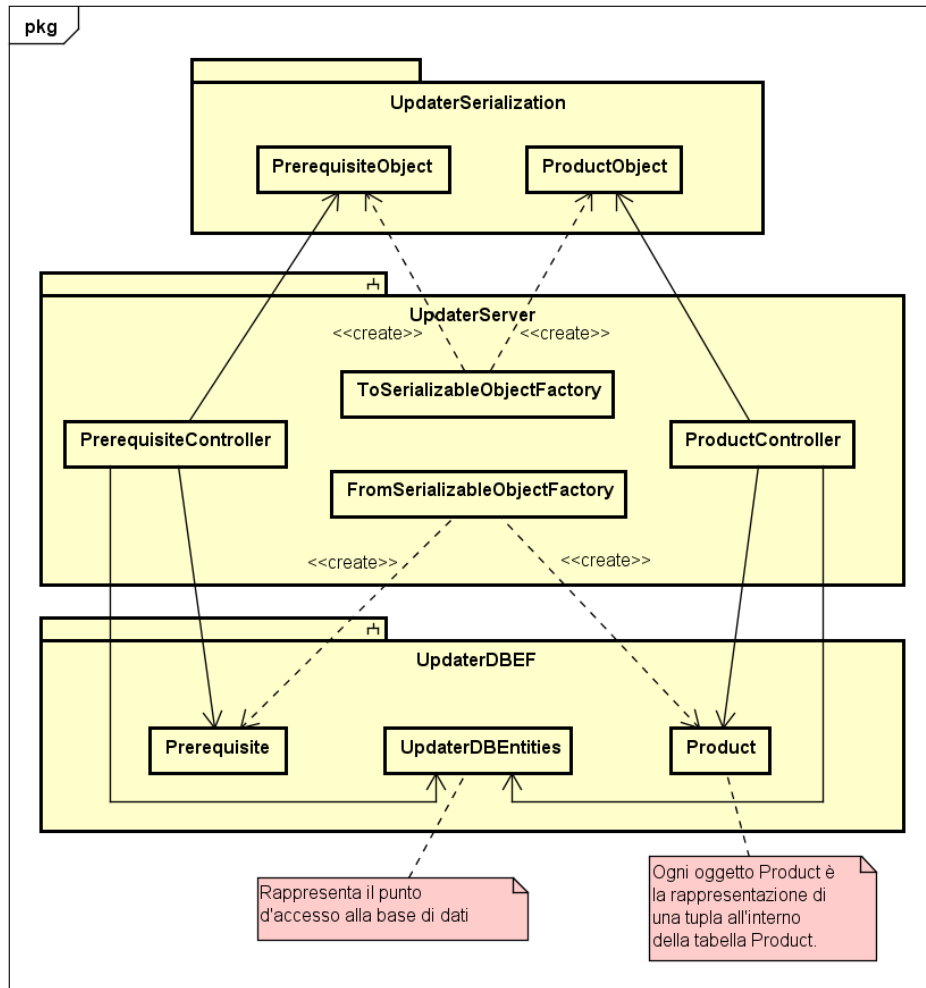


Figura 4.11: Relazioni componenti di Updater Server con le componenti di Updater Serialization e UpdaterEBEF

Capitolo 5

Dettagli componenti

Nel seguente capitolo si presentano i dettagli implementativi ritenuti più interessanti e utili delle macro componenti presentate in precedenza. La lettura di questa parte può essere utilizzata come approfondimento della parte architetturale o come approfondimenti indipendenti sullo sviluppo di parti del sistema Updater. Di seguito vengono presentate le sezioni accompagnate da una breve descrizione.

- Updater - Parte client;
 - **Interfaccia grafica 5.1.1:** descrive attraverso le immagini le componenti grafiche che compongono l'interfaccia grafica del programma Updater;
 - **Flusso principali operazioni 5.1.2:** descrive attraverso l'uso di diagrammi di attività gli algoritmi che effettuano le operazioni principali;
- Updater - parte server;
 - **Sito web amministrativo 5.2.1:** descrive come è costruito il sito web e come sono usate le tecnologie;
 - **Web Api documentazione 5.2.2:** documenta l'uso degli uri per effettuare richieste al servizio REST;
- **Concorrenza 5.3:** descrive come è gestita la concorrenza all'interno del programma Updater e il pattern utilizzato;
- **PowerShell script 5.4:** descrive lo script che permette il collegamento tra il sistema di *continuous integration* e il sistema Updater sviluppato.

5.1 Updater - Parte client

5.1.1 Interfaccia grafica

L'interfaccia grafica sviluppata per il programma Updater attualmente è in uno stato **prototipale**, ogni immagine presentata è usata come riferimento per indicare come l'utente può usufruire delle funzionalità del programma.

5.1.1.1 Componenti

Il programma Updater è costituito dalla finestra principale creata dall'oggetto:

```
MainWindow;
```

La quale è costituita da tre elementi grafici i quali ne racchiudono altri resi disponibili dal framework WPF e per questo non considerati. Essi sono:

1. `ProductsListView`;
2. `ProductControlView`;
3. `ProductInfoView`.

Nella figura 5.1 sono evidenziati e indicati attraverso la corrispondenza del numero dell'elenco precedente.

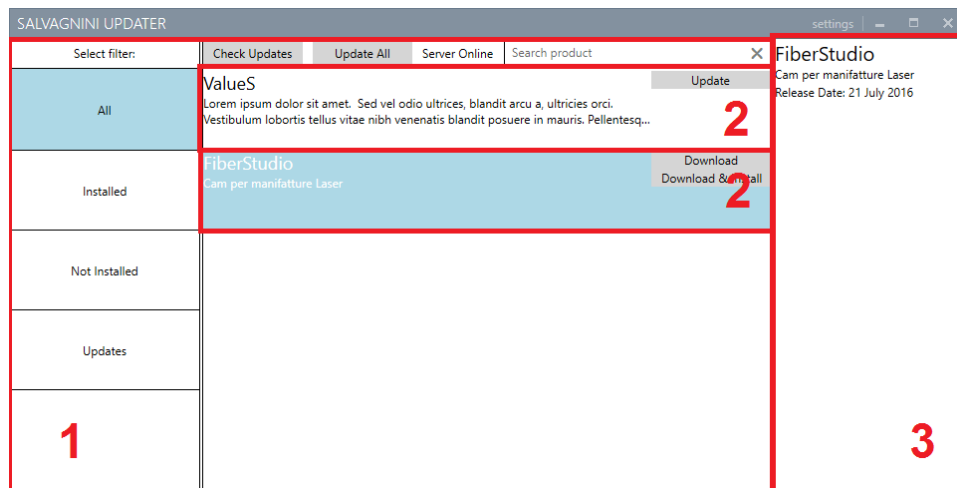


Figura 5.1: Componenti grafiche della finestra principale

È inoltre presente una schermata delle impostazioni del programma che si sovrappone agli elementi grafici 5.3, essa è definita nel componente `SettingsView`.

5.1.1.2 Funzionalità

La schermata principale (figura 5.2) permette subito di avere la lista dei prodotti software interagendo su di essi attraverso alla lista di pulsanti relativa. Questa cambia rendendo disponibili diversi pulsanti in base allo stato del prodotto software. Alla sinistra è possibile filtrare la lista di prodotti software mentre in alto al centro è possibile filtrarla attraverso la ricerca del nome all'interno della casella di ricerca. Sempre in alto è possibile trovare i pulsanti:

Check Updates che aggiorna la lista dei prodotti software attraverso una nuova richiesta al server;

Update All che aggiorna tutti i prodotti software che hanno aggiornamenti disponibili pendenti.

Alla sinistra invece troviamo uno spazio disponibile a mostrare tutti i dati disponibili di un prodotto software.

Tramite il click della parola *Settings* situata in alto a sinistra è possibile aprire una schermata che si sovrappone agli elementi grafici (figura 5.3). Da qui è possibile impostare l'esecuzione automatica di alcune funzionalità.

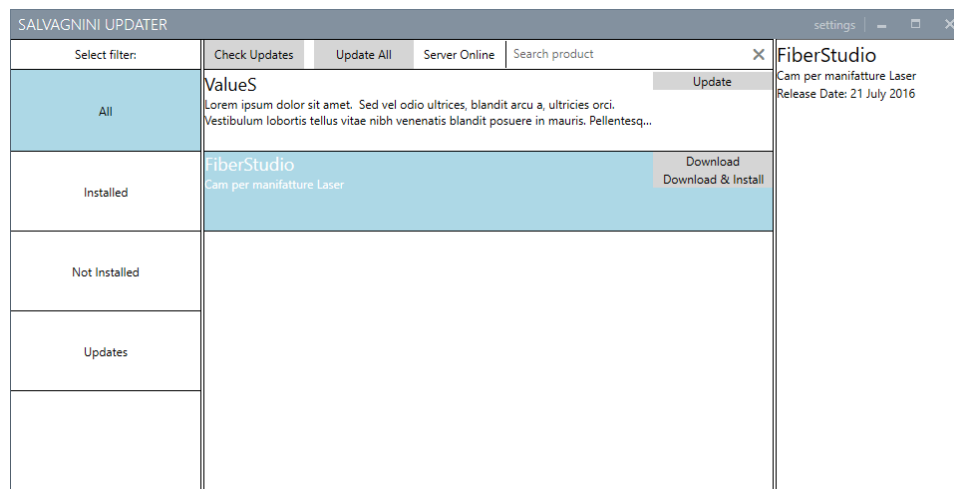


Figura 5.2: Updater - Finestra principale

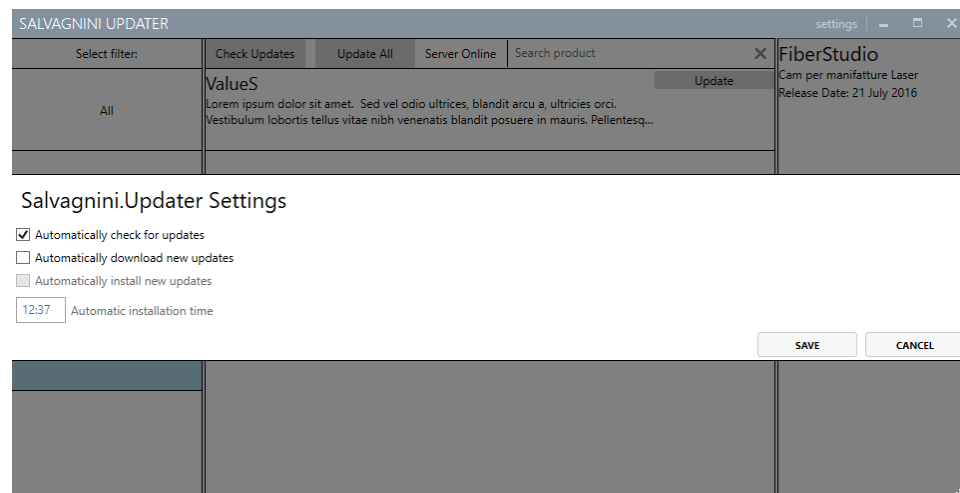


Figura 5.3: Updater - Schermata delle impostazioni

5.1.2 Flusso d'esecuzione operazioni principali

Di seguito vengono presentati i diagrammi di attività che mostrano in dettaglio i passaggi che la logica del programma effettua alla richiesta di esecuzione delle seguenti operazioni:

- Download (figura 5.4);
- Installazione (figura 5.5);
- Aggiornamento (figura 5.6);
- Reinstallazione;
- Download e installazione;

In particolare si vuole mostrare nel modo più chiaro possibile come si sia risolto il problema della gestione dei prerequisiti prima dell'installazione di un prodotto software che li richiede. I diagrammi manterranno nella colonna centrale il flusso con i casi peggiori. Per i diagrammi *“Reinstallazione”* e *“Download e installazione”* si veda il diagramma *“Aggiornamento”* (figura 5.6).

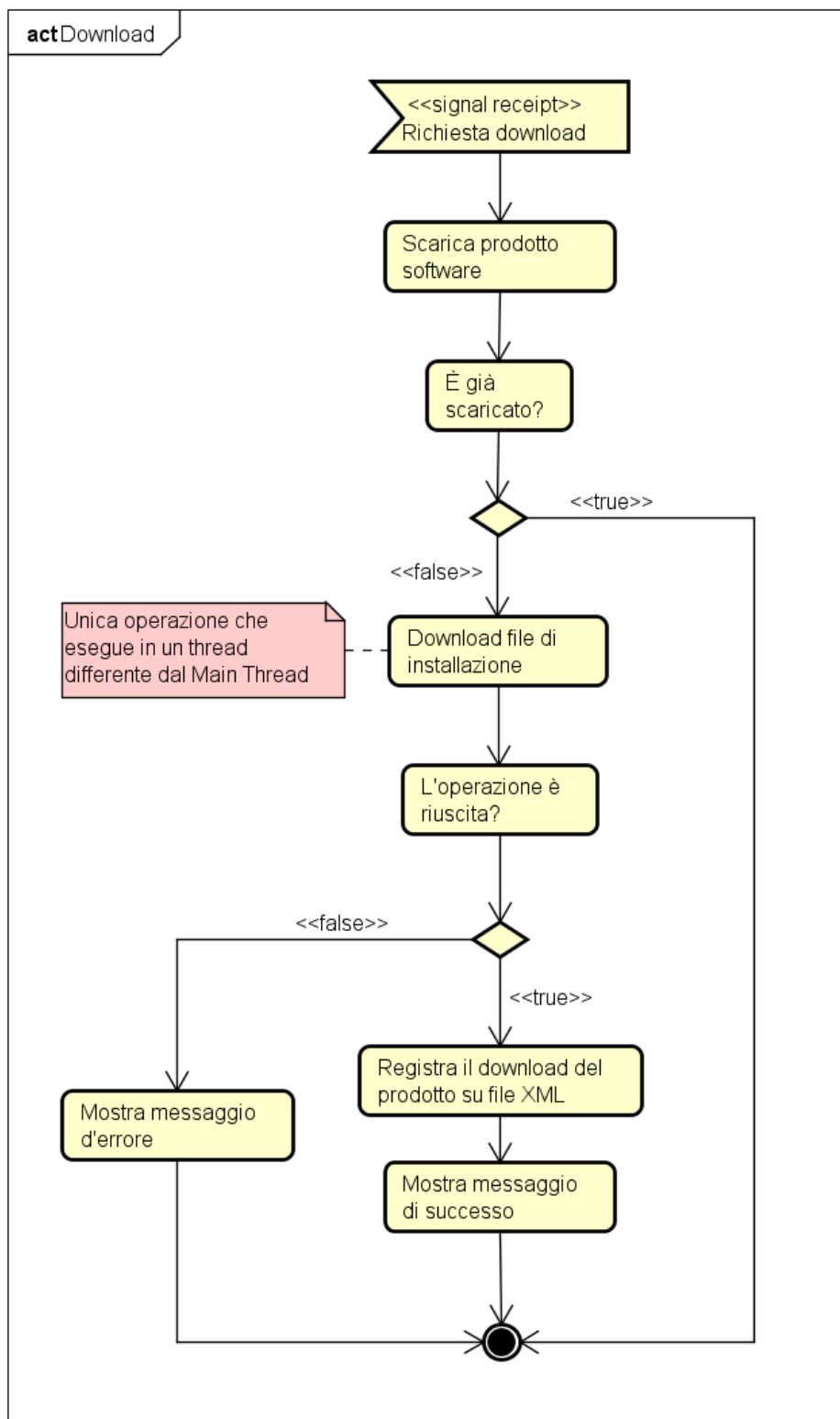


Figura 5.4: Diagramma di attività - Download

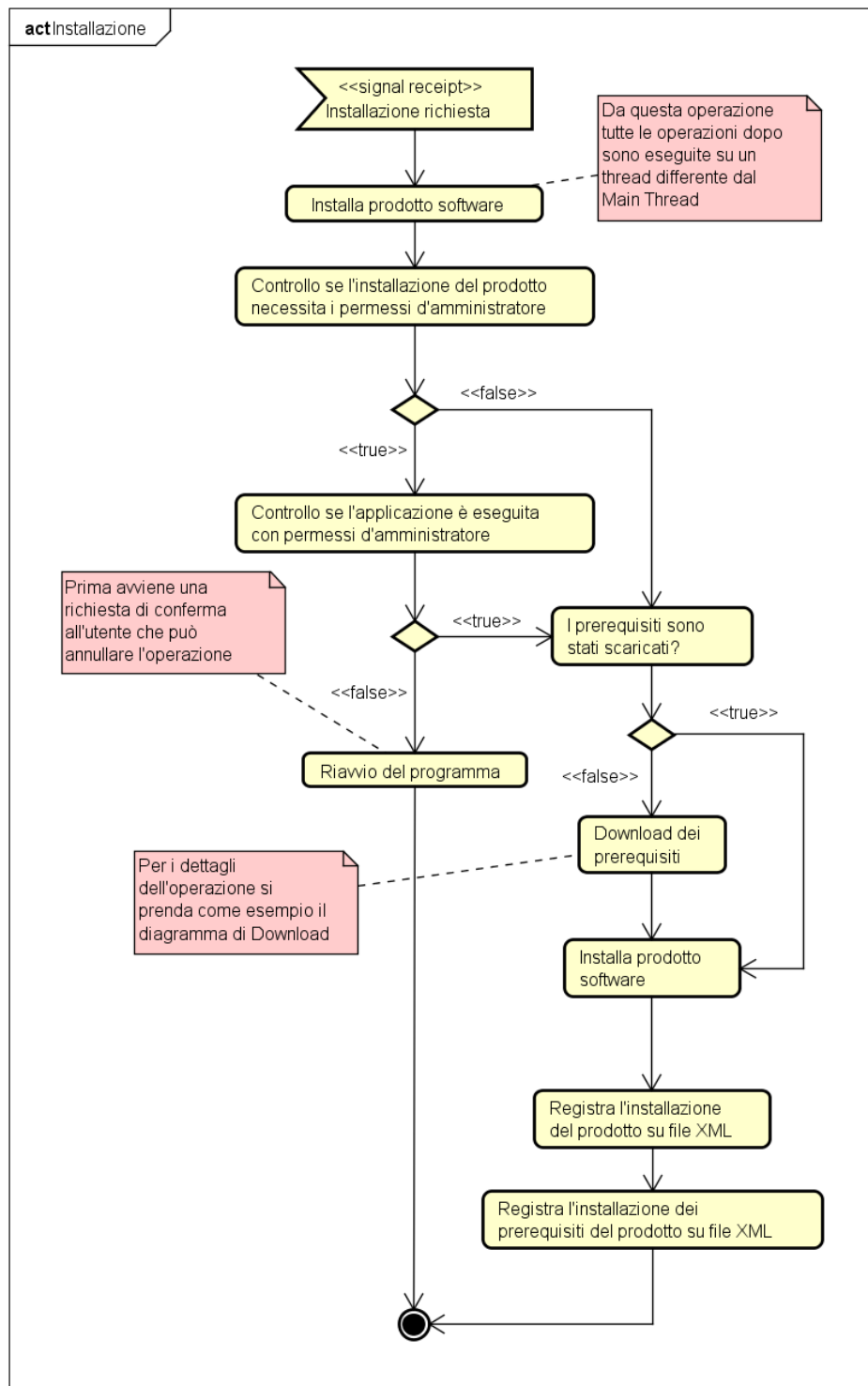


Figura 5.5: Diagramma di attività - Installazione

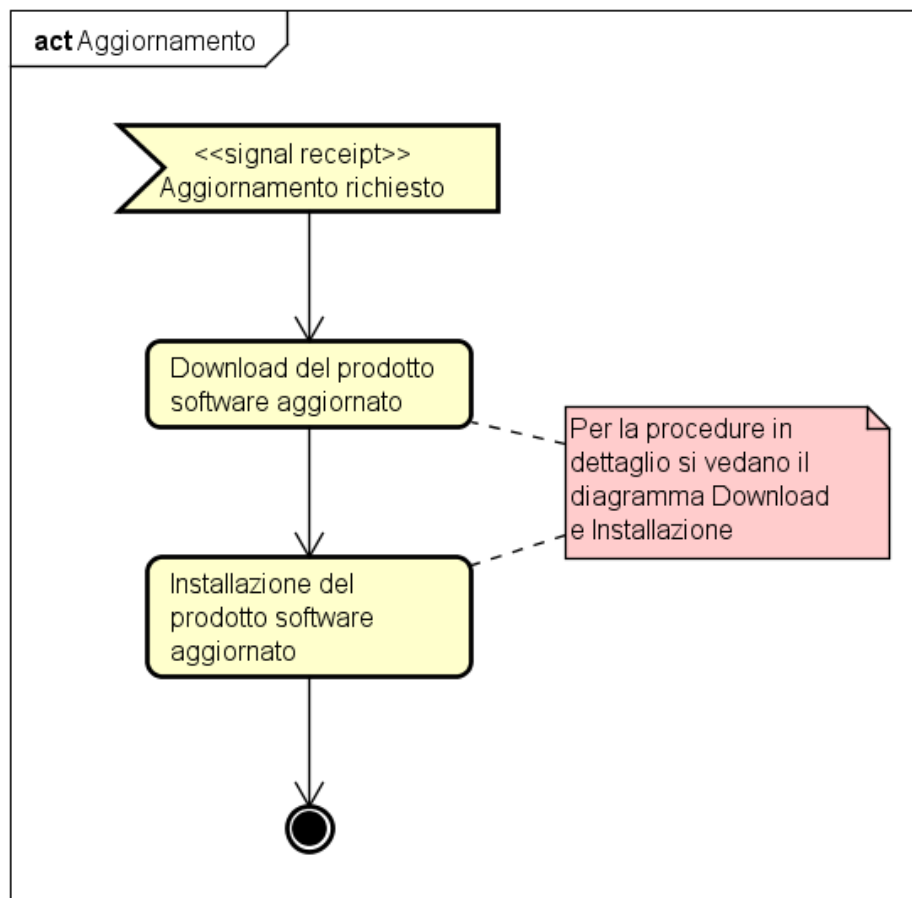


Figura 5.6: Diagramma di attività - Aggiornamento

5.2 Updater - Parte server

5.2.1 Sito web amministrativo

Per semplificare e agevolare le modifiche alla base di dati si è costruito un piccolo sito web costituito da quattro pagine web composte con elementi di Bootstrap e rese dinamiche grazie all'uso di Javascript con la libreria KnockoutJs.

5.2.1.1 Pagine web

Login.html permette l'autenticazione per accedere alle altre pagine;

Products.html permette di aggiungere, eliminare o modificare un prodotto software (figura 5.7).





Prerequisites.html permette di aggiungere, eliminare o modificare un prerequisito software (figura 5.8).

Dependencies.html permette di associare e dissociare i prodotti software dai prerequisiti software (figura 5.9).

Salvagnini Updater Admin

LogoutProductsProduct dependenciesPrerequisites

Products

Id	Name	Description	Arc	Min OS	Path	
83	ValueS	Lorem ipsum dolor sit amet. Sed vel odio ultrices, blandit arcu a, ultricies orci. Vestibulum lobortis tellus vitae nibh venenatis blandit posuere in mauris. Pellentesque auctor convallis tellus in eleifend. Nulla suscipit tellus nec massa consequat scelerisque.	64	6.1	C:\Salvagnini\syscon\bin\ValueS\ValueS.exe	 
87	FiberStudio	Cam per manifatture Laser	64	6.1		 

Add Product

Name

Description

Architecture

x64

Minimum OS version

Windows XP

Executable path

C:\Salvagnini\syscon\bin\Example\Example.exe

Upload product setup

C:\your\path\setup.exe

Browse

Add

Reset

Figura 5.7: Pagina web - Products.html

Salvagnini Updater Admin

Logout

Products

Product dependencies

Prerequisites

Prerequisites

Id	Name	
27	dotnetfx20sp1.exe	<div><div></div><div></div></div>
28	dotnetfx45.exe	<div><div></div><div></div></div>
29	vcredist2005_x64.exe	<div><div></div><div></div></div>
30	vcredist2005_x86.exe	<div><div></div><div></div></div>
31	vcredist2013_x64.exe	<div><div></div><div></div></div>
32	vcredist2013_x86.exe	<div><div></div><div></div></div>
33	vcredist_x86.exe	<div><div></div><div></div></div>
34	vcredist_x64.exe	<div><div></div><div></div></div>

Add Prerequisite

Upload prerequisite setup

C:\your\path\setup.exe

Browse

Add

Reset

Figura 5.8: Pagina web - Prerequisites.html

Salvagnini Updater Admin

LogoutProductsProduct dependenciesPrerequisites

Select product

Id	Name	Description	Arc	Min OS	Path	
83	ValueS	Lorem ipsum dolor sit amet. Sed vel odio ultrices, blandit arcu a, ultricies orci. Vestibulum lobortis tellus vitae nibh venenatis blandit posuere in mauris. Pellentesque auctor convallis tellus in eleifend. Nulla suscipit tellus nec massa consequat scelerisque.	64	6.1	C:\Salvagnini\syscon\bin\ValueS\ValueS.exe	Select
87	FiberStudio	Cam per manifatture Laser	64	6.1		Select

Select dependencies

Product: ValueS - Id product: 83

☒ Id: 27 - dotnetfx20sp1.exe

☒ Id: 28 - dotnetfx45.exe

☒ Id: 29 - vcredist2005_x64.exe

☒ Id: 30 - vcredist2005_x86.exe

☒ Id: 31 - vcredist2013_x64.exe

☒ Id: 32 - vcredist2013_x86.exe

☐ Id: 33 - vcredist_x86.exe

☐ Id: 34 - vcredist_x64.exe

ApplicaReset

Figura 5.9: Pagina web - Dependencies.html

5.2.1.2 Script Javascript

Per rendere le pagine web dinamiche si è utilizzato l'attributo **data-binding** di HTML5 e la libreria KnockoutJs che lo gestisce. Per ogni pagina si è codificato il relativo file di script più un file **Common.js** che contiene funzioni utilizzate in tutte le pagine.

La struttura degli script è la seguente:

1. Si assegna all'elemento HTML l'attributo **data-bind** al cui interno vengono usate keyword che Knockout riconosce e interpreta. Ad esempio:

```
<p data-bind="text : message"></p>
```

2. A questo punto sul file script associato alla pagina si crea un oggetto che rappresenta il ViewModel di quella pagina web:

```
var ViewModel = function () {  
    // ...  
}  
  
ko.applyBindings(new ViewModel);
```

3. All'interno del ViewModel saranno codificati i campi dati e i metodi che agiranno agli input dell'utente nella pagina html. Per rendere l'aggiornamento dei dati (*data-binding*) automatico è sufficiente definire i campi dati del ViewModel con un costrutto offerto dalla libreria (il nome delle variabili deve corrispondere alla stringa inserita nel data-bind delle pagine web):

```
var message = ko.observable();
```

4. Ora qualsiasi cambiamento sul valore della variabile **message** sarà automaticamente riflesso nella pagina web.

5.2.2 Web Api - Documentazione

Di seguito si elencano i **controller** definiti nella costruzione del servizio REST Updater Server. Per ogni controller vengono elencati gli indirizzi uri per poter effettuare una richiesta HTTP. Le tipologie di richieste HTTP supportate sono:

- Get: alla richiesta viene risposto restituendo dati;
- Post: alla richiesta vengono processati i dati contenuti in essa, se conformi vengono salvati nella base di dati;
- Delete: alla richiesta viene effettuata una cancellazione definitiva dei dati indicati;
- Put: alla richiesta vengono processati i dati contenuti in essa, se conformi vengono salvati nella base di dati sovrascrivendo quelli indicati.

La trasmissione dei dati avviene attraverso la serializzazione in formato JSON.

5.2.2.1 ProductsController

Operazioni GET:

- **api/Products**
Restituisce tutti i prodotti software all'interno della base di dati;
- **api/Products/{id}**
Restituisce il prodotto software identificato dall'id indicato;
- **api/Products/{productId}/Prerequisites**
Restituisce la lista di prerequisiti software del prodotto identificato da productId;
- **api/Products/check/{id}/{fileName}**
Restituisce vero se il prodotto software identificato dall'id indicato ha il fileName indicato altrimenti falso;

Operazioni PUT:

- **api/Products/{id}**
Sovrascrive con nuovi dati le informazioni associate al prodotto software identificato dall'id indicato;

Operazioni POST:

- **api/Products**
Aggiunge un prodotto software con i dati contenuti nella richiesta;

- `api/Products/{productId}/Prerequisites`

Associa i prerequisiti contenuti nella richiesta al prodotto software identificato dal `productId` indicato. Dopo l'operazione solo i prerequisiti segnalati saranno associati al prodotto;

Operazioni DELETE:

- `api/Products/{id}`

Elimina il prodotto software identificato dall'`id` indicato.

5.2.2.2 PrerequisiteController

Operazioni GET:

- `api/Prerequisites`

Restituisce tutti i prerequisiti software all'interno della base di dati;

- `api/Prerequisites/check/{id}/{fileName}`

Restituisce vero se il prodotto software identificato dall'`id` indicato ha il `fileName` indicato altrimenti falso;

- `api/Prerequisites/check/{id}/hasDependencies`

Restituisce vero se il prerequisito software identificato dall'`id` indicato è associato almeno ad un prodotto software;

Operazioni PUT:

- `api/Prerequisites/{id}`

Sovrascrive con nuovi dati le informazioni associate al prerequisito software identificato dall'`id` indicato;

Operazioni POST:

- `api/Prerequisites`

Aggiunge un prerequisito software con i dati contenuti nella richiesta;

Operazioni DELETE:

- `api/Prerequisites/{id}`

Elimina il prerequisito software identificato dall'`id` indicato.

5.3 Concorrenza

La concorrenza ha coinvolto solamente la parte client del sistema sviluppato. La necessità di ricorrere alla concorrenza giunge quando si vuole rendere la propria applicazione responsive, ossia non impegnare il Main Thread, definito anche come UI Thread, in operazioni lunghe mostrando di fatto all'utente l'interfaccia grafica bloccata. Mostrare all'utente l'interfaccia bloccata non è mai buona pratica, l'utente infatti traduce questo feedback in qualcosa del tipo: *"l'applicazione si è bloccata"* e non *"L'applicazione sta elaborando qualcosa per te"*. Per questo motivo nella parte client del sistema si è deciso di inserire multipli thread, e incapsulare le elaborazioni più complesse in task.

5.3.1 Pattern

In C# esistono vari pattern per l'esecuzione di un ambiente concorrente, l'ultimo di questi, nonché quello consigliato dalla stessa Microsoft¹ e scelto per il progetto, è il *Task-based Asynchronous Pattern* (TAP).

5.3.1.1 Applicazione Task-based Asynchronous Pattern

Esso si basa sui tipi `Task` e `Task<TResult>` definiti all'interno del namespace `System.Threading.Tasks` che sono utilizzati per rappresentare operazioni asincrone. Il pattern permette una notevole semplificazione nella codifica per introdurre codice concorrente, inoltre attraverso l'uso delle keyword `async` e `await`, parte del linguaggio C# è molto più facile capire quali porzioni di codice sono eseguite concorrentemente e quali no. La definizione di un metodo asincrono sarà di questo tipo:

```
public async Task<TResult> AsyncMethod(Type param)
```

e potrà essere chiamato attraverso l'uso di `await`:

```
var returnValue = await AsyncMethod(param);
```

la concorrenza viene creata attraverso l'uso del metodo statico `Task.Run()`:

```
var returnValue = await Task.Run(() => AsyncMethod());
```

Da qui in poi tutti i metodi definiti `async` a partire da `AsyncMethod()` saranno eseguiti in un thread differente dal Main Thread.

5.3.2 Componenti coinvolte

Di seguito si elencano le componenti e i relativi metodi che includono la concorrenza:

¹Stephen Toub, *Asynchronous Programming Patterns* [https://msdn.microsoft.com/it-it/library/jj152938\(v=vs.110\).aspx](https://msdn.microsoft.com/it-it/library/jj152938(v=vs.110).aspx)

```

ProductWrapper: InitializePrerequisitesWrapper();

ProductWrapper: Download() (attraverso l'uso della classe WebClient
di .NET);

ProductsPanelModel: LoadProduct();

ProductControlModel: InstallProduct().

```

In generale, nell'implementazione si sono seguite queste linee guida:

- Qualsiasi operazione che richiede l'accesso alla rete è asincrona;
- Le operazioni che non richiedono l'accesso a internet ma comunque necessitano di tempo per i calcoli sono asincrone e sono contenute esclusivamente nel macro componente *Updater UI Controls*;

5.3.3 Criticità e miglioramenti

Fortunatamente le operazioni da eseguire in concorrenza non riportano i problemi di accessi condivisi a risorse e questo ha semplificato l'implementazione. Ciononostante la *best practice* di mantenere separato il codice concorrente dal codice non concorrente non è seguita. Seppur le componenti che includono l'avvio di nuovi task sono poche e i casi in cui si verifica sono limitati a quelli elencati precedentemente la linea di codice:

```
return Value = await Task.Run(() => object.Method()));
```

non risulta mai ben isolata dal resto del codice sincrono.

Una più corretta soluzione sarebbe quella di incorporare l'avvio di nuovi task all'interno di singoli oggetti ben definiti in cui è chiaramente visibile l'implementazione della concorrenza.

5.3.4 Approfondimenti

- Stephen Cleary, *Async & Await* <http://blog.stephencleary.com/2012/02/async-and-await.html>;
- Stephen Cleary, *Asynchronous Programming Best Practices* <https://msdn.microsoft.com/magazine/jj991977.aspx>;
- Stephen Toub, *Task-based Asynchronous Pattern*, disponibile al seguente indirizzo <https://www.microsoft.com/en-us/download/details.aspx?id=19957>;
- Stephen Cleary, *Concurrency in C# Cookbook*.

5.4 PowerShell script

Nella presente sezione viene presentato e descritto lo script powershell che si occupa di automatizzare l'aggiornamento dei prodotti software all'interno della base di dati.

5.4.1 Ambito applicativo dello script

Il sistema di *Continuous Integration* (CI) aziendale consente di definire dei processi per la compilazione del codice, l'esecuzione dei test e l'esecuzione di script batch. L'aggiornamento di un prodotto software attraversa i seguenti passaggi:

1. **Commit** dello sviluppatore nel repository;
2. **Esecuzione processo** definito nella piattaforma di **CI** (solitamente compilazione ed esecuzione dei test);
3. **Creazione dell'eseguibile binario** di installazione del prodotto se il passaggio precedente non ha riscontrato errori;

A questo punto entra in gioco la nuova parte sviluppata:

4. **Esecuzione** di uno **script batch**, il quale contiene gli input per lo script powershell;
5. **Esecuzione dello script powershell**. A questo punto lo script esegue una normale richiesta PUT al servizio REST e aggiorna il prodotto specificato dagli input in ingresso.

5.4.2 Implementazione script powershell

Input richiesti

- **\$InFile**: il percorso dell'eseguibile per l'installazione;
- **\$uri**: l'indirizzo per effettuare la richiesta PUT al server;
- **\$user**: il nome utente per l'autenticazione della richiesta;
- **\$pass**: la password per l'autenticazione della richiesta.

Dipendenze esterne con .NET

- **System.Web**
- **System.Net.Http**

Logica operativa

La logica operativa segue la costruzione di una richiesta HTTP di tipo PUT con autorizzazione base:

1. Definizione dell'autorizzazione base attraverso la coppia username e password per l'invio della richiesta HTTP;
2. Costruzione `ContentDispositionHeaderValue` di tipo *form-data* e inserimento del file e del suo nome;
3. Costruzione dell'header della richiesta HTTP inserendo all'interno l'oggetto precedentemente creato: `ContentDispositionHeaderValue`;
4. Invio della richiesta HTTP.

Capitolo 6

Conclusioni

Tutti i requisiti sono stati soddisfatti. Il sistema sviluppato offre tutte le funzionalità base per poter garantire la distribuzione dei prodotti software ai clienti e quindi soddisfa l'obiettivo principale. Nonostante il risultato positivo raggiunto il sistema è da ritenersi ancora come un **sistema software immaturo** per un rilascio ufficiale esterno all'azienda. Infatti sono quantomeno necessari ulteriori refinimenti.

6.1 Futuri sviluppi

Il progetto sviluppato nonostante soddisfi tutti i requisiti individuati e richiesti è ancora lontano dalla consegna all'utente finale. I punti su cui ancora bisogna lavorare sono:

- **Configurazione di una macchina server** all'interno dell'azienda. La quale deve saper sopportare un carico di trasmissione ancora oggi sconosciuto e non considerato poiché si è sempre lavorato in un ambiente di sviluppo virtuale.
- **Rafforzare il livello di sicurezza** del servizio REST. Ad oggi i dati trasmessi non sono sensibili ma ulteriori estensioni, come per esempio la distribuzione di licenze, richiederanno un livello di sicurezza molto più alto e pensato.
- **Migliorare** notevolmente **il design grafico** del programma Updater. Di fatto quello mostrato nel presente documento è solo un'interfaccia prototipale. Visto che l'obiettivo dell'intero progetto è quello di semplificare al cliente la fruizione dei prodotti software aziendali è estremamente importante che il primo prodotto installato dall'utente abbia un impatto positivo sia per l'aspetto grafico sia per l'usabilità.
- Migliorare il design e la struttura del codice già sviluppato soprattutto nei punti critici indicati nel capitolo 4. Eseguire l'opportuno refac-

toring di queste parti del sistema garantirà che le estensioni future possano essere aggiunte senza l'aggravamento dei costi e della leggibilità dell'intero sistema.

6.2 Future estensioni

Durante lo sviluppo nuove possibili funzionalità sono state individuate. Esse sono:

- Implementazione di un'autenticazione nel programma Updater gestendo gli account di ogni cliente (già in parte implementato). Questo consentirebbe una diversa distribuzione dei prodotti software tra i clienti aziendali.
- Gestione centralizzata delle licenze dei prodotti software in uso dall'utente.

Bibliografia

- [1] J. Albahari and B. Albahari. *C# 6.0 in a Nutshell: The Definitive Reference*. O'Reilly Media, 2015.
- [2] K. Beck. *Implementation Patterns*. The Addison-Wesley signature series. Addison-Wesley, 2008.
- [3] S. Cleary. *Concurrency in C# Cookbook*. O'Reilly Media, 2014.
- [4] D. Crockford. *JavaScript: The Good Parts*. O'Reilly Media, 2008.
- [5] P.M. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Pearson Education, 2007.
- [6] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.
- [8] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2010.
- [9] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin series. Prentice Hall, 2009.
- [10] Stephen Toub. The task-based asynchronous pattern, 2012.

Glossario

Baloon Piccola finestra pop-up che informa l'utente dell'avvenimento di un problema non critico.

Code Behind Termine utilizzato per descrivere il codice che viene unito agli oggetti definiti dal markup quando viene compilata una pagina XAML.

Continuos Delivery (CD) Continuous Delivery è una disciplina dello sviluppo software in cui si costruisce software garantendo che questo possa essere rilasciato in produzione in ogni momento.

Continuos Integration (CI) La Continuos Integration (Integrazione continua) è una pratica dello sviluppo software in cui i membri di un team integrano il loro lavoro frequentemente con quello fatto dagli altri, spesso ogni individuo integra almeno una volta al giorno. Ogni integrazione è verificata da un sistema di build automatico che include test il quale avvisa tempestivamente se ci sono errori di integrazione. Tutto ciò porta a sviluppare un software coeso più rapidamente[5].

Data binding Tecnica che lega dei dati di un elemento fornitore (per es. una struttura dati) ad un elemento consumatore (per es. un componente grafico che rappresenta una lista) garantendo che entrambi siano sempre sincronizzati.

Design Pattern Soluzione progettuale generale ad un problema ricorrente.

Framework Architettura logica di supporto su cui un software può essere progettato e realizzato facilitandone lo sviluppo da parte del programmatore.

HyperText Transfer Protocol (HTTP) L'Hypertext Transfer Protocol (HTTP) è un protocollo a livello applicativo per sistemi informativi distribuiti.

Macro componente Insieme di componenti (classi), può coincidere con un namespace (C#) o package (Java).

Metro Design Nuovo linguaggio di design sviluppato da Microsoft per i propri prodotti per la creazione di interfacce grafiche eleganti, veloci e moderne.

Prerequisiti Prodotti software sviluppati da terzi su cui si basa lo sviluppo e il funzionamento di altri prodotti software (per es. l'ambiente .NET o JDK).

Prodotto software Software sviluppati dall'azienda e coinvolti nella distribuzione del sistema Updater.

Refactoring Il refactoring è il processo di modifica di un sistema software che ha l'obiettivo di migliorare la struttura interna di esso mantenendo invariato il suo comportamento esterno.

Repository ambiente di un sistema informativo in cui vengono gestiti metadati attraverso tabelle relazionali.

REpresentational State Transfer (REST) Tipo di architettura software per i sistemi distribuiti nel web.

Solution File di configurazione dell'ambiente di sviluppo Visual studio che rappresenta un insieme di progetti.

TaskBar Speciale barra degli strumenti dei sistemi operativi Windows in cui è possibile interagire con programmi del sistema o programmi eseguiti in background.