

UNIVERSITÀ DEGLI STUDI DI UDINE

LAUREA IN INFORMATICA

Importazione dati in ambiente Odoo

Laureando:

Eriberto MOMENTÈ

Relatore:

Dottor Giorgio BRAJNIK

Corso di laurea triennale in Informatica

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Settembre 2018

Acknowledgements

Questa mia tesi nonchè la conclusione di un mio percorso durato tre anni non sarebbe stata possibile se non grazie a una moltitudine di persone che mi hanno sostenuto.

- Ringrazio la mia famiglia, i miei fans numeri uno, che non hanno mai smesso di farmi il tifo e motivarmi per ambire al meglio.
- Alice, la mia compagna di avventure, che mi ha supportato con il suo amore e la sua stima fin dai primi giorni.
- Paolo Cazzitti, che ha avuto lo slancio accettarmi nel suo team più di una volta. Hai investito su di me molto tempo ed energie. Spero con questo lavoro di poterti dare soddisfazione.
- Al relatore di questa tesi, il professor Giorgio Brajnik.
- All'azienda Cogito Srl per l'opportunità di collaborazione e il clima piacevole che mi hanno sempre fatto respirare nella loro sede.
- Infine, ma non per ordine di importanza, ai miei amici. Non mi sono mai mancate le occasioni per festeggiare un buon risultato o trovare un'occasione per divertirsi!

Indice

Indice	v
1 Descrizione del Problema	3
1.1 Caratteristiche del problema	3
1.2 Come affrontare il problema	4
1.3 Possibili soluzioni	4
1.4 Cogito Srl e cenni all'esperienza di tirocinio	5
1.5 Chi potrebbe beneficiare del mio progetto	6
2 Odoo e il problema dell'importazione	7
2.1 Descrizione generale del framework	7
2.1.1 Storia e Evoluzione	7
2.1.2 Tutto il necessario all'interno di un'esperienza utente d'ecellenza	7
2.1.3 Novità introdotte e sfide preposte	8
2.1.4 I guadagni di Cogito da Odoo	9
2.2 Contesto di importazione dati	9
2.2.1 Il database di Odoo	9
2.2.2 L'architettura MVC in Odoo	10
2.3 Struttura di un modulo Odoo	10
2.4 Sviluppo di un modulo Odoo	11
2.4.1 __init__.py	12
2.4.2 __openerp__.py	12
2.4.3 Modelli Odoo	12
2.4.4 Viste Odoo	14
2.5 Strumenti per lo Sviluppo	16
2.5.1 Developer Mode	16
2.5.2 I logs	17
2.5.3 Installazione e aggiornamento di un modulo	18
3 Il mio progetto: synchronizer	21
3.1 I requisiti di Cogito	21
3.2 Estensione del progetto	22
3.2.1 Limiti del progetto sviluppato a Cogito	22

3.2.2	Introdurre scalabilità	22
3.3	Descrizione implementativa	22
3.3.1	Descrizione del File-System di synchronizer-main	23
	Le due classi: synchronizer e synchronizer_record	23
	Synchronizer - Viste	28
	Synchronizer - Controller	29
3.3.2	Descrizione del File-System di synchronizer-json	30
	synchronizer_record - model	31
	synchronizer_json - views	32
3.4	Esempi e screenshot	33

Dedicato alla mia mamma

Introduzione

Al giorno d'oggi i sistemi informatici sono ormai diffusi in ogni settore industriale. È impensabile l'idea di creare e/o supportare un'attività aziendale senza strumenti informatici basilari atti alla gestione della contabilità, dello sviluppo, delle analisi sulle risorse e molto altro ancora. Questo fatto comporta la diffusione di innumerevoli strumenti e piattaforme che elaborano e si scambiano flussi di dati per semplificare il lavoro di ognuno di noi. Siamo ormai entrati in un'epoca in cui scorrono enormi flussi di dati. Ogni concetto, informazione o anche ogni cosa che possiamo materialmente toccare è esprimibile con un dato, una rappresentazione, un'astrazione. Per ogni trasmissione che vediamo in televisione, chiamata che riceviamo al cellulare o email sul nostro portatile, ci sono giunti inconsapevolmente flussi di dati che seguendo protocolli e arrivando da chissà quale parte del mondo sono stati *importati* nei dispositivi che teniamo ogni giorno in mano.

Capitolo 1

Descrizione del Problema

Quando ci si pone in una realtà produttiva con del nuovo software che ha la necessità di interfacciarsi con il mondo esterno, ovvero realtà che non nascono con noi, sorge un problema. Solitamente quando si va ad installare del nuovo software, non capita sempre di incontrare una start-up nata il giorno prima, bensì un'azienda, spesso, che ha la sua storia e per questo ha quantomeno la necessità di interfacciarsi con una base dati esistente. Una base dati spesso molto eterogenea. Come spiega il project manager di Cogito, spesso la base dati delle piccole aziende soprattutto nel nostro territorio è basata sui fogli di calcolo Excel. Di conseguenza quando ci si trova a inserire un nuovo strumento gestionale, un ERP, che si pone la sfida di gestire a 360 gradi il funzionamento dell'azienda, si incontra il problema di dover importare e gestire un flusso di dati che già esiste. Stiamo parlando di dati sotto forma di decine o centinaia di fogli di calcolo, database Access oppure piattaforme più strutturate come magari altri gestionali che vanno sostituiti o con cui bisogna interfacciarsi. Un esempio molto calzante e ricorrente è il sistema di gestione delle paghe che spesso è un sistema di terze parti, ovvero di aziende specializzate nella gestione delle paghe. Se si vuole acquisire quei dati nasce il bisogno di fare un'importazione ma non ci si può permettere di importare tutta la base di dati di interesse con un'importazione manuale.

1.1 Caratteristiche del problema

Con lo sviluppo della tecnologia sono state create numerose piattaforme per la gestione dei dati, anche molto simili tra loro e nate spesso senza una logica di possibile scambio di dati. Sicuramente questa eterogeneità crea degli svantaggi quando ci si ritrova in scenari come quello descritto precedentemente. Ma cerchiamo di capire innanzitutto quali sono state le ragioni per cui nel tempo si è creata questa diversità:

- In prima ragione bisogna considerare fattori di evoluzione. Non ha senso continuare a supportare tecnologie obsolete o che non rispettano più le nuove esigenze del mercato a favore della compatibilità. Il progresso prima di tutto.

- Fattori economici. Esistono tecnologie con interessi proprietari e vantaggi commerciali. In certi casi, può costare di meno implementare da zero una tecnologia simile che pagarne i diritti a chi l'ha già implementata. O meglio ancora ri-implementare una tecnologia può fruttare ulteriori nuovi guadagni.
- Usi specifici. Esistono tecnologie create ad-hoc per un particolare sistema o hardware. Pensiamo ai sistemi embedded che necessitano di risparmi di risorse e di energia.
- Infine consideriamo fattori diversi, quali efficienza, flessibilità o sicurezza.

1.2 Come affrontare il problema

Come precedentemente illustrato, sorge la necessità di interfacciarsi con gli altri sistemi. Una soluzione semplice consiste nel fare un'importazione una-tantum se la mole di dati da importare è minima.

Una soluzione più ottimale, invece, prevede la costruzione di un connettore a lunga durata che continua ad aggiornare i dati da una sorgente esterna nel sistema informativo aziendale.

1.3 Possibili soluzioni

Un connettore è un'astrazione di un plug-in, ovvero un software che eredita del codice da un altro software e aggiunge delle funzionalità extra. Il connettore deve poter ricevere dei flussi di informazioni dall'esterno e immetterle nel sistema di cui fa parte. Spesso però l'ambiente esterno è una realtà che non nasce con il sistema in questione e quindi sfrutta standard, formati o dati incomprensibili al nostro sistema. La soluzione adottata maggiormente, secondo una fonte presa Wikipedia, a questo tipo di scenari è di sviluppare un exchange-format o semplicemente prenderne in considerazione uno esistente e scrivere qualche decina di routines diverse per tradurre ogni data-scheme nell'exchange-format poi facilmente convertibile con una sola routine in informazioni utili al nostro sistema. Questo richiede molto meno lavoro rispetto a sviluppare e debuggare centinaia di diverse routine che richiederebbero la traduzione diretta da ogni data-scheme ad ogni target-scheme. Nella figura 1.1 ogni freccia rappresenta una routine per convertire dei dati da un formato dati ad un altro. È visibile come nel secondo caso, inserendo un exchange-format nel mezzo il numero totale di routines diminuisca considerevolmente.

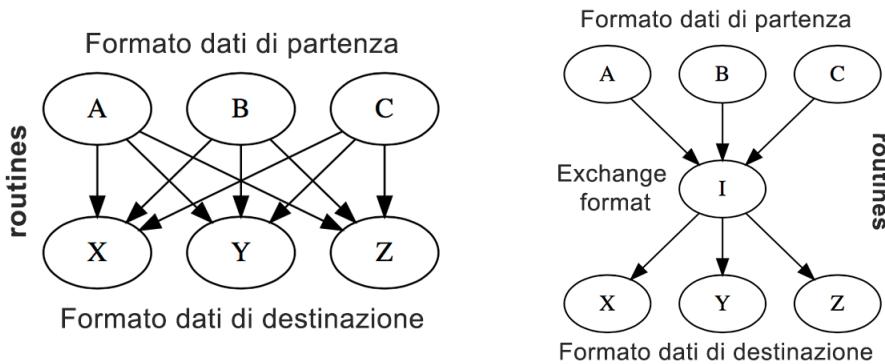


FIGURA 1.1: Il numero di routines di traduzione diminuisce con l'inserimento di un exchange-format

1.4 Cogito Srl e cenni all'esperienza di tirocinio

Cogito è una software house fondata nel 1991 con due vocazioni convergenti: lo sviluppo e l'interfaccia utente. Si occupa di creare soluzioni personalizzate per applicativi gestionali, app per mobile, integrazione di sistemi aziendali complessi. Cogito è caratterizzata da una costante attenzione verso le nuove tecnologie e le tendenze del mercato IT. Negli ultimi anni, con l'ingresso del loro nuovo socio e presidente, Cogito ha assunto una vocazione orientata alla fornitura di servizi per aziende in particolare per l'automazione industriale. Gli ultimi progetti prevedono lo sviluppo di una suite di prodotti dedicati al mondo industriale con l'idea poi di portare questo modello verso altre realtà.

Ho avuto il piacere di collaborare con il team di Cogito nel corso di circa due anni, sia in occasioni personali (extra-universitarie), sia tramite il tirocinio universitario.

Fin dal primo giorno di collaborazione con Cogito, nel giugno del 2016, mi è stata presentata la proposta di iniziare a lavorare con un nuovo strumento su cui l'azienda avrebbe voluto investire dal quel momento: Odoo, un applicativo gestionale open-source.

Nel corso di tutta l'estate del 2016 ho avuto modo, quindi, di imparare a sviluppare semplici moduli custom per Odoo oltre che apprendere nozioni a quel tempo per me sconosciute. È proprio a Cogito, per esempio, che ho avuto i primi cenni su che cos'era il paradigma Model-View-Controller, ho imparato a utilizzare strumenti per la gestione di versione come Git e, banalmente, ho avuto modo di osservare dall'interno l'organizzazione di una software-house.

Nel corso dell'estate successiva, raccogliendo i frutti del lavoro fatto l'anno prima e sfruttando le conoscenze acquisite nel secondo anno accademico, ho avuto modo di addentrarmi maggiormente nello sviluppo di moduli Odoo, preparando le

basi per questa tesi. È stato proprio in quell'estate che ho cominciato a sviluppare il progetto di questa tesi: un modulo di importazione dati in Odoo.

Questo progetto è stato da me portato a termine, in un primo momento, in quell'estate per fornire a Cogito la soluzione di cui aveva bisogno e successivamente migliorato nel corso del mio tirocinio universitario al fine di renderlo un progetto più completo e scalabile.

1.5 Chi potrebbe beneficiare del mio progetto

Dato che Odoo è un ERP strutturato a moduli, è in grado di coprire o lo è potenzialmente, se sviluppato, una qualsiasi realtà aziendale, dalla gestione del personale, alle vendite, ai magazzini, tutto quello che ha a che fare con una realtà aziendale. Quindi chiunque abbia intenzione di acquisire l'Odoo all'interno del proprio sistema informativo aziendale e metterlo in relazione con dati già esistenti o provenienti da fonti esterne potrà usufruire del mio progetto.

Capitolo 2

Odo e il problema dell'importazione

2.1 Descrizione generale del framework

Gli ERP, Enterprise Resource Planning, di ultima generazione stanno cominciando ad essere intesi più come framework piuttosto che dei prodotti finiti. Questo perché le realtà aziendali soprattutto se non sono delle micro-realtà, hanno il bisogno sempre più spesso di adeguare il software gestionale ai propri influssi.

2.1.1 Storia e Evoluzione

Odo nasce nel 2005 ad opera di Fabien Pinckaers con il proposito di rivoluzionare il mondo del software per le imprese. Tre anni dopo il nome venne cambiato in OpenERP. L'azienda cominciò subito a svilupparsi fino ad arrivare nel 2010 a contare più di cento dipendenti.

Nel 2014 l'azienda si spostò oltre le tradizionali barriere di un ERP e per questo proposito scelse un nuovo nome che non imponesse vincoli a un'evoluzione in un'altra qualsiasi direzione: Odo - On Demand Open Object. Nel 2015, Inc. Magazine classificò Odo tra le 5000 aziende private con la più alta crescita in Europa.

2.1.2 Tutto il necessario all'interno di un'esperienza utente d'eccellenza

Oggi Odo è una potente piattaforma gestionale che offre un'ampia gamma di applicazioni aziendali adattabili ad ogni esigenza. È un software aziendale *all-in-one* che può includere CRM, sito web/e-commerce, gestione delle risorse, magazzino, inventario ed eventualmente supportare il project management. Tutte queste funzionalità permettono ad Odo di definirsi un ERP, Enterprise Resource Planning, ovvero un software di pianificazione delle risorse aziendali. Il suo modello open-source ha permesso a migliaia di sviluppatori ed esperti di business di costruire centinaia di

applicazioni in pochi anni. Con solide fondamenta tecniche, la struttura di Odoo è unica. Fornisce usabilità eccellente che si adatta a tutte le applicazioni. In questo modo, si evolve molto più velocemente di qualsiasi altra soluzione.

2.1.3 Novità introdotte e sfide preposte

Tralasciando le funzionalità di base di un ERP, che Odoo ha sempre a disposizione, sono stati sviluppati oltre 14 mila applicazioni, o plug-in, di terze parti chiamati in gergo tecnico *moduli*. Ognuno di essi è stato sviluppato per un'esigenza specifica di una o più aziende e oggi, Odoo, si può considerare una delle soluzioni ERP open-source più usate nel mercato. Ma quali sono i punti di forza che hanno reso Odoo sempre più diffuso? Ecco alcune caratteristiche:

- È open-source, non ci sono costi proprietari
- Flessibilità. Se si necessita di alcune features personalizzate si possono semplicemente sviluppare con l'aiuto di uno sviluppatore.
- Scalabilità. La dimensione dell'azienda non è un fattore vincolante, si può aggiungere un numero indeterminato di utenti.
- Supporto globale attraverso forum di utenti e sviluppatori e mailing list.
- È un prodotto già ampiamente diffuso e testato. Più di 3 milioni di persone stanno facendo crescere il loro business supportati da Odoo. Qualche nome? Toyota e Hyundai.
- È user-friendly. Oltre all'interfaccia standard di Odoo si possono scaricare o sviluppare completamente nuovi temi da applicare.
- Sempre aggiornato con nuove versioni di sistema e al passo con le tecnologie esistenti.
- Altamente modulare. Non bisogna installare l'intero ambiente Odoo nel proprio sistema, è sufficiente selezionare e installare solamente le applicazioni utili per la propria attività.
- Facilmente integrabile con servizi e applicazioni di terze parti.
- Ha un basso TCO - Total Cost of Ownership - rispetto a un classico ERP. Spesso le aziende non riescono a vedere completamente il potenziale di un sistema ERP sviluppato ad-hoc poiché limitati dal costo di investimento iniziale e dai preventivi dei costi di manutenzione. Odoo dunque è il giusto compromesso che offre un sistema moderno, adattabile, facile da installare e usare senza enormi investimenti iniziali.

La sfida di Odoo per il futuro è quella di ampliare la sua gamma di servizi, in linea con le innovazioni fatte finora, per adattarsi sempre più alle diverse realtà aziendali nel mondo arrivando probabilmente a poter offrire moduli per ogni esigenza, senza il bisogno di uno sviluppo ad-hoc.

2.1.4 I guadagni di Cogito da Odoo

Cogito è stata subito molto attratta dalla versatilità, modularità e l'alta personalizzazione che Odoo è in grado di offrire. Vediamo ora come questa azienda ha spostato una parte del suo business sulla piattaforma.

Odoo offre la possibilità di ottenere dei guadagni in due modi; Cogito li sfrutta entrambi:

- Attraverso lo sviluppo di moduli personalizzati be-spoken.
- Mediante la vendita di formazione e manutenzione del sistema. L'azienda Odoo, stessa, fornisce il servizio di hosting della piattaforma e Cogito a sua volta lo dispone per le aziende.

2.2 Contesto di importazione dati

2.2.1 Il database di Odoo

Odoo ha una struttura molto intima e integrata con il suo database al punto che, come ci suggerisce il project manager di Cogito, possiamo considerarlo il suo vero motore. Il database a cui ci stiamo riferendo è un database relazionale basato su PostgreSQL che Odoo sfrutta con una logica estremamente transazionale. Che cosa significa? Esistono sistemi contrari che non lavorano in maniera transazionale che eseguono le operazioni man mano che il processo scorre. Se si verifica un errore a metà processo, ci troviamo in una situazione che metà del lavoro è stato fatto e metà no, di conseguenza abbiamo dei dati inconsistenti. Un software legacy solitamente lavora così e non gestisce il flusso delle transazioni in maniera atomica. Odoo invece è un sistema transazionale, ovvero ogni operazione viene racchiusa in un'unità elementare di lavoro a cui si vogliono associare particolari caratteristiche di correttezza, robustezza ed isolamento. Un sistema come Odoo che mette a disposizione meccanismi per la definizione e l'esecuzione di transazioni viene detto sistema transazionale. Perfino l'apertura di una connessione è registrata sul database da Odoo; questo permette agli utenti di avere garanzie piuttosto forti. In più Odoo utilizza il database come strato di persistenza per tutte le sue operazioni all'infuori dei documenti/allegati salvati sul suo file system, perfino le variabili temporanee che un sistema concorrente potrebbe conservare in una memoria cache. Lo svantaggio di

avere un sistema di questo tipo è il conseguente aumento di carico del sistema e la necessità di un buon sistema hardware efficiente.

2.2.2 L'architettura MVC in Odoo

Il Model-View-Controller è un pattern architettonale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti. È basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

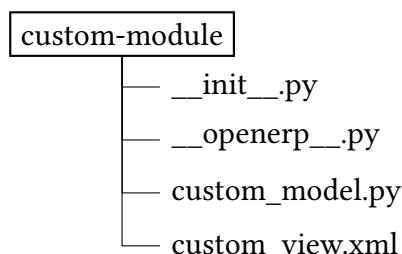
- il **model** si occupa di gestire i dati e la logica dell'applicazione
- la **view** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti
- il **controller** riceve i comandi dell'utente e li attua modificando lo stato degli altri due componenti

In Odoo questo schema architettonale si riflette nel seguente modo:

- i modelli vengono definiti in classi Python da cui vengono create le entità, in forma di tabella, nel database.
- le viste vengono definite nei file XML
- i controller possono essere già implementati da Odoo attraverso l'azione di bottoni presenti nell'interfaccia o completamente sviluppabili con codice Python.

2.3 Struttura di un modulo Odoo

Un modulo Odoo è sostanzialmente una cartella, denominata con il nome identificativo del modulo in questione, contenente una serie di file di base:



- **__init__.py** funge da entry-point per eseguire il codice Python del modulo

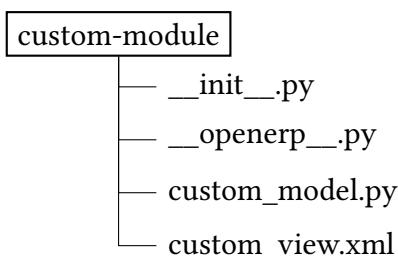
- **__openerp__.py** è il manifesto del modulo. Contiene un dizionario python con informazioni sul modulo, sui moduli da cui dipende e file che verranno caricati.
- **custom_model.py** è la classe Python del modulo con tutte le definizioni logiche del modello, informazioni sulla struttura della relativa tabella nel database e possibili funzioni d'utilità. Nel caso si abbia la necessità di definire più classi, si può creare una cartella `model` dove includerli ricordandosi di creare un ulteriore entry-point `__init__.py` per il server Odoo.
- **custom_view.xml** in cui viene definita l'interfaccia del modulo e le specifiche su come inserirla nella vista generale dell'ambiente Odoo. Anche in questo caso più viste possono essere raggruppate in una cartella `view` modificando i nuovi riferimenti ad esse nel file `__openerp__.py`.

Sebbene quella appena presentata sia la struttura necessaria e sufficiente per l'installazione di un modulo, può tornare utile ai fini della comprensione di questa tesi e non solo, presentare degli elementi *extra* che possono essere inclusi in un modulo Odoo:

- **custom_controller.py** o una cartella `controller` se si ha la necessità di gestire dei controllori personalizzati secondo una logica MVC completa.
- **static/description/icon.png** ovvero l'immagine 300x300 identificativa del modulo.
- **static/src/css/custom_css.css** fogli di stile che devono essere citati nel manifesto. Allo stesso modo è possibile includere, con un percorso parallelo, dei file Javascript.

2.4 Sviluppo di un modulo Odoo

Analizziamo ora le fasi per lo sviluppo di un modulo Odoo. Prendiamo in considerazione un modulo con il file-system minimo, come quello citato nella sezione precedente:



2.4.1 `__init__.py`

Come già detto questo file è l'entry-point per il server Odoo al codice python del nostro modulo `custom_model.py`.

```
# -*- coding: utf-8 -*-
import custom_model
```

2.4.2 `__openerp__.py`

Il manifesto di un modulo Odoo non è altro che un dizionario Python con alcuni attributi particolari. Nello specifico, include informazioni sul modulo come nome, descrizione, autore e altre informazioni più tecniche come eventuali dipendenze con altri moduli o il percorso di altri file da caricare, come le viste.

```
# -*- coding: utf-8 -*-
{
    'name': "Nome del Modulo",
    'summary': "Sommario del modulo",
    'description': "Descrizione di questo modulo",
    'author': "Eriberto",
    # Categories can be used to filter modules in modules listing
    'category': 'Manufacturing',
    'version': '0.1',
    # any module necessary for this one to work correctly
    'depends': [],
    # always loaded
    'data': [
        # views
        'custom_view.xml',
        # menus
        'custom_menu.xml',
    ],
    'application': True,
    'installable': True
}
```

2.4.3 Modelli Odoo

Il modello, come sappiamo, è il cuore della logica e della gestione dei dati nel database. Vediamone un esempio:

```
#-- coding: utf-8 --
from openerp import models, fields
class LibraryBook(models.Model):
    _name = 'library.book'
    name = fields.Char('Title', required=True)
    data_release = fields.Date('Release Date')
    author_ids = fields.Many2many('res.partner',
        string="Authors")
```

Analizzando il codice:

```
#-- coding: utf-8 --
```

È un semplice commento Python in cui specifichiamo la codifica con cui è scritto il codice. È buona norma specificarlo.

```
from openerp import models, fields
```

Permette di importare dalla classe `openerp` le definizioni degli oggetti `models` e `fields`

```
class LibraryBook(models.Model):
```

Viene definito l'oggetto di questa classe. I modelli Odoo sono oggetti che derivano dalla classe python `Odoo Model`. Quando un nuovo modulo viene definito, viene anche aggiunto al registro centrale dei modelli così da facilitare eventuali modifiche su di esso da parte di altri modelli. `models.Model` definisce alcuni metodi e attributi di base di un oggetto Odoo.

In Odoo, i campi dei modelli sono definiti come attributi della classe Python.

Alcuni di essi possono iniziare con un underscore:

```
_name = 'library.book'
```

`_name` fornisce un nome univoco ad Odoo come identificatore dell'oggetto.

A seguito possono esserci una serie di attributi di diverso tipo, ne ho presentati un paio di tipo `Char` e `Date`

```
name = fields.Char('Title', required=True)
data_release = fields.Date('Release Date')
```

Per ciascun campo viene definito un nome univoco (`name`, `data_release`) che lo indichi, delle etichette che verranno visualizzati sulla viste (`Title`, `Release Date`) ed eventuali opzioni come l'obbligo di inserimento in `name`.

Esistono anche attributi di tipo relazionale, come il seguente:

```
author_ids = fields.Many2many('res.partner', string="Authors")
```

Quest'ultimo definisce una relazione *ManyToMany* tra un oggetto di tipo `library.book` e uno di tipo `res.partner`. Un libro può avere diversi autori e un autore può scrivere diversi libri.

È importante sottolineare per ora che, sebbene non sia stato illustrato nell'esempio, in un file Python si può anche aggiungere tutta la business logic definendo o ereditando dei metodi. Nel capitolo 3, vedremo ulteriori esempi.

2.4.4 Viste Odoo

Una volta creato un buon modello per gestire i dati, è fondamentale avere una vista per poter interfacciarsi con essi. Un modello può ovviamente avere diverse viste, è consigliabile in quel caso raggrupparle in una cartella `views` e specificare i collegamenti ad esse nel file `__openerp__.py`. Odoo stesso mette a disposizione diversi tipi di viste: tree, form, kanban; noi vedremo un esempio di form.

La cornice di una vista Odoo è la seguente:

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
    <data>
        <!-- Data records go here! -->
    </data>
</openerp>
```

All'interno inseriamo il codice XML per la visualizzazione del form del nostro modello e dei suoi campi, definiti precedentemente

```
<record id="library_book_view_form" model="ir.ui.view">
    <field name="name">Library Book Form</field>
    <field name="model">library.book</field>
    <field name="arch" type="xml">

        <form>
            <group>
                <field name="name" />
                <field name="author_ids" widget="many2many_tags" />
```

```

</ group>
<group>
    <field name="date_release" />
</ group>
</ form>

</ field>
</ record>
```

Analizziamo ora riga per riga:

```
<record id="library_book_view_form" model="ir.ui.view">
```

Questo tag contiene l'intera vista. Come si può dedurre, viene specificato un id e il modello a cui appartengono questi dati ovvero il modello delle viste Odoo, `ir.ui.view`.

```

<field name="name">Library Book Form</field>
<field name="model">library.book</field>
```

In queste due righe sono state definite: il nome della vista e il modello di riferimento a cui corrispondono i dati di questo form.

```

<field name="arch" type="xml">
    ...
</field>
```

Questo tag, invece, contiene i dati veri e propri su com'è strutturata la vista. Nel nostro caso abbiamo organizzato i campi del modello in due colonne identificate dal sotto-tag `<group>`.

Vediamo ora, invece, un esempio di come collegare la nostra vista appena creata con una voce di menù.

Prima di tutto occorre creare un'*Azione* che apra la vista:

```
<act_window
    id="library_book_action"
    name="Library_Books"
    res_model="library.book"
    view_mode="form" />
```

E solo successivamente aggiungere la voce di menu rendendola visibile agli utenti:

```
<menuitem
    id="library_book_menu"
    name="Library"
    action="library_book_action"
    parent=""
    sequence="5" />
```

2.5 Strumenti per lo Sviluppo

2.5.1 Developer Mode

Odoo mette a disposizione alcuni strumenti per facilitare il lavoro degli sviluppatori come la *Developer mode* disabilitata di default e attivabile solo con particolari permessi, di norma attribuiti all'amministratore di sistema. Per Attivare la Developer Mode bisogna aprire la pagina *Configurazione Amministrazione* → *Informazioni sull'Odoo* e cliccare su *Attiva la modalità sviluppatore*.

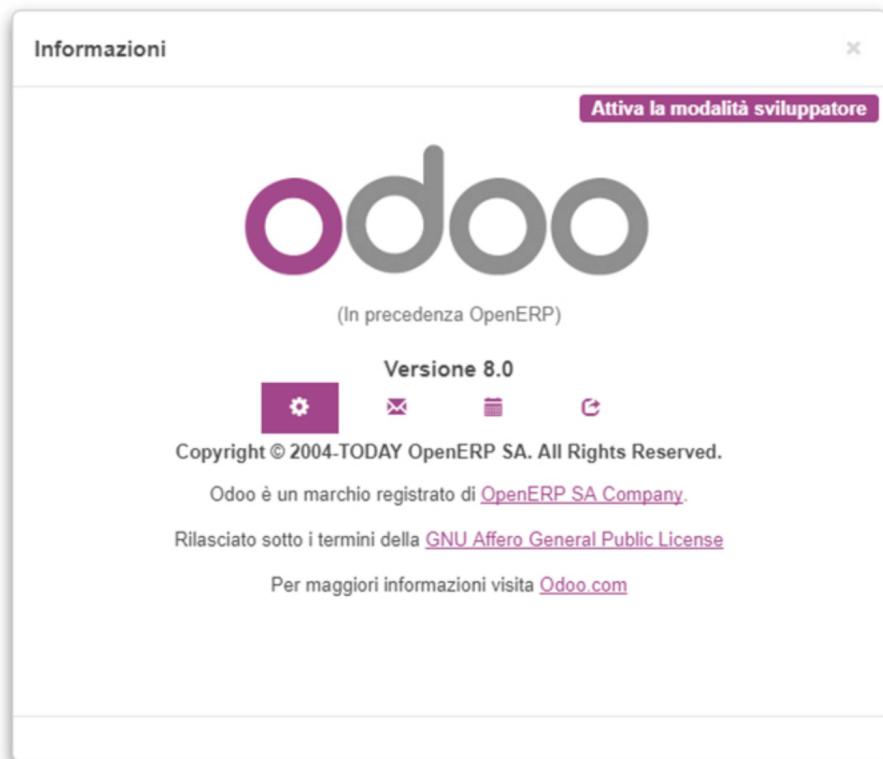


FIGURA 2.1: Come attivare la developer mode

La modalità sviluppatore in Odoo permette di visualizzare informazioni utili per il debugging. Personalmente mi torna molto comoda nelle seguenti situazioni: ottenere l'ID della vista che visualizzo, vedere quali sono le proprietà dei campi del modello corrente (esempio se è un campo è required o nascosto) o capire quale sia la relativa tabella del database su cui sto modificando dei dati.

2.5.2 I logs

I server log sono utili per capire cosa succede all'esecuzione del programma poco prima di un errore o un evento non previsto. Possono essere usati, ed è consigliabile farlo, anche per fornire ulteriori informazioni sull'esecuzione del programma, ad esempio stampare quale dei due rami di un *if* il programma sta eseguendo. In quest'ultimo caso queste informazioni non servono solo al programmatore che ha sviluppato il programma per fare del debugging, ma sia a lui per avere un livello di chiarezza e controllo più alto sull'esecuzione del suo codice, sia a chi potrebbe in futuro dover usare il codice e capire come funziona.

Odoo registra di default alcune sue operazioni in un log-file, accessibile mediante terminale. Con le istruzioni a seguito mostrerò come stampare una stringa in quel file.

- Nel file Python in cui si vuole inserire un log importare il modulo *logging*:

```
import logging
```

- Prima della definizione della classe, creiamo un logger:

```
_logger = logging.getLogger(__name__)
```

La variabile `__name__` è impostata automaticamente dall'interprete Python al momento dell'importazione dei moduli e si prevede che contenga il nome completo del modulo in questione.

- stampare la stringa contenuta in `mystring` con uno dei seguenti metodi:

```
_logger.debug(mystring)
_logger.info(mystring)
_logger.warning(mystring)
_logger.error(mystring)
_logger.critical(mystring)
_logger.exception(mystring)
```

A seconda del livello di log che è stato configurato vengono visualizzati, o meno, certi metodi.

Per visualizzare il contenuto del file in cui vengono raccolti tutti i log è opportuno spostarsi nella directory in cui è contenuto il file. Nella mia installazione:

```
cd /srv/webapp/instances/odoo/prod/buildout/var  
e invocare il seguente comando:
```

```
tail -f log.log
```

In questo modo verranno visualizzati in tempo reale gli ultimi messaggi stampati.

```
HTTP/1.0" 200 -
2018-09-18 19:55:18,459 1619 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:18] "POST /web/dataset/call_kw/synchronizer/fields_get
HTTP/1.0" 200 -
2018-09-18 19:55:18,503 1620 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:18] "POST /web/dataset/call_kw/synchronizer/fields_vie
w_get HTTP/1.0" 200 -
2018-09-18 19:55:18,506 1619 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:18] "POST /web/dataset/search_read HTTP/1.0" 200 -
2018-09-18 19:55:19,699 1620 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:19] "POST /web/dataset/call_kw/synchronizer/fields_vie
w_get HTTP/1.0" 200 -
2018-09-18 19:55:19,954 1620 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:19] "POST /web/dataset/call_kw/synchronizer/read HTTP/
1.0" 200 -
2018-09-18 19:55:20,068 1619 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:20] "POST /web/dataset/search_read HTTP/1.0" 200 -
2018-09-18 19:55:20,159 1620 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:20] "POST /web/dataset/call_kw/ir.attachment/get_attac
ment_external HTTP/1.0" 200 -
2018-09-18 19:55:21,113 1622 INFO eplaundry openerp.modules.loading: loading 1 modules...
2018-09-18 19:55:21,127 1622 INFO eplaundry openerp.modules.loading: 1 modules loaded in 0.01s, 0 queries
2018-09-18 19:55:21,387 1622 INFO eplaundry openerp.modules.loading: loading 158 modules...
2018-09-18 19:55:21,442 1622 INFO eplaundry openerp.addons.report: Will use the Wkhtmltopdf binary at /usr/local/bin/wkhtmltopdf
f
2018-09-18 19:55:22,244 1622 INFO eplaundry openerp.modules.loading: 158 modules loaded in 0.94s, 0 queries
2018-09-18 19:55:23,186 1619 INFO eplaundry openerp.addons.synchronizer.main.model.synchronizer: sync function
2018-09-18 19:55:23,188 1619 INFO eplaundry openerp.addons.synchronizer.json.model.synchronizer_json: inherit sync function: JSON
2018-09-18 19:55:23,189 1619 INFO eplaundry openerp.addons.synchronizer.main.model.synchronizer: http://192.168.56.101/test_generator?data_fo
rmat=json&model=res.bank&last_id=&last_date='2000-01-01'
2018-09-18 19:55:23,203 1620 INFO eplaundry openerp.addons.synchronizer.main.controller.API_controller: calling test generator
2018-09-18 19:55:23,203 1620 INFO eplaundry openerp.addons.synchronizer.main.controller.API_controller: calling test generator
2018-09-18 19:55:23,217 1619 INFO eplaundry openerp.addons.synchronizer.main.model.synchronizer: create ext_id 890 {u'name': u'API_test_m7Neh
U'}
2018-09-18 19:55:23,234 1619 INFO eplaundry openerp.addons.synchronizer.main.model.synchronizer:
-----
Import result:
  1 records read
  1 inserts
  0 updates
  0 errors
  error ids: []

2018-09-18 19:55:23,238 1619 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:23] "POST /web/dataset/call_button HTTP/1.0" 200 -
2018-09-18 19:55:23,265 1620 INFO eplaundry werkzeug: 127.0.0.1 - - [18/Sep/2018 19:55:23] "POST /web/menu/load_neadction HTTP/1.0" 200 -
2018-09-18 19:55:23,359 1622 INFO eplaundry openerp.modules.loading: Module loaded
2018-09-18 19:55:23,367 1622 INFO eplaundry openerp.modules.loading: 127.0.0.1 - - [18/Sep/2018 19:55:24] "POST /calendar/notify HTTP/1.0" 200 -
2018-09-18 19:57:27,528 1622 INFO eplaundry openerp.modules.loading: loading 1 modules...
2018-09-18 19:57:27,535 1622 INFO eplaundry openerp.modules.loading: 1 modules loaded in 0.01s, 0 queries
2018-09-18 19:57:27,715 1622 INFO eplaundry openerp.modules.loading: loading 158 modules...
2018-09-18 19:57:27,808 1622 INFO eplaundry openerp.modules.loading: 158 modules loaded in 0.08s, 0 queries
2018-09-18 19:57:28,993 1622 INFO eplaundry openerp.modules.loading: Modules loaded.
```

FIGURA 2.2: Esempio di log generati parte da Odoo e parte dal modulo synchronizer

2.5.3 Installazione e aggiornamento di un modulo

Una volta sviluppato un nuovo modulo cosa occorre fare per installarlo e poterlo utilizzare?

È abbastanza semplice ma occorre sapere alcune cose.

Quando si sviluppa un nuovo modulo, che come abbiamo già detto è un insieme di file raggruppati in una cartella con il nome del modulo, occorre posizionarlo nella cartella dei moduli di Odoo oppure creare un link simbolico (comando `ln -s source_path dest_path`) verso di essa.

Odoo però, non prevede di tenere traccia automaticamente della lista dei moduli contenuti in essa, ragion per cui occorre effettuare una scansione della lista dei moduli con la seguente procedura:

Configurazione → Aggiorna lista moduli

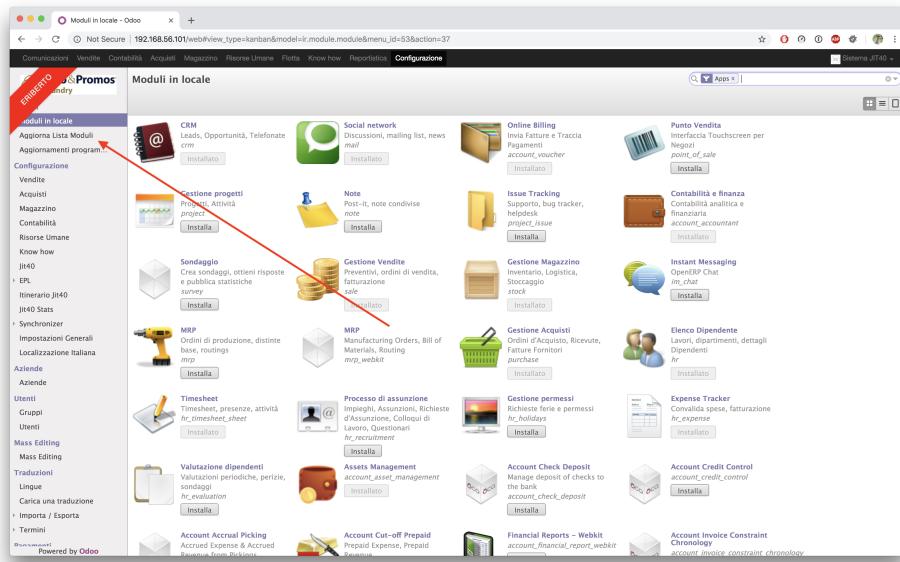


FIGURA 2.3: Voce di menù Aggiorna lista moduli

Successivamente, andando sulla pagina in *Configurazione → Moduli locali* potremo trovare il nostro modulo e cliccando su di esso arriviamo alla schermata di presentazione del modulo. A questo punto com'è facilmente intuibile basta cliccare su *Installa*.

Una volta installato un modulo con successo, nella schermata di presentazione del modulo, al posto del pulsante *Installa* verrà visualizzato il pulsante *Aggiorna*.

Nella fase di sviluppo di un modulo, è opportuno aggiornare un modulo ogni volta che occorre caricare un nuova vista o si è fatto delle modifiche su una di esse e occorre caricare la nuova versione.

Per visualizzare le modifiche effettuate su un modello Python, invece, occorre riavviare il server Odoo da terminale.

Fogli di stile e file Javascript aggiuntivi vengono invece caricati ad ogni refresh della pagina.

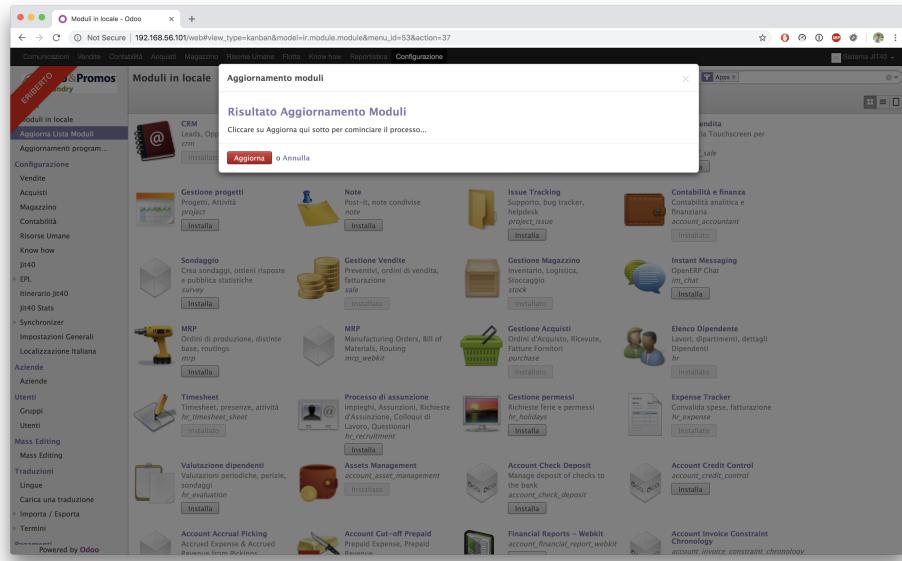


FIGURA 2.4: Aggiorna lista moduli

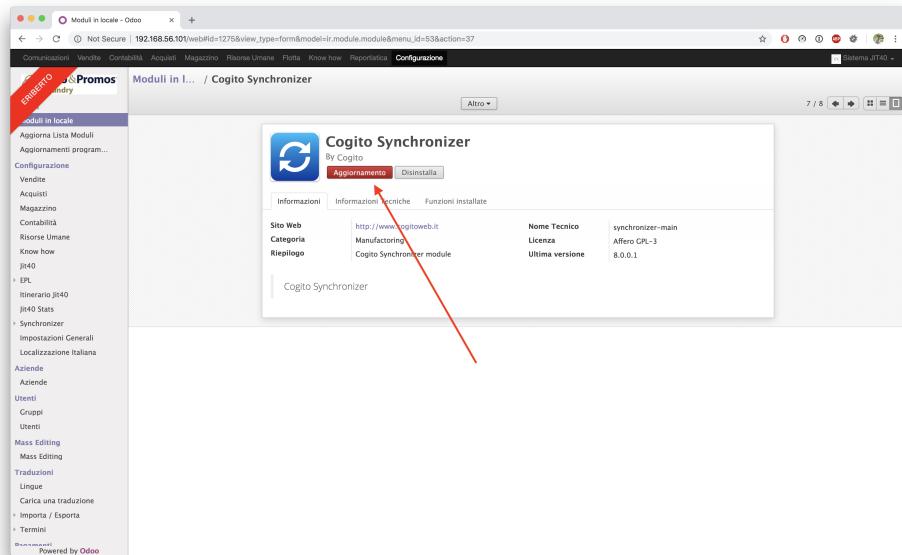


FIGURA 2.5: Voce di menù Aggiorna lista moduli

Capitolo 3

Il mio progetto: synchronizer

3.1 I requisiti di Cogito

Nell'estate del 2017 mi venne affidato il progetto di creare un modulo Odoo in grado di importare dati da risorse memorizzate in locale o che venissero inviate da un server remoto. Il modulo si aspettava che i dati da importare fossero sottoforma di oggetti Json e codificassero oggetti di modelli già esistenti nell'installazione Odoo corrente. Il modulo doveva più precisamente caricare o ricevere l'oggetto Json, controllare se quell'oggetto fosse già stato importato e in caso negativo crearne un clone sul database, altrimenti, in caso positivo, aggiornare i record corrispondenti nel database qualora ci fossero attributi differenti. Allo stesso modo della creazione, il modulo doveva poter eliminare dal database gli oggetti importati.

L'importazione dei modelli poteva essere ricorsiva, ovvero se l'attributo di un oggetto principale consisteva in un altro oggetto, chiamiamolo secondario, si richiedeva di poter prima importare, e quindi creare, l'oggetto secondario per poi importare l'oggetto principale in questione e creare il riferimento tra i due. Allo stesso tempo se si fosse deciso di eliminare dal database Odoo l'oggetto principale importato occorreva prima eliminare tutte le referenze all'oggetto secondario oltre che l'oggetto stesso, per poi eliminare quello principale senza creare inconsistenza nei dati.

Il modulo doveva prevedere anche una serie di funzionalità aggiuntive quali una Basic Autentication qualora il server remoto richiedesse delle credenziali per inviare i dati, degli header opzionali da aggiungere nella chiamata e la possibilità di associare un *cron*, ovvero uno scheduler time-based per aggiornare periodicamente e automaticamente i dati delle importazioni.

3.2 Estensione del progetto

3.2.1 Limiti del progetto sviluppato a Cogito

Il modulo sviluppato a Cogito rispondeva alle esigenze dell’azienda tuttavia presentava un grosso limite: importava solo dati in formato Json. Se un server remoto fosse programmato per inviare dati solo in formato XML o se si avesse la necessità di importare dati da un file in CSV, il mio modulo era inutile.

Il limite principale del modulo sviluppato non riguardava tanto la scelta di quale formato si fosse scelto per importare i dati, quanto la nulla scalabilità del progetto. Se si avesse avuto l’esigenza di importare dati in XML occorreva ricreare un nuovo modulo da zero copiandone parte delle funzioni principali, quali la creazione/eliminazione di record nel database, i controller per l’interazione e buona parte dell’interfaccia utente. Così per ogni formato dati che si occorreva importare.

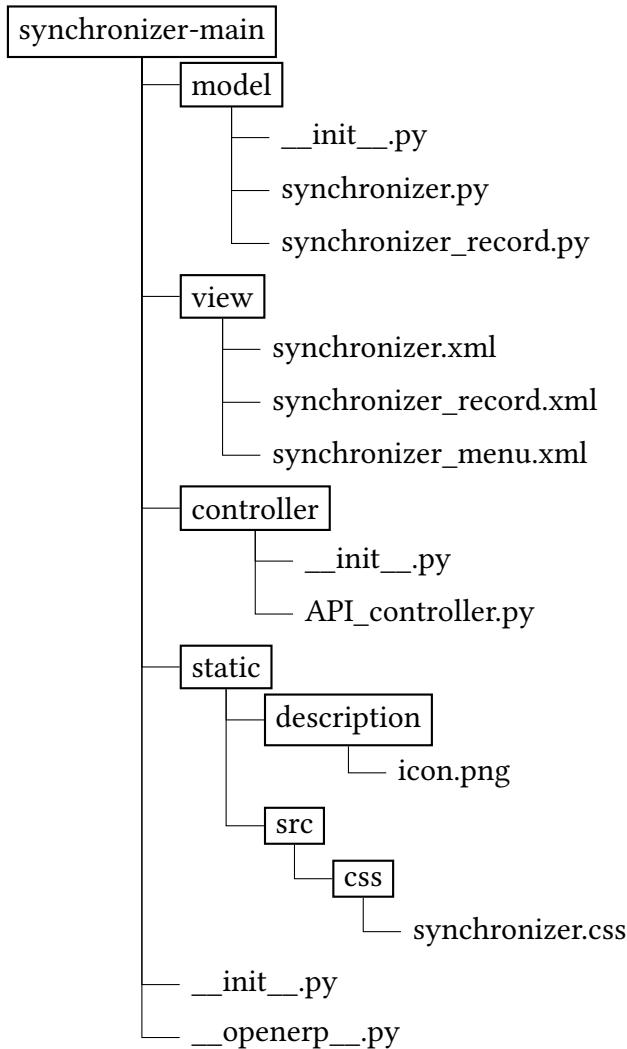
3.2.2 Introdurre scalabilità

L’idea proposta dal relatore di questa tesi, è stata quindi quella di sviluppare un modulo principale agnostico a qualsiasi formato dati, contenente tutte le funzioni e le viste comuni necessarie a importare un qualsiasi dato contenuto in un dizionario Python. Successivamente creare diversi moduli, molto più semplici, che ereditassero il modulo citato precedentemente a aggiungessero le funzionalità minime necessarie per poter convertire un solo tipo di formato dati in un dizionario python, delegando poi tutto il lavoro di importazione al modulo principale.

3.3 Descrizione implementativa

Passando ora in una sezione molto più tecnica, in cui descriverò nei dettagli la struttura e il codice del mio progetto, occorre specificare con precisione la nomenclatura data ai moduli sviluppati. Come già anticipato, il modulo principale verrà citato con il nome `synchronizer-main`, mentre gli altri moduli specifici per ogni formato dati verranno chiamati `synchronizer-_nome_del_formato_` (esempio: il modulo che importa dati Json si chiamerà `synchronizer-json`). A seguito illustrerò il modulo `synchronizer-main` e `synchronizer-json`. I moduli `synchronizer-xml` e `synchronizer-csv` non verranno descritti poichè analoghi a `synchronizer-json`.

3.3.1 Descrizione del File-System di synchronizer-main



Come si può facilmente vedere il mio modulo `synchronizer`, strutturato secondo i principi del paradigma MVC, presenta due classi di modelli `synchronizer` e `synchronizer_record` e tre viste di cui due per i rispettivi modelli e una per inserire un elemento nel menù. Sono presenti anche un controller Python e un foglio di stile che analizzeremo a seguito.

Le due classi: `synchronizer` e `synchronizer_record`

La classe `synchronizer` si pone l'obiettivo di astrarre l'importazione di un flusso di dati, più precisamente cerca di rappresentare un connettore tra la sorgente dati esterna e il database di Odoo. La classe `synchronizer_record` funge da supporto. Vediamo in che modo.

Un’istanza di synchronizer possiamo considerarla come un sincronizzatore tra due risorse. Il synchronizer può importare una risorsa esterna a Odoo e inserirla nel database oppure grazie agli eventuali cron, possiamo far fluire flussi di dati periodici per degli aggiornamenti. Purtroppo non possiamo sapere al momento dell’importazione se l’oggetto che si sta importando sia già stato importato, e quindi occorra aggiornare un record sul database, oppure sia la prima volta che viene importato, e quindi creare un nuovo record.

In seconda analisi, se anche sapessimo che l’oggetto è già stato importato, come possiamo trovare l’ID dell’oggetto da aggiornare nel database senza dover effettuare una scansione lineare estremamente inefficiente di tutti i record sul database? Sulla base di queste esigenze abbiamo creato la classe di supporto *synchronizer_record*.

Intraprendiamo la spiegazione con un approccio bottom-up. Partiamo dalla classe *synchronizer_record*.

```
class SynchronizerRecord(models.Model):
    _name = 'synchronizer_record'
    _sql_constraints = [
        ('field_unique',
         'unique(odoo_model, ext_id)',
         'Impossible to import two elements
         of the same model with the same ID!')
    ]
    synchronizer_id = fields.Many2one('synchronizer',
                                       string="Synchronizer ID", required=True)
    odoo_id = fields.Integer('Id of the imported record')
    odoo_model = fields.Char('Model of the imported record')
    ext_id = fields.Integer('Id of the external record')
    creation_date = fields.Datetime(string="Creation date",
                                      select=True)
    sync_date = fields.Datetime(string="Sync date", select=True)
```

Analizziamo il codice.

```
_name = 'synchronizer_record'
```

_name è l’attributo che indica il nome del modello come avevo già illustrato nel capitolo 2.

```
_sql_constraints = [
    ('field_unique',
     'unique(odoo_model, ext_id)',
```

```
'Impossible to import two elements
of the same model with the same ID!')
]
```

Questo particolare attributo impone un vincolo a tutte le istanze di questa classe: la coppia di attributi `odoo_model` e `ext_id` dev'essere una chiave del modello.

```
synchronizer_id = fields.Many2one('synchronizer',
    string="Synchronizer ID", required=True)
```

È il riferimento al modello synchronizer. Dato che possono esserci più oggetti importati con una singola importazione è sufficiente creare una singola istanza di synchronizer da associare a tanti oggetti synchronizer_record quanti gli oggetti in questione nell'importazione. In questo modo ci sarà un'istanza di synchronizer_record come riferimento per ogni oggetto da importare.

```
odoo_id = fields.Integer('Id of the imported record')
```

È l'ID dell'oggetto nel database di Odoo. Se un oggetto è già stato importato e occorre aggiornare qualche suo campo abbiamo la chiave per compiere un solo accesso in scrittura al database ed effettuare l'operazione efficientemente.

```
odoo_model = fields.Char('Model of the imported record')
```

Memorizza il modello a cui corrisponde l'oggetto importato.

```
ext_id = fields.Integer('Id of the external record')
```

ID dell'oggetto nel database della sorgente dati esterna.

```
creation_date = fields.Datetime(string="Creation date",
    select=True)
sync_date = fields.Datetime(string="Sync date", select=True)
```

Infine questi record memorizzano le rispettive date di creazione e ultimo aggiornamento del synchronizer_record.

Vediamo ora una parte di codice relativa alla classe `synchronizer` e cerchiamo di capire come questa classe collabora con la precedente.

```

class Synchronizer(models.Model):
    _name = 'synchronizer'
    name = fields.Char('Name', required=True)

    src = fields.Char('URL/Source', required=True)
    src_params = fields.Char('Params')
    source_type = fields.Selection([], string='Source type',
        required=False)

    basic_auth_username = fields.Char('Username')
    basic_auth_password = fields.Char('Password')

    cron_id = fields.Many2one(string='Odoo Cron',
        comodel_name='ir.cron', readonly=True)
    record_properties = fields.Text('Properties')
    model_hierarchy = fields.Text('Hierarchy')
    optional_header = fields.Text('Optional Header')

    # methods
    def create(self, context=None): [...]
    def unlink(self): [...]
    def nuke(self): [...]
    def sync(self, sync_id=None): [...]
    def _validate_url(self, sync_id): [...]
    def _sync_resource(self, resource, sync): [...]

```

Incominciamo con l'illustrare gli attributi di questa classe evitando quelli comuni agli esempi presentati precedentemente.

- `src` è un campo di testo in cui specificare l'URL remoto della risorsa o un path da cui caricala
- `source_type` specifica il formato degli oggetti che si sta importando. Questo attributo è di tipo Selection e prevede l'elenco dei possibili formati selezionabili. Questi tuttavia non vengono specificati; saranno i moduli figli di synchronizer che erediteranno questo campo e inseriranno la possibilità di scelta per il loro formato al momento dell'installazione.
- `src_params` funge da supporto per aggiungere parametri opzionali da inserire nella richiesta della risorsa.

- Successivamente notiamo due campi Char per inserire le eventuali credenziali per una Basic Authentication.
- A seguito un attributo di tipo relazionale per inserire un riferimento a un possibile cron. È merito di un oggetto di questi tipi se l'importazione potrà avvenire ciclicamente in automazione.
- Infine, è doveroso spendere maggiore attenzione alla funzionalità di `model_hierarchy`. Supponiamo di dover importare l'oggetto I di una possibile classe, esempio Impiegato. Questo oggetto I ha un attributo relazionale con riferimento A, istanza della classe Azienda, per specificare per quale azienda A quell'impiegato I lavora. Se volessimo importare con il modulo synchronizer, l'oggetto I ma nel database di Odoo non esiste alcun oggetto A occorre procedere nel seguente verso:
 - Creare l'oggetto A
 - Creare l'oggetto I
 - Inserire il riferimento ad A in I

E se volessimo eliminare dal database l'oggetto A ? In quel caso occorrerà procedere a ritroso rispetto al verso della creazione e modificare o eliminare tutti i riferimenti pendenti sugli oggetti in relazione che da quel momento in poi creerebbero inconsistenza.

Concentriamoci ora, invece, sui metodi principali di synchronizer di cui ho solo precedentemente riportato la dichiarazione:

- `create` è il costruttore della classe.
- `unlink` è il distruttore ovvero elimina l'oggetto synchronizer dal database Odoo.
- `nuke` è un metodo molto potente. Distrugge l'oggetto synchronizer, elimina tutti gli oggetti importati in quella sincronizzazione e gli eventuali cron associati.
- `sync` ingloba tutta la logica dell'importazione. È la funzione che viene collegata al bottone dell'interfaccia.
- `_validate_url` è una funzione che viene ereditata e chiamata dai moduli figli di synchronizer come vedremo in seguito. Riesce a caricare la risorsa dalla sorgente dati remota o dal file system locale.
- `_sync_resource` viene anch'essa chiamata solo dai moduli figli di synchronizer come la precedente. Inserisce i dati di un dizionario python strutturato in un determinato modo nel database Odoo.

Synchronizer - Viste

Le viste di questo modello le ho sviluppate in tre file XML distinti.

Partiamo analizzando il primo, *synchronizer.xml*. In questo file ci sono tre sezioni fondamentali: la definizione della vista di tipo form di synchronizer, quella di tipo tree e l'inserimento di un foglio di stile custom. Dato che abbiamo già illustrato come si sviluppano viste come le prime due, analizziamo solamente l'inserimento del foglio di stile:

```
<template id="assets_backend" name="synchronizer_assets"
    inherit_id="web.assets_backend">
    <xpath expr=". " position="inside">
        <link rel="stylesheet" href="/synchronizer-main/static/"
            "/src/css/synchronizer.css"/>
    </xpath>
</template>
```

Grazie alle linee di codice soprastanti, ora Odoo dispone delle informazioni sufficienti a caricare il nostro foglio di stile seguendo il path contenuto nel tag link. Il framework però non sa ancora come utilizzare le informazioni del file synchronizer.css, quindi all'interno di una vista (in questo caso nella form) si associa una classe presente nel foglio di stile (sync-container) a uno dei tag xml:

```
<div name="example" class="sync-container"> </div>
```

Il contenuto di questo tag rimane per il momento vuoto, verrà modificato dai moduli figli di synchronizer come avremo modo di vedere fra qualche sezione.

Nel secondo file, *synchronizer_record.xml* ho implementato delle semplici viste form e tree, banali, che non mostrerò in questo documento.

Infine in *synchronizer_menu.xml* ho implementato le voci di menù:

```
<menuitem id="synchronizer_menu" name="Synchronizer"
    parent="base.menu_config" sequence="30" />

<menuitem id="synchronizer_params_menu" name="Sync_Settings"
    parent="synchronizer_menu" sequence="25"
    action="synchronizer_action" />
<menuitem id="synchronizer_record_menu" name="Sync_Record"
    parent="synchronizer_menu" sequence="30"
    action="synchronizer_record_action" />
```

e le azioni corrispondenti per visualizzare le viste precedenti:

```

<record id="synchronizer_action" model="ir.actions.act_window">
    <field name="name">Synchronizer</field>
    <field name="res_model">synchronizer</field>
    <field name="view_mode">tree,form</field>
    <field name="view_id" ref="synchronizer_tree"/>
    <field name="help" type="html">
        <p class="oe_view_nocontent_create">
            Click to add a Sync.
        </p>
    </field>
</record>

<record id="synchronizer_record_action" model="ir.actions.act_window">
    <field name="name">Synchronizer Record</field>
    <field name="res_model">synchronizer_record</field>
    <field name="view_mode">tree,form</field>
    <field name="view_id" ref="synchronizer_record_tree"/>
    <field name="help" type="html">
        <p class="oe_view_nocontent_create">
        </p>
    </field>
</record>

```

Synchronizer - Controller

Come accennavo all'inizio della sezione 3.3.1 per questo modulo ho sviluppato anche un controller per testare con facilità il funzionamento durante fase di sviluppo. Ovviamente questo controller è doveroso rimuoverlo quando il modulo va in produzione ma dato che può essere utili ai fini di questa tesi lo presenterò. A seguito una parte di codice della classe Python *API_controller.py*:

```

class MyController(http.Controller):

    @http.route('/test_generator', type="http", auth="public")
    def my_test_generator(self, model, data_format,
                          last_id=None, last_date=None):
        _logger.info("calling test generator")
        if model == "res.partner":
            return self._gen_res_partner(data_format)
        elif model == "res.partner.bank":
            return self._gen_res_partner_bank(data_format)

```

```

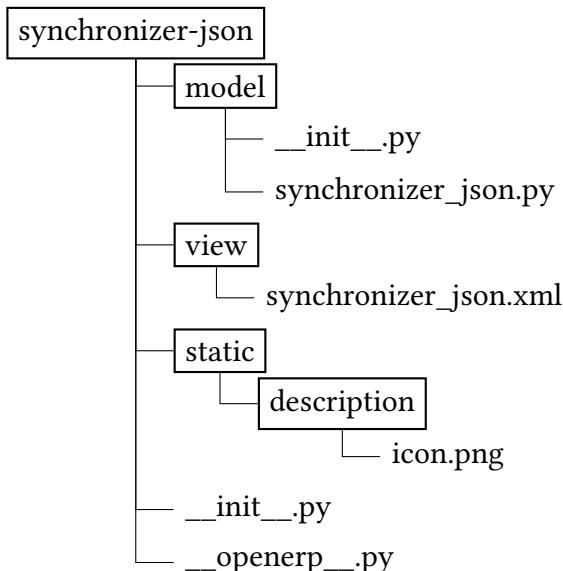
elif model == "res.bank":
    return self._gen_res_bank(data_format)
else:
    _logger.warning("Other models not supported yet")
    return {}

```

Sopra la definizione di `my_test_generator` troviamo un decorator grazie al quale possiamo specificare la rotta o il link per il quale il controller deve attivarsi e invocare la suddetta funzione. Ci si aspetta che la chiamata http abbia dei parametri, dei quali `model` e `data_format` obbligatori, che Odoo passa direttamente alla funzione. Successivamente il controller, in base a quale parametro viene passato in `model`, restituisce il risultato di una funzione che crea un oggetto nel formato contenuto in `data_format` strutturato in un preciso modo ma con dei dati casuali. Il valore di ritorno di questa funzione è la risorsa che viene inviata nella risposta http all'Odoo.

A cosa serve esattamente questo controller? Serve a simulare un possibile server remoto che risponde a una chiamata http con una risorsa. In questo modo mi è stato possibile testare la logica del mio modulo senza avere a disposizione un server remoto che lo facesse veramente.

3.3.2 Descrizione del File-System di synchronizer-json



Passiamo ora ad analizzare il modulo `synchronizer-json`.

Il modulo `synchronizer-json` presenta solo una classe Python e una vista in xml. L'idea fondamentale di questo gruppo di moduli è quella di aver spostato nel modulo gentitore la maggior parte della logica di programmazione e poter ridurre così il loro

compito. Come cita il principio di buona programmazione “*K.I.S.S. Keep It Sweet & Simple!*”

Vediamo rapidamente le parti fondamentali di codice di questo modulo.

synchronizer_record - model

La classe Python deve:

- ereditare la classe *synchronizer* del modulo gentitore
- aggiungere all’attributo di tipo *selection source_type* il formato dati per cui con questo modulo sarà possibile importare dati. In questo caso specifico dati Json.

```
class Synchronizer_json(models.Model):
    _inherit = ['synchronizer']
    source_type = fields.Selection(selection_add=[('json', 'JSON')])
```

Successivamente la funzione di importazione dovrà:

- chiamare la funzione corretta definita in *synchronizer* per ottenere la risorsa
- convertire la risorsa dal formato per cui questo modulo è stato implementato a un dizionario python
- importare o aggiornare il corrispondente record della risorsa in database

```
def sync_json(self, sync_id=None):
    _logger.info("inherit sync function: JSON")
    sync, result = super(Synchronizer_json, self).validate_url(sync_id)
    resource = json.load(result)
    return super(Synchronizer_json, self).sync_resource(resource, sync)
```

Ciò che occorre implementare di modulo in modulo è una funzione che converte l’oggetto che si sta importando dal formato dati per cui è implementato il modulo in un dizionario python. In questo caso il modulo python *json* prevede già *json.load()* che svolge questo ruolo.

Rimane ancora da chiarire un’ultima questione: il modulo come si aspetta che vengano strutturati i dati nella risorsa? In particolare, come fa a conoscere il nome del modello a cui corrispondono?

La soluzione sta nel strutturare con criterio i dati.

Ogni risorsa deve contenere, per essere considerata valida, un attributo `model` con il nome del modello a cui corrisponde e uno `data` contenente la lista di oggetti che si vogliono importare di quel modello, rappresentati con un elenco chiave-attributo, a seconda della libertà fornita dal formato dati. Per arrivare a questa struttura può essere utile all'utente del mio modulo sapere questa informazione e magari vederne una rappresentazione di esempio. Le viste dei moduli figli di synchronizer servono a questo scopo. Vediamo come.

synchronizer_json - views

```
<record id="synchronizer_json_form_inherit" model="ir.ui.view">
<field name="name">Synchronizer_json.form.inherit</field>
<field name="model">synchronizer</field>
<field name="inherit_id" ref="synchronizer-main.synchronizer_form"/>
<field name="arch" type="xml">
<xpath expr="//div[@name='example']" position="inside">
<div>
<br />
<h2>Format JSON data example</h2>
<br />
{
  "model": "res.partner",<br />
  "data": [{<br />
    "id": "123",<br />
    "name": "Partner_A",<br />
    "invoice_warn": "no-message",<br />
    "notify_email": "always",<br />
    "own_lot_counting": "1",<br />
    "res_partner_bank_ids": {<br />
      "model": "res.partner.bank",<br />
      "data": [{<br />
        "id": "456",<br />
        "acc_number": "789",<br />
        "state": "bank"<br />
      }]}<br />
    }]<br />
  --}<br />
}
<br />
</div>
</xpath>
```

```
</ field >
</ record >
```

All'interno del tag **record** viene ereditata la vista *synchronizer_form*, modificato il tag **div** con nome *example* e inserito al suo interno l'intero tag **div** che troviamo in questo file annidato in *xpath*.

Questa grande stringa verrà visualizzata ai piedi del form così da aiutare l'utente che sta creando un synchronizer a verificare che i dati da importare siano ben strutturati. Riferimento alla figura 3.1

Qualora i dati non rispettassero la struttura prestabilita, l'importazione fallisce riportando un messaggio di errore specifico.

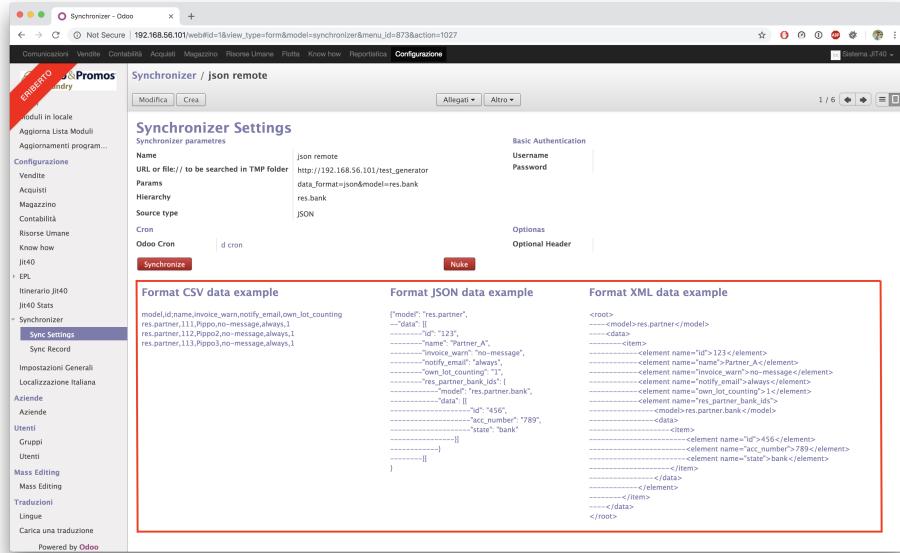


FIGURA 3.1: L'effetto delle viste dei moduli figli sulla vista gentitore

3.4 Esempi e screenshot

A seguito uno screenshot della schermata di presentazione per ognuno dei quattro moduli da me sviluppati:

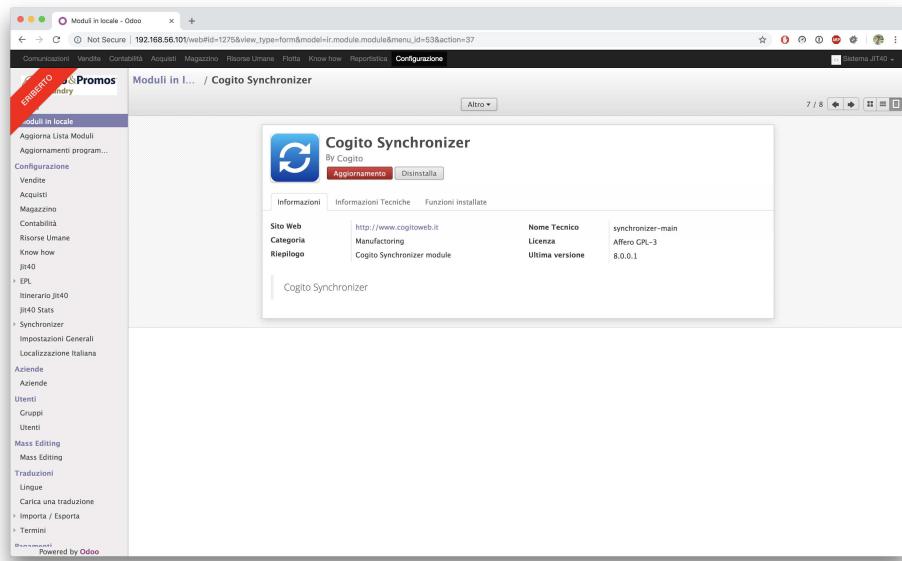


FIGURA 3.2: Modulo synchronizer

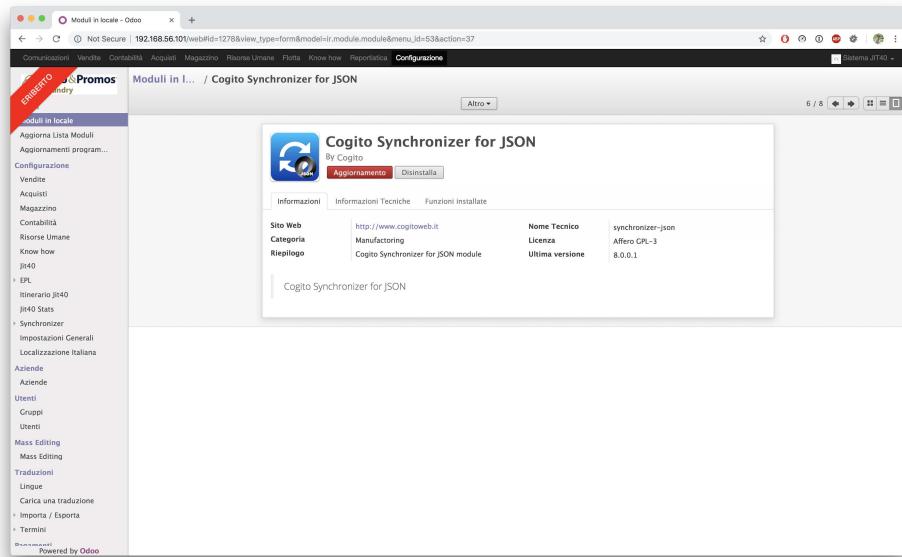


FIGURA 3.3: Modulo synchronizer-json

3.4. Esempi e screenshot

35

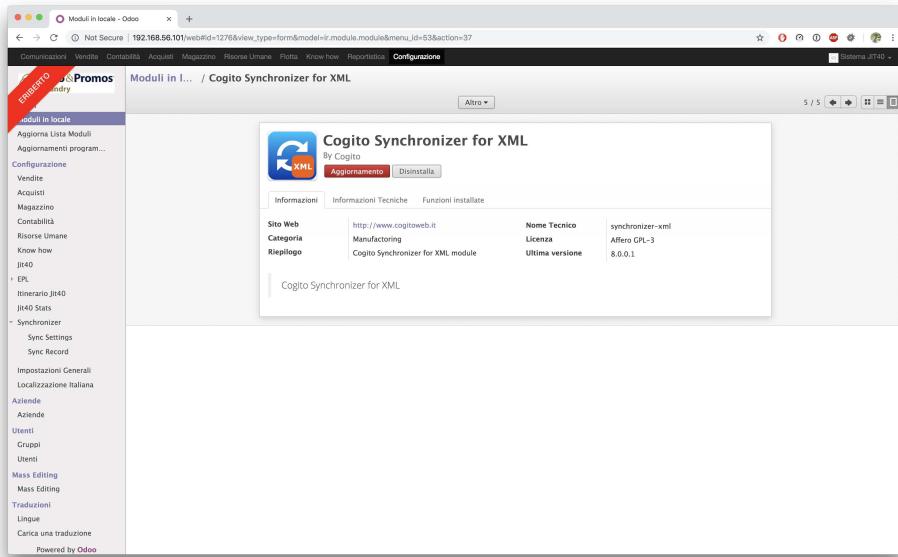


FIGURA 3.4: Modulo synchronizer-xml

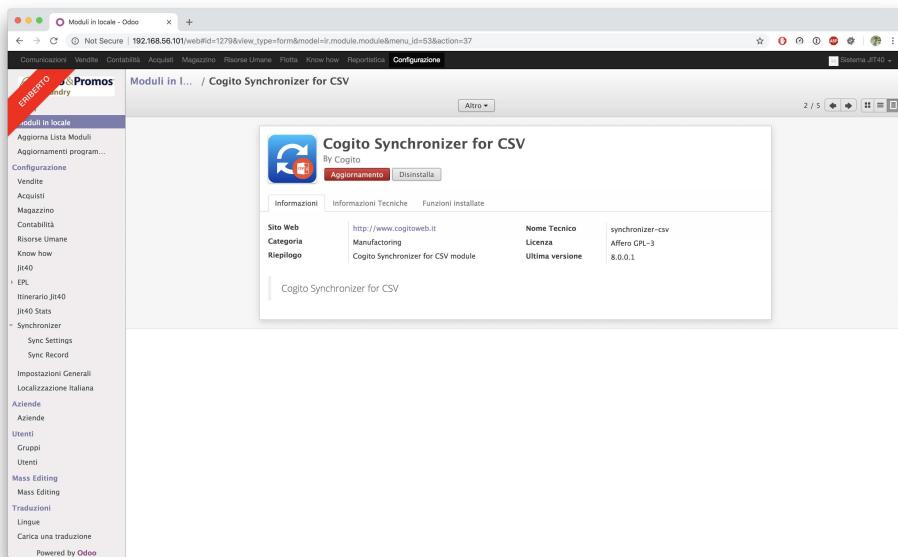


FIGURA 3.5: Modulo synchronizer-csv

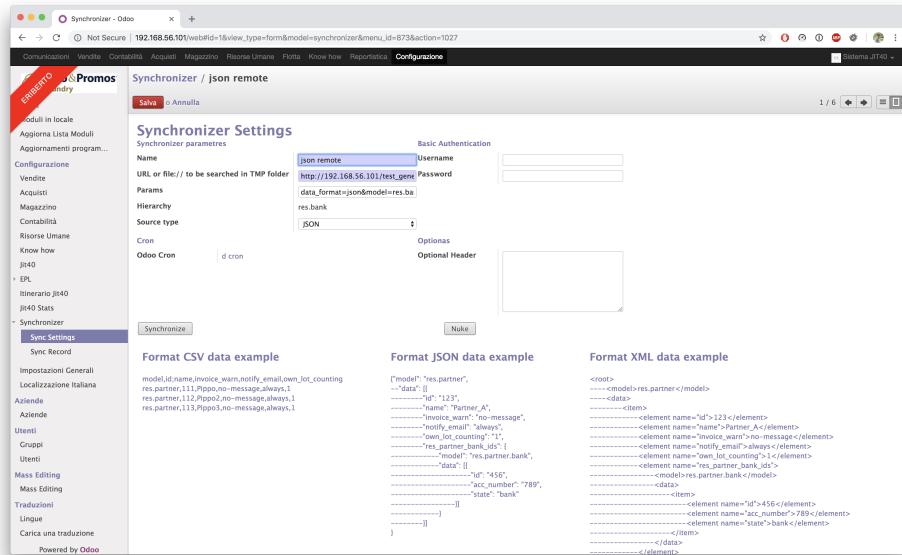


FIGURA 3.6: Vista form del modulo synchronizer

Synchronizer - Odoo							
Synchronizer							
Crea o Importa							
Moduli in locale		Name	URL/Source	Params	Username	Password	Source type
Aggiorna Lista Moduli		json remote	http://192.168.56.101/test_generator	data_format=json&model=res.bank			JSON
Aggiornamenti program...		xml remote	http://192.168.56.101/test_generator	data_format=xml&model=res.bank			XML
Configurazione		xml local	file://odoo_data/custom_example.xml				xml local cron
Vendita		csv local	file://odoo_data/easy_example.csv				CSV local cron
Acquisti		csv remote	http://192.168.56.101/test_generator	data_format=csv&model=res.partner			CSV remote cron
Magazzino		json local	file://odoo_data/custom_example.json				JSON local cron
Contabilità							
Risorse Umane							
Know how							
JIT40							
EPL							
Itinerario JIT40							
JIT40 Stats							
Synchronizer							
Sync Settings							
Sync Record							
Impostazioni Generali							
Localizzazione Italiana							
Aziende							
Aziende							
Utenti							
Gruppi							
Utenti							
Mass Editing							
Mass Editing							
Traduzioni							
Lingue							
Carica una traduzione							
Powered by Odoo							

FIGURA 3.7: Vista tree del modulo synchronizer

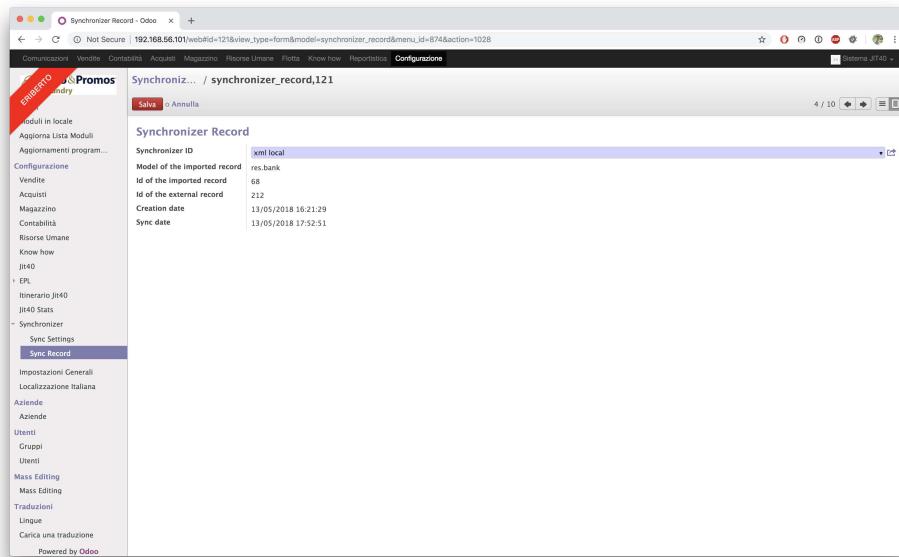


FIGURA 3.8: Vista form del modulo synchornizer-json



FIGURA 3.9: Visualizzazioni delle viste di menu

Conclusione

Sintesi

In questa progetto di tesi ho sviluppato un modulo per il gestionale Odoo che importasse dati da un risorsa remota o da un file in locale. Inizialmente il modulo si aspettava di gestire oggetti Json formattati in una struttura ben predefinita. Il lavoro di ampliamento e miglioramento fatto a posteriori riguardava lo scorporamento del modulo sviluppato in un primo modulo principale che racchiudesse tutta la logica di importazione dati e una serie di altri moduli specifici ognuno per un diverso formato dati che sfruttassero la logica del modulo principale.

Limiti e cose non fatte

Il mio progetto presenta tuttavia ancora dei limiti.

- In primo luogo la formattazione dei dati. Per quanto risulti ora facile, importare dati in un qualsiasi formato, si richiede sempre che gli stessi vengano strutturati all'interno di due campi fondamentali: model e data.
- Può essere utile sviluppare un modulo figlio di synchronizer che importi dati direttamente da un database, fornendo nella form i campi necessari ad interfacciarsi con i più comuni DBMS.
- Il linguaggio XML fornisce una descrizione formale della grammatica di un linguaggio chiamata DTD (Document Type Definition). Questo strumento sarebbe stato molto utile da usare nella vista di synchronizer-xml per dare all'utente non un esempio ma delle vere e proprio regole grammaticali su come strutturare i dati da importare. Purtroppo non mi è stato possibile trovare niente di simile per il caso dell'importazione del CSV e quindi, per completezza, ho inserito degli esempi di dati in tutti e tre i moduli figli di synchronizer.

Bibliografia

- Daniel Reis - Odoo Development Essentials (2015) - Packt Publishing
- Greg Moss - Working With Odoo (2015) - Packt Publishing
- Fabio Missio - Sviluppo di moduli in ambiente Odoo (2017)

Sitografia

- <https://www.odoo.com/>
- <https://en.wikipedia.org/wiki/Odoo>
- <https://www.cybrosys.com/blog/what-is-odoo-openerp>
- <https://www.sodexis.com/services/what-is-odoo-longwood-orlando-florida.html>
- <https://www.quora.com/What-is-Odoo>
- <https://www.process.st/checklist/what-is-odoo-3/>
- <http://mining2007.awardspace.com/alessandro/Tran.htm>