Analisi e Valutazione di Integrazione tra Linguaggi di Programmazione Differenti in un Contesto Aziendale

FRANCESCO PICCHIETTI

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Laurea in Informatica

Relatori :

Giacomo Cabri

Riccardo Martoglia

Introduzione

La tesi è composta da 5 capitoli divisi in due parti:

Parte I – Il caso di studio

- Presentazione del contesto e dei linguaggi coinvolti
- Integrazione

Parte II – Software integrato in contesto aziendale

- Software sviluppato
- Misurazioni e risultati ottenuti
- Conclusioni

Introduzione

All’interno della prima parte della tesi si parla della teoria che sta dietro al software sviluppato e ai processi di integrazione.

Viene affrontato l’argomento dei due linguaggi di programmazione: Python e C#, ponendo particolare attenzione alle differenze di entrambi. Successivamente viene spiegato quali siano i due meccanismi per poter integrare i due linguaggi e, anche in questo caso, vengono evidenziati i loro pregi e difetti. All’interno di questa parte è stato inoltre mostrato l’ambito aziendale nel quale sono stati sviluppati i software.

Nella seconda parte si è passati al lato pratico. Vengono presentati i software sviluppati e i processi di integrazione che sono stati adottati, segue un analisi relativa alla velocità di esecuzione dei programmi. Alla fine vengono fatte le conclusioni relative all’integrazione.

Ambito applicativo

Durante questo studio verrà affrontato, in maniera teorica e pratica, l'integrazione di software scritti in linguaggi differenti.

I linguaggi in questione sono Python e C#. Per testare a livello pratico questa integrazione sono stati sviluppati due software: il primo per l'estrapolazione di informazioni hardware relative ad una macchina, il secondo per effettuare delle operazioni di pulizia di un database.

L'integrazione avviene nel momento in cui i software sviluppati devono essere inseriti all'interno di un programma creato per poter automatizzare alcuni processi di assistenza e setup su determinate macchine.

Verranno poi valutati i metodi di integrazione utilizzati tramite delle misurazioni cronometriche per calcolarne le prestazioni

- Cenni teorici sull’integrazione
- Software sviluppati
- Conclusioni

1.1 Il contesto aziendale

Lo studio e l'utilizzo di sistemi di integrazione sono stati fatti all'interno di un contesto aziendale durante il periodo di tirocinio.

L'azienda ospitante si occupa della progettazione e della creazione di macchinari per la stampa di prodotti ceramici dette stampanti. Le occupazioni primarie dell'ufficio informatico sono:

- Lo sviluppo del software per l'interazione con componenti della macchina
- Assistenza sulle stampanti acquistate dalle aziende

Due principali processi di assistenza si prestavano bene ad un processo di automazione:

- Software di estrazione delle informazioni riguardo alla macchina
- Software per l'operazione di *shrinking* di un database

1.2 Python

Python può essere considerato come un linguaggio di programmazione di più "alto livello" rispetto alla maggior parte degli altri linguaggi, esso è orientato a oggetti, il che gli consente, tra gli altri usi, a sviluppare applicazioni complesse, scripting, applicazioni web computazione numerica e system testing.

Python è un linguaggio *interpretato*, *multi paradigma* e *tipizzato dinamicamente*.

I suoi utilizzi sono principalmente nei seguenti ambiti:

- Applicazioni web
- Computazione scientifica
- Intelligenza artificiale
- Data mining

1.3 C#

C# è un linguaggio multi-paradigma ma che si presta principalmente a quella che è detta programmazione a oggetti. Esso è stato sviluppato da Microsoft e si appoggia al framework .NET. La sintassi di C# prende ispirazione da linguaggi come C++, Java e Delphi.

C# è un linguaggio principalmente orientato alla programmazione ad oggetti e che utilizza una sintassi più ricca rispetto a Python, con l'utilizzo di « **;** », « **{ }** ».

I suoi utilizzi sono principalmente nei seguenti ambiti :

- Sviluppo web
- Creazione applicazioni Windows
- Sviluppo di videogiochi

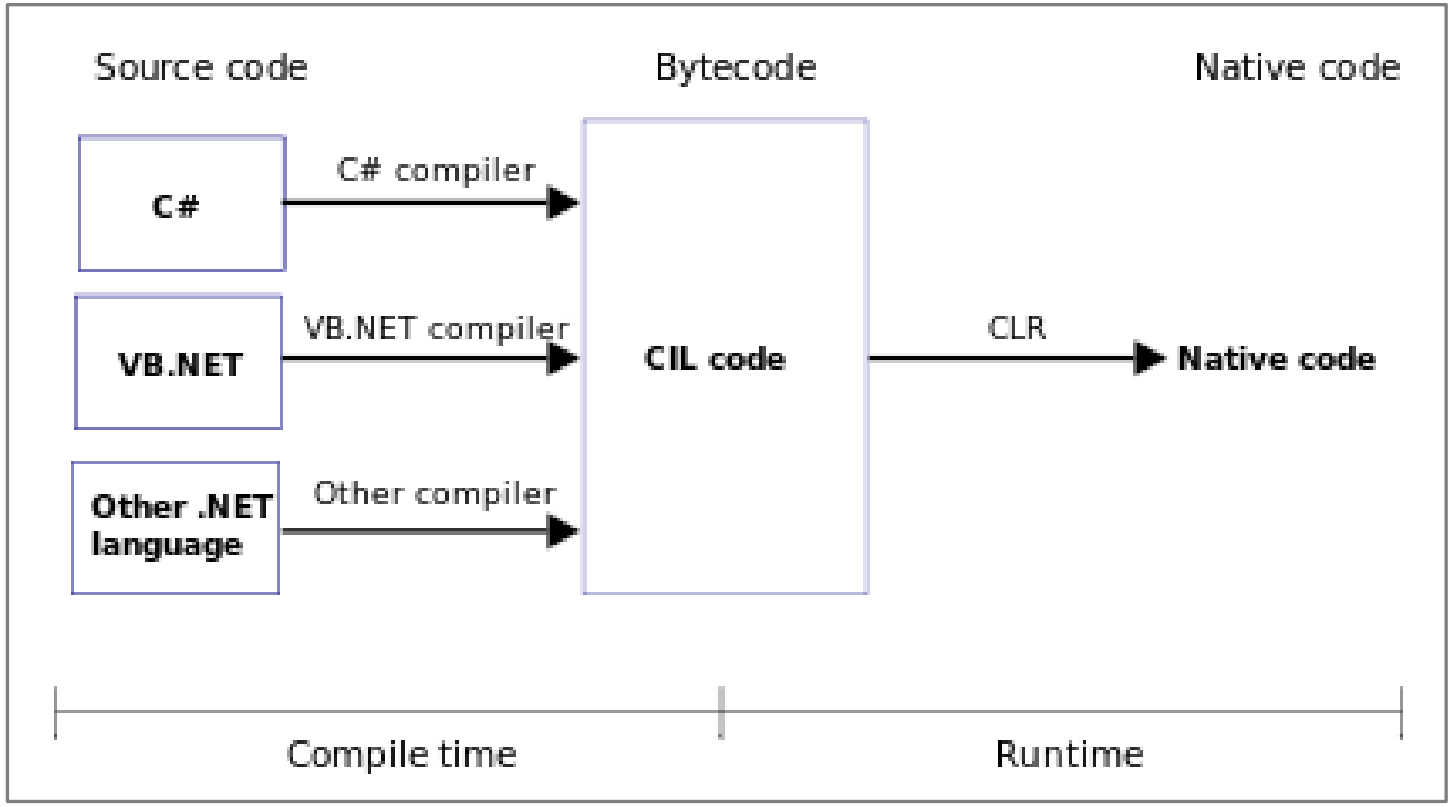
1.3 C#

C# è di difficile definizione per quanto riguarda il processo di compilazione. Esso subisce infatti un processo di compilazione tramite i criteri della *Just In Time (JIT).*

La procedura è composta da due passaggi :

- Prima dell'esecuzione, il codice viene convertito in codice intermedio detto CIL
- Durante l'esecuzione, il codice intermedio viene convertito dalla Common Language Runtime in un linguaggio macchina specifico per l'hardware ospite.

1.3 C#


Source code
Bytecode
Native code
C#
C# compiler
VB.NET
VB.NET compiler
Other .NET language
Other compiler
CIL code
CLR
Native code
Compile time
Runtime

1.4 Differenze tra i due linguaggi

Le differenze tra i due linguaggi possono essere riassunti nella seguente tabella :

	Python	C#
Facile apprendimento	✔	✗
Open source	✔	✗
Tipizzato staticamente	✗	✔
Parentesi graffe e semicolon	✗	✔
Indentazione	✔	✗
Compilazione	✗	✔
Gestione puntatori	✗	✔
Multi ereditarietà	✔	✗
Multi paradigma	✔	✗

2.1 Introduzione ad IronPython

IronPython è un software che si occupa di integrare codice Python all'interno del codice C# e viceversa.

Tramite questo programma è possibile utilizzare classi e metodi scritti in linguaggio Python direttamente all'interno del codice C# in modo molto semplice.

Il motore di IronPython si occupa di compilare il codice Python in codice IL, il quale viene eseguito dalla *Common Language Runtime* (*CLR)* come per il codice C#.

Nelle versioni aggiornate di IronPython viene sfruttato il meccanismo della *Dynamic Language Runtime (DLR).*

2.2 Compilazione di Python

Python, essendo un linguaggio interpretato, non porta alla creazione di un eseguibile come per i linguaggi C o C++, è quindi necessario utilizzare un software in grado di compilarlo.

PyInstaller è un software in grado di compilare codice scritto in linguaggio Python prendendo tutte le dipendenze del caso e riunendo il tutto in un unico pacchetto con un eseguibile. L'eseguibile creato può essere eseguito anche su macchine sprovviste dell'interprete Python.

PyInstaller non è un *cross-compiler,* infatti l'eseguibile che viene creato è compatibile solamente con il sistema operativo sul quale *PyInstaller* viene eseguito. Questo software segue questi passaggi:

- Risale ad ogni libreria necessaria per eseguire il software
- Copia i file con l'interprete Python e ne viene creato un eseguibile

2.3 Differenze

Per fare un confronto tra i due metodi ne vengono discussi i punti di forza e le debolezze.

I punti di forza di **IronPython** sono i seguenti:

- Alto livello di integrazione
- Integrazione semplice
- Utilizza Python 3.4

Svantaggi :

- Debug complesso
- Incompatibilità con alcune librerie Python

2.3 Differenze

I punti di forza della **compilazione** sono i seguenti :

- Utilizza sempre la versione di Python più aggiornata
- Compatibilità con le librerie
- Portabilità

Le debolezze sono :

- Mantenimento del codice più complesso
- Più basso livello di integrazione

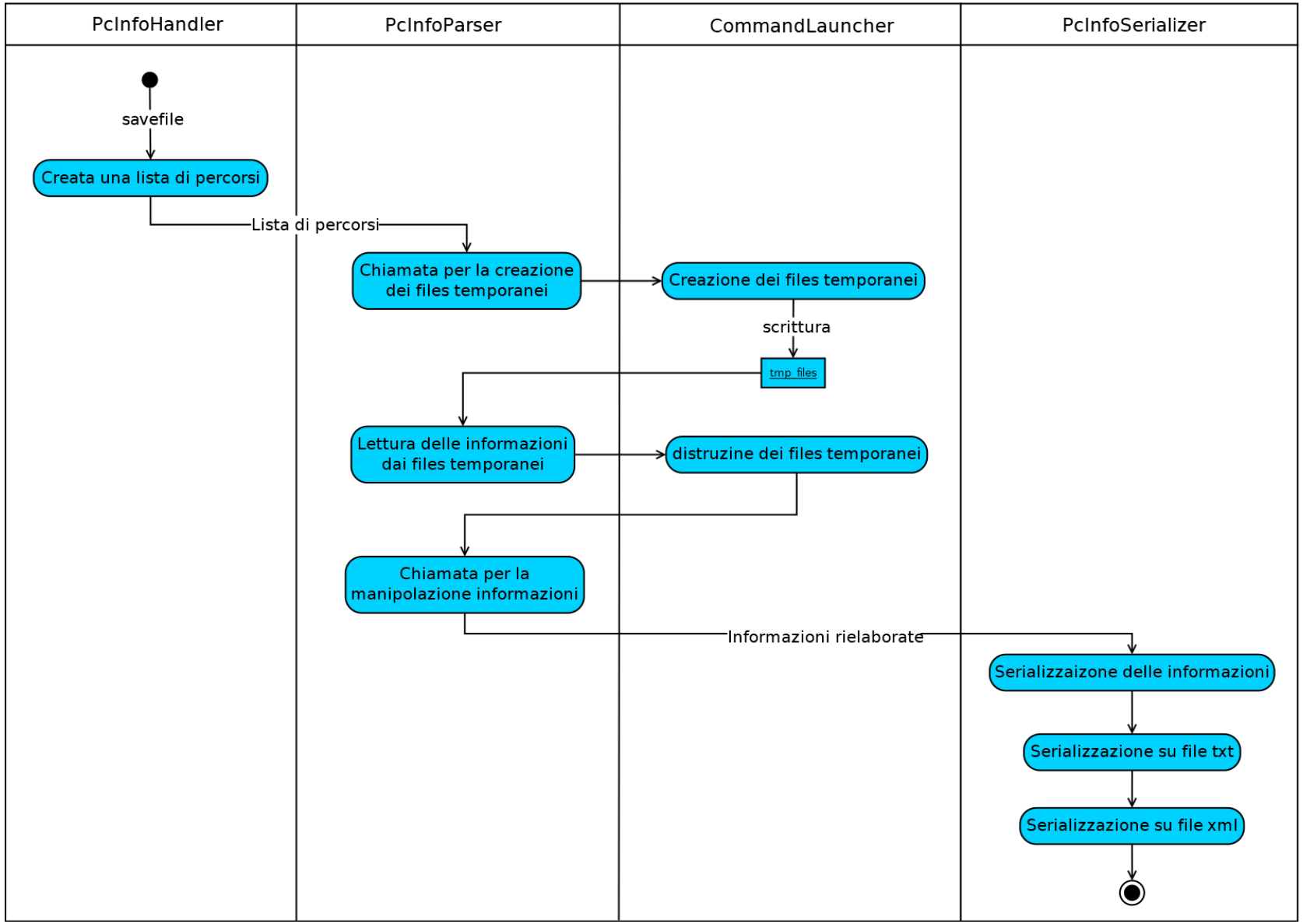
- Cenni teorici sull’integrazione
- Software sviluppati
- Conclusioni

3.1 Software per estrazione informazioni

Il primo software sviluppato serve per poter estrarre le seguenti informazioni da una macchina:

- Sistema operativo
- Scheda di rete
- CPU
- GPU
- Scheda Madre
- Driver delle schede di rete

Il funzionamento del software è descritto dal seguente *flow diagram*.


PcInfoHandler
PcInfoParser
CommandLauncher
PcInfoSerializer
savefile
Creata una lista di percorsi
Lista di percorsi
Chiamata per la creazione dei files temporanei
Creazione dei files temporanei
scrittura
tmp_files
Lettura delle informazioni dai files temporanei
distruzine dei files temporanei
Chiamata per la manipolazione informazioni
Informazioni rielaborate
Serializzaizone delle informazioni
Serializzazione su file txt
Serializzazione su file xml

3.1 Software per estrazione informazioni

I risultati ottenuti tramite l'utilizzo di questo software sono stati due file in formato TXT e XML. All'interno dei file sono state serializzate tutte le informazioni elencate in precedenza. Nelle immagini si possono notare solo le informazioni riguardo ai *driver* delle schede di rete

```
// DRIVERS //

deviceclass :   NET
driverversion   :   12.18.11.1
driverprovidername  :   Intel
devicename  :   Intel(R) I210 Gigabit Network Connection
location    :   PCI bus 13, device 0, function 0

DRIVERS-2

deviceclass :   NET
driverversion   :   12.18.11.1
driverprovidername  :   Intel
devicename  :   Intel(R) I210 Gigabit Network Connection
location    :   PCI bus 12, device 0, function 0
```

```
<DRIVERS>
  <deviceclass>NET</deviceclass>
  <driverversion>12.18.11.1</driverversion>
  <driverprovidername>Intel</driverprovidername>
  <devicename>Intel(R) I210 Gigabit Network Connection</devicename>
  <location>PCI bus 13, device 0, function 0</location>
</DRIVERS>
<DRIVERS2>
  <deviceclass>NET</deviceclass>
  <driverversion>12.18.11.1</driverversion>
  <driverprovidername>Intel</driverprovidername>
  <devicename>Intel(R) I210 Gigabit Network Connection</devicename>
  <location>PCI bus 12, device 0, function 0</location>
</DRIVERS2>
```

3.1 Software per estrazione informazioni

L’integrazione di tale software è stata possibile tramite entrambe le metodologie, quindi sia tramite la creazione di un eseguibile che tramite l’utilizzo di IronPython.

Il processo integrativo tramite l’utilizzo di IronPython è stato composto di tre passaggi :

1. Preparazione dei percorsi
2. Creazione dell’*engine* e dello *scope*
   - L’*engine* di IronPython è un contesto di esecuzione separato dagli altri
   - Lo *scope* rappresenta un namespace creato dall’engine
3. *Variable injection* e utilizzo di classi Python
   - Tramite i metodi GetVariable e SetVariable è possibile utilizzare le classi come se fossero appartenenti a C#

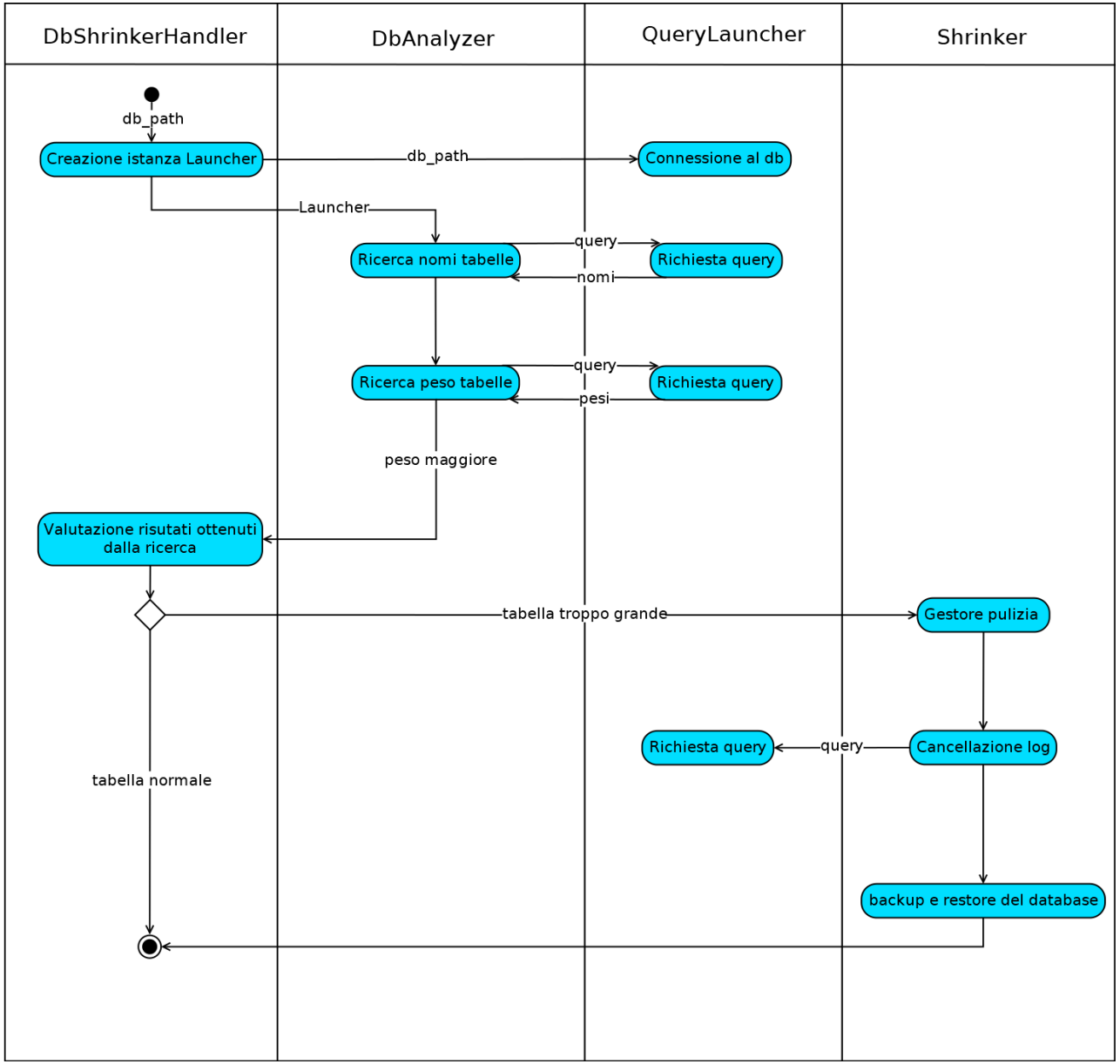
3.2 Software per pulizia di un database

Il secondo software è stato creato per automatizzare la pulizia di un database Firebird detta *shrinking*.

Lo *shrinking* è composto da tre passaggi :

- Cancellazione dei Log dalla tabella *TAB_Logs*
- Backup del database
- Restore del database

Il funzionamento del programma è descritto dal diagramma seguente


DbShrinkerHandler
DbAnalyzer
QueryLauncher
Shrinker
db_path
Creazione istanza Launcher
db_path
Connessione al db
Launcher
Ricerca nomi tabelle
query
Richiesta query
nomi
Ricerca peso tabelle
query
Richiesta query
pesi
peso maggiore
Valutazione risutati ottenuti dalla ricerca
tabella troppo grande
Gestore pulizia
Richiesta query
query
Cancellazione log
tabella normale
backup e restore del database

3.2 Software per pulizia di un database

Per garantire il corretto funzionamento di questo software è stato necessario utilizzare una libreria chiamata *fdb*. Tale libreria consente un supporto per la connessione con i database Firebird.

Il problema maggiore dell'utilizzo di questa libreria è che essa non è compatibile con IronPython, per questo non è stato possibile integrare tale software con questo metodo.

Il metodo utilizzato è stata la creazione di un eseguibile. L'integrazione è stata possibile utilizzando il software PyInstaller seguendo questi passaggi passaggi:

- Preparazione dei percorsi per l'utilizzo del software
- Creazione di un nuovo processo
  - Nel quale inserire il comando e le opzioni per eseguire il programma precedentemente compilato

- Cenni teorici sull’integrazione
- Software sviluppati
- Conclusioni

4.1 Misurazione delle performance

Per poter testare la velocità di esecuzione dei due meccanismi è stato utilizzato l'unico software con il quale è stato possibile effettuare entrambi i metodi di integrazione, ovvero il programma per l'estrazione delle informazioni. Prima di misurare le prestazioni è bene mostrare le caratteristiche hardware e software della macchina su cui sono stati eseguiti i test.

Caratteristiche	
Sistema operativo	Windows 10 Pro
Architettura	64 bit
Cpu	Intel Core i7-5820K
Clock	3.3 GHz
Numero di core	6
Gpu	NVIDIA GeForce GT 710
Ram	8 Gb
Memoria	SSD da 500 GB

4.1 Misurazione delle performance

Sono stati studiati due tipi di meccanismi di misurazione delle performance:

- Decoratore Python
- Classe *StopWatch* di C#

Il decoratore Python è una funzione che ne «decora» altre, aggiungendo funzionalità senza modificarne la struttura interna. Il decoratore che è stato creato è chiamato *timer* ed è stato utilizzato per misurare le prestazioni del software ancora da integrare.

La Classe *StopWatch* di C# è stata utilizzata per misurare l'effettiva velocità di esecuzione del software integrato secondo le due metodologie viste durante questo studio.

4.2 Misurazioni ottenute

Le misurazioni che sono state fatte hanno coinvolto entrambi i metodi di integrazione e sono state fatte utilizzando la classe *StopWatch*.

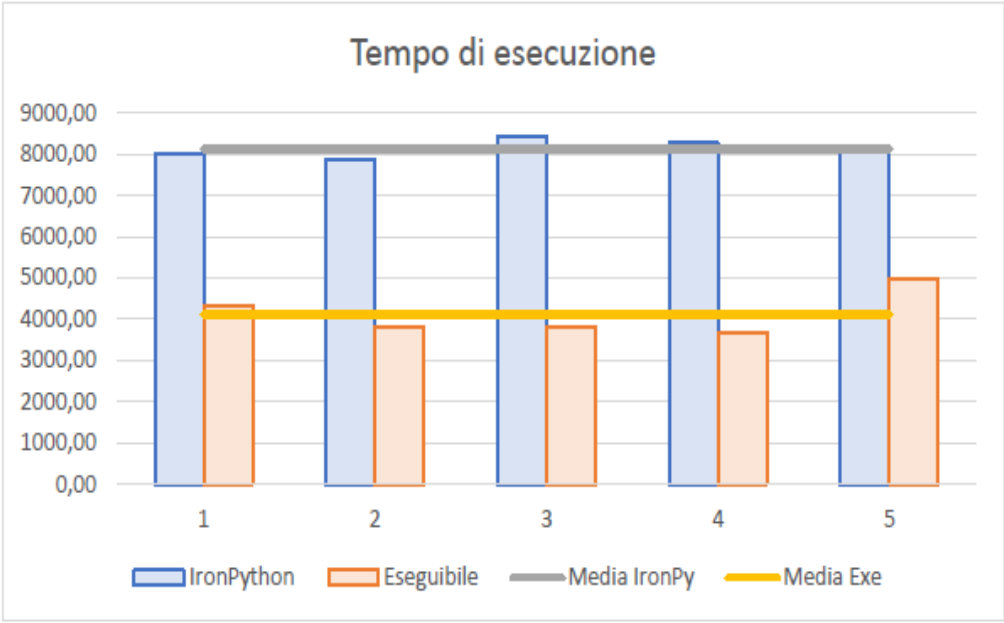
I test, per evitare un'alterazione dei risultati dovuta a processi interni alla macchina, sono stati ripetuti cinque volte. Sempre per lo stesso motivo si è cercato di chiudere la maggior parte dei processi e di isolare la macchina da connessioni esterne. Una volta ottenute le misurazioni, ne è stata fatta una media.

Prima di poter arrivare a un confronto vero e proprio dei risultati è necessario fare una premessa. IronPython sfrutta il meccanismo della DLR, ovvero un meccanismo di interpretazione, mentre PyInstaller esegue una vera e propria compilazione del codice. Nel grafico successivo è possibile osservare i risultati

4.2 Misurazioni ottenute

Le misurazioni sono state fatte in millisecondi.

IronPython	Eseguibile
8013,00	4337,00
7874,00	3803,00
8415,00	3809,00
8270,00	3675,00
8082,00	4978,00
Media :	Media :
8130,8	**4120,40**



Questo tipo di studio ha confermato non solo che l'eseguibile è risultato essere più veloce, ma lo è stato di un 50% in più.

5 Conclusioni

Tramite questo studio si è arrivati a capire come e in che modo è possibile integrare software. Arrivati a questo punto sorgono spontanee due domande :

- E’ veramente utile utilizzare un meccanismo di integrazione ?
- Qual è il meccanismo più efficace?

5 Conclusioni

Per quanto riguarda la prima domanda, è utile utilizzare l'integrazione, ma a determinate condizioni. Prima di poter utilizzare IronPython è necessario valutare se tutte le librerie che si ha intenzione di utilizzare siano compatibili con il software. Invece, indipendentemente dall'utilizzo di IronPython, l'integrazione è stata fondamentale per la creazione del secondo software analizzato.

La seconda domanda trova risposta nel livello di integrazione di cui abbiamo bisogno. L'utilizzo di IronPython è, come già detto, per definizione meno prestazionale rispetto all'utilizzo di un eseguibile, ma rende possibile una vera integrazione fra i due software.

Grazie dell’attenzione.