Applicazioni database con SQLAlchemy e QT4

Il progetto PyPaPi

Alberto “azazel” Berti

PyCon Due, 11 maggio 2008

Outline

Protocollo? Procedimenti

- PyPaPi sta per **Py**thon **P**ubblica **a**mministrazione **P**rotocollo **i**nformatico.
- Il nome non l'ho inventato io :-).
- Tiziano Lattisi ha sviluppato l'applicazione protocollo in Python/GTK+.
- L'applicazione protocollo ricalca 1:1 la precedente applicazione Gupta/Centura.

continua...

- É in funzione al comune di Riva del Garda e in alcune altre istituzioni.
- Il comune di Riva del Garda sta migrando la sua base dati da MS-SQLServer a PostgreSQL.
- Il comune intende rinnovare la suite di applicazioni in uso.
- Visti questi presupposti, è nato PyPaPi/Procedimenti.

Outline

L'applicazione Procedimenti

- Gestione iter procedurale.
- Persistenza dei dati su database PostgreSQL.
- Sostituzione dell'esistente applicazione sviluppata in Gupta/Centura.
- GUI standard (no HTML, AJAX, JavaScript).

Outline

La situazione

- Ci sono molti sviluppatori in grado di mettere insieme una form.
- Pochi che siano in grado di creare da zero un *binding* tra form e dati.
- Ancora meno che sappiano progettare una *buona* base dati (*efficace*, *coerente*, *non ridondante* , *espressiva*).
- Queste competenze sono spesso distribuite tra varie *figure*. A volte no, dipende dalla grandezza del gruppo.

Il problema

- Consentire anche a sviluppatori discretamente esperti di produrre forms *data bound*.
- Mantenere separata la *business logic* dall'applicazione GUI in se.
- Valorizzazione di ogni singolo passo o grado di sviluppo, perché diventi una *risorsa*, non un *incombenza!*

- Riutilizzo della logica in altre applicazioni e altri ambienti (front-end al cittadino via web?).
- maggiore testabilità.

Il problema

- Consentire anche a sviluppatori discretamente esperti di produrre forms *data bound*.
- Mantenere separata la *business logic* dall'applicazione GUI in se.
- Valorizzazione di ogni singolo passo o grado di sviluppo, perché diventi una *risorsa*, non un *incombenza!*

- Riutilizzo della logica in altre applicazioni e altri ambienti (front-end al cittadino via web?).
- maggiore testabilità.

La risposta?!

- Utilizzo delle moderne librerie di accesso ai dati (SQLAlchemy).
- Componentizzazione a "buccia di cipolla" .
- Foundation di librerie per lo sviluppo di applicazioni data-bound.
- Strumenti e *best practices* che valorizzino la struttura dati.
- Riutilizzo degli strumenti già disponibili.

Quali sono queste figure?

- Il modellatore della base dati.
- Il programmatore che implementa la business logic.
- Lo sviluppatore che crea le forms per la gestione dati.
- Come si riflette tutto ciò in PyPaPi?

Outline

La base dati

- É la *vera* risorsa.
- Occorre valorizzarlo.
- Non viene utilizzato SQLAlchemy per questo scopo.
- In PyPaPi sono stati introdotti una serie di tools che usano reST.

Esempio di codice reST

```
Procedimenti.AzioniIterProcedurale
----------------------------------

Alla fine della fiera, tutto questo ambaradan si riduce a questo, una
raccolta di azioni__, che devono prima o poi essere *svolte*, in ordine
più o meno arbitrario.

Durante il `setup dell'iter procedurale`, piuttosto che durante la
vita stessa della pratica, vengono inserite qui delle azioni,
generalmente durante l'esecuzione__ di un'altra azione, legata od un
vincolo, o...

Nota che queste azioni sono legate a una particolare fase__, quindi vi
possono essere sia azione legate a un generico `iter procedurale`__
sia a quello specifico per una certa pratica__.

__ Anagrafiche.Azioni_
__ `Esecuzione delle azioni`_
__ Procedimenti.FasiIterProcedurale_
__ Procedimenti.IterProcedurali_
__ Anagrafiche.Pratiche_
```

```
.. argomento:: Informazione sui singoli campi

   IDAzioneIterProcedurale
     Identificativo di questa azione su un certo iter

   IDFaseIterProcedurale
     Identificativo della fase dell'iter procedurale a cui è
     associata questa azione

   IDAzione
     Azione interessata da questo azione

   IstanteEsecuzione
     Ora di effettiva esecuzione di questa azione, NULL indica
     che non è ancora stata eseguita
```

```
.. script:: Generatore Procedimenti.AzioniIterProcedurale
   :language: sql
   :depends: Procedimenti

   create sequence Procedimenti.Gen_IDAzioneIterProcedurale

.. script:: Procedimenti.AzioniIterProcedurale
   :language: sql
   :depends: TimeStamped, Procedimenti

   create table Procedimenti.AzioniIterProcedurale
   (
    IDAzioneIterProcedurale largeid_t not null,
    IDFaseIterProcedurale largeid_t not null,
    IDAzione smallid_t,
    IstanteEsecuzione timestamp_t,

    constraint PK_AzioniIterProcedurale primary key (IDAzioneIterProcedurale)
   ) inherits(TimeStamped)
```

Gestione dello schema

- grazie ad una dichiarazione delle dipendenze, il file sql per la generazione del db viene generato in maniera corretta.
- viene prodotta una versione html da utilizzare come documentazione.
- genera un diagramma della struttura del db.

Outline

Dai dati all'utente

- SQLAlchemy fa un ottimo lavoro di mapping:
  - resta dietro le quinte pur consentendo di definire o portare a galla le relazioni tra i dati;
  - è così poco invasivo che integrare dati provenienti da altre sorgenti è fattibile.
- Man mano che il focus si sposta dai dati all'utente, si sposta anche la competenza dello sviluppatore.

Dai dati all'utente (continua...)

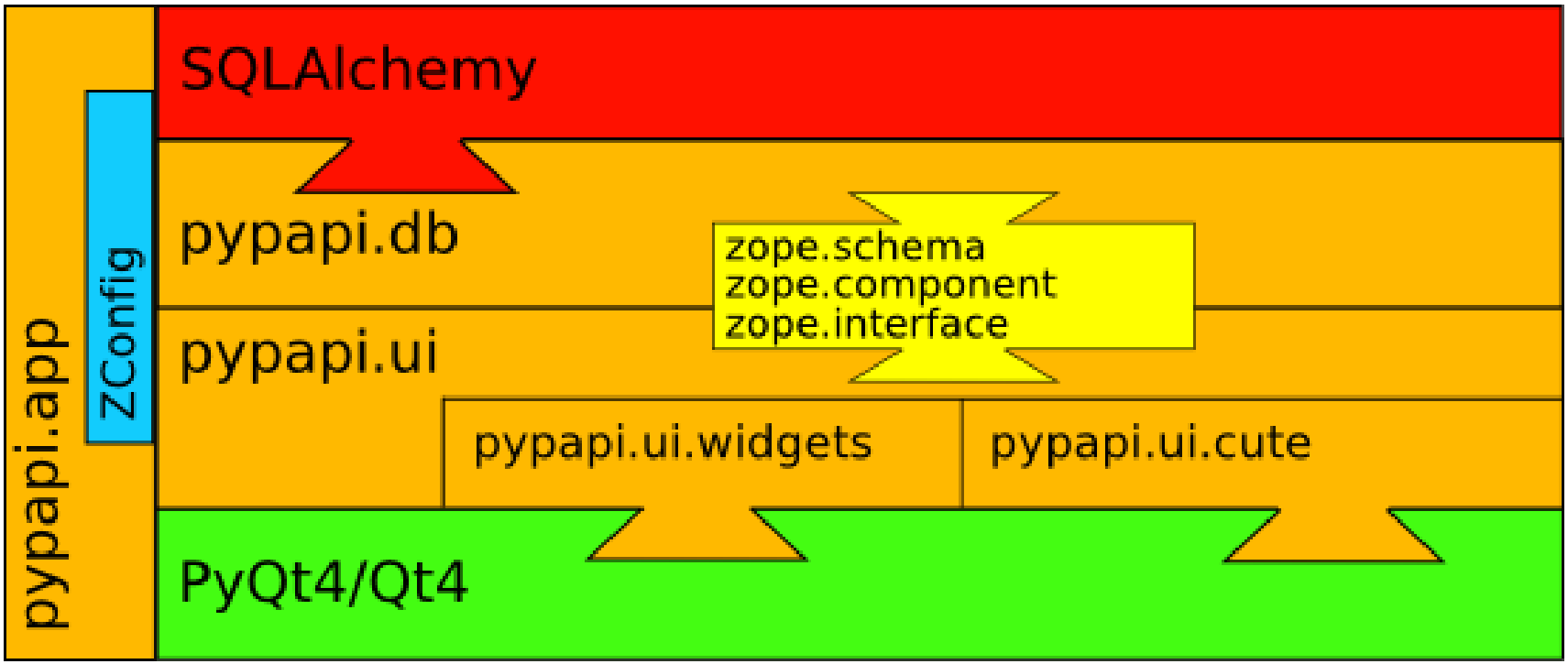
- Qt4 è "fresco" ma allo stesso tempo maturo:
  - i componenti MVD sono una nuova feature di questa major release;
  - grazie a KDE e a tutti gli altri progetti che lo usano, è pronto per essere usato (QDataWidgetMapper aggiunto in Qt 4.2);
  - dopo un iniziale scetticismo (C++), il feeling è ottimo;
  - è più coerente rispetto a GTK+ sotto molti aspetti: API, documentazione, strumenti;
  - ottimi bindings Python (anche se non c'è un repository pubblico);
  - In Qt 4.4 è stato integrato WebKit;

Dai dati all'utente (continua...)

- Qt Designer è assolumente superiore a Glade:
  - può realmente essere usato da persone che hanno una passata esperienza con dei RAD;
  - integrare e mettere a disposizione dei widgets sviluppati in Python è facile;

Outline

Struttura della libreria PyPaPi



pypapi.db

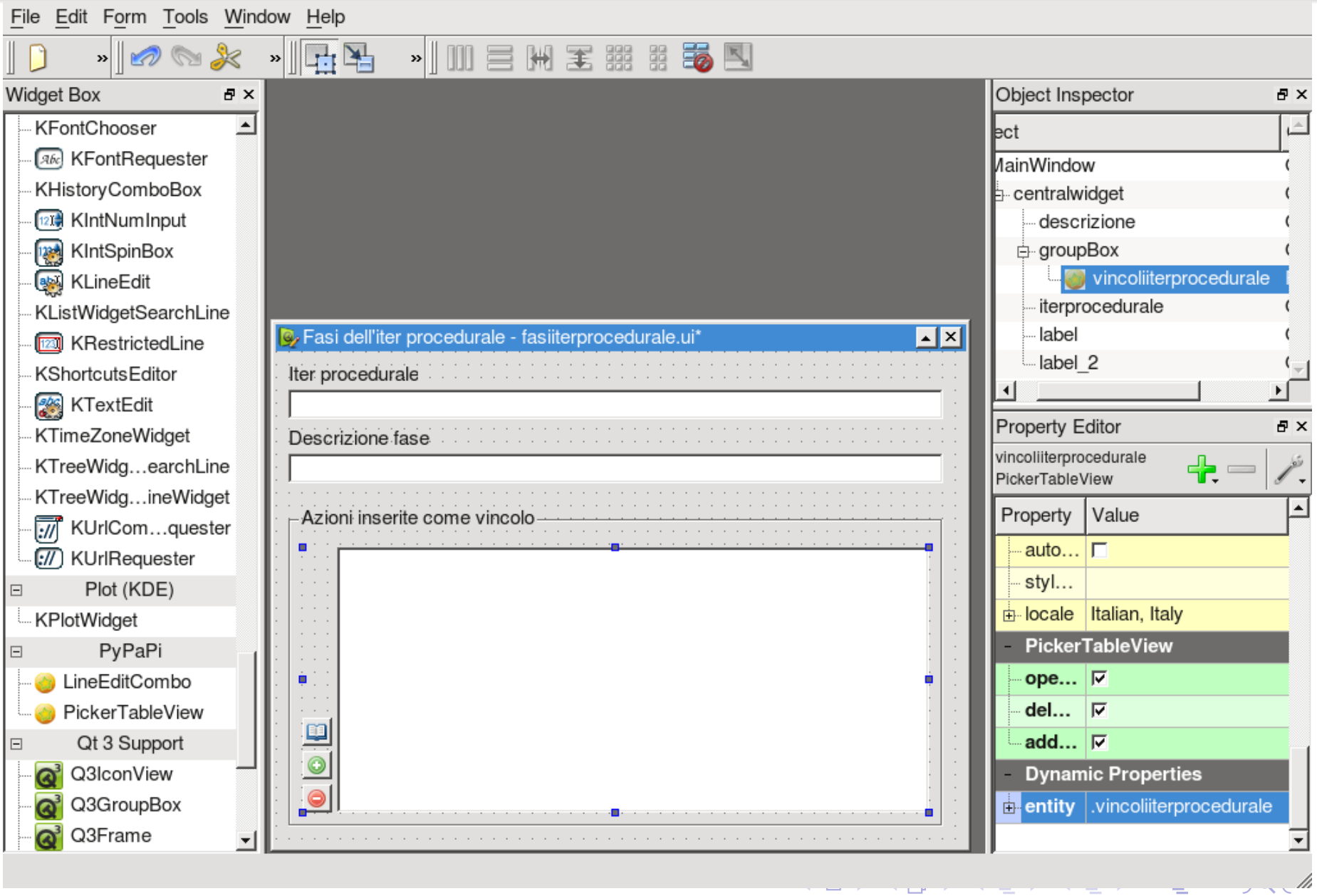
- Utilizza *zope.schema* per definire elementi accessori utili nell'interazione con l'utente:
  - definisce il *tipo* dei campi (singola riga, "textarea", di lookup, ecc..)
  - definisce eventuali validatori;
  - introduce il concetto di *sorgente*, utilizzato per specificare il *superset* di provenienza dei dati legati tramite relazioni 1-1, 1-n a basso livello e nelle *ricerche;*
  - lo *store* è il mediatore tra le operazioni di *aggiunta* e *rimozione* di istanze (records) e la persistenza (sessione SQLAlchemy).

pypapi.db (esempio)

```
class IFase(IEntitaBase):
    """
    Anagrafica delle fasi (o stati) che può assumere una generica
    pratica dall'apertura alla sua archiviazione. L'elenco delle fasi che
    assume una determinata categoria di pratiche è definito all'interno
    dell'iter procedurale (vedi fasi dell'iter procedurale).
    """

    idfase = zope.schema.Int(title=u"Identificativo",
                             description=u"Identificativo univoco della fase",)
    idfase.setTaggedValue('layout', COLLAPSE)

    ufficio = zope.schema.Choice(title=u"Ufficio responsabile",
                                 description=u"Se la fase è esclusiva, solo l'ufficio "
                                             u"specificato può "
                                             u"assegnare tale fase ad una pratica",
                                 source=DbSource())

    descrizione = zope.schema.TextLine(title=u"Descrizione",
                                       description=u"Descrizione",)
    descrizione.setTaggedValue('layout', EXPAND)

    esclusiva = zope.schema.Bool(title=u"Fase esclusiva",
                                 description=u"Solo l'ufficio specificato può assegnare "
                                             u"tale fase ad una pratica",)
```

pypapi.ui

- Decora form standard prodotte con Qt Designer con il binding ai dati
- Ogni form gestisce uno store di oggetti omogenei.
- Per ogni relazione specificata a livello .db è possibile definire visibilità e caratteristiche dei campi.
- Come *hint* per legare widget e dati viene utilizzato il "path" delle relazioni o il nome del campo (il widget visualiza un valore scalare o un intero dataset).
- Nel Designer il nome del campo può essere il nome del widget o una *Dynamic Property*. Il path è sempre reso con una D. P.
- Qualsiasi widget può diventare data bound.


File Edit Form Tools Window Help
Widget Box
KFontChooser
KFontRequester
KHistoryComboBox
KIntNumInput
KIntSpinBox
KLineEdit
KListWidgetSearchLine
KRestrictedLine
KShortcutsEditor
KTextEdit
KTimeZoneWidget
KTreeWidg…earchLine
KTreeWidg…ineWidget
KUrlCom…quester
KUrlRequester
Plot (KDE)
KPlotWidget
PyPaPi
LineEditCombo
PickerTableView
Qt 3 Support
Q3IconView
Q3GroupBox
Q3Frame
Fasi dell'iter procedurale - fasiiterprocedurale.ui*
Iter procedurale
Descrizione fase
Azioni inserite come vincolo
Object Inspector
MainWindow
centralwidget
descrizione
groupBox
vincoliiterprocedurale
iterprocedurale
label
label_2
Property Editor
vincoliiterprocedurale
PickerTableView
Property
Value
auto…
styl…
locale
Italian, Italy
PickerTableView
ope…
del…
add…
Dynamic Properties
entity
.vincoliiterprocedurale

pypapi.app

- Integra *ZConfig* per gestire la configurazione di forms, database, ecc..
- Implementa la main form e le altre form accessorie.
- "decora" le forms con oggetti generici (navigatore, MDI)

```
<application>
<eventlog>

   level debug

   <logfile>

     path        STDOUT

     format      %(asctime)s [%(levelname).1s %(name)s]: %(message)s

     dateformat  %H:%M:%S

   </logfile>
</eventlog>
<database>

   uri postgres://localhost/pypapi
</database>
<form FasiIterProcedurale>

     class pypapi.app.procedimenti.form.FasiIterProcedurale

     interface pypapi.db.interfaces.IFaseIterProcedurale

     uifile ui/fasiiterprocedurale.ui

     title Gestione fasi iter procedurali

     toplevel False

     <entity .>

       <column idfaseiterprocedurale />

       <column descrizione />

       <column iterprocedurale />

     </entity>

     <entity .vincoliiterprocedurale>

       <column idvincoloiterprocedurale />

       <column descrizione />

     </entity>
</form>
</application>
```

Riassunto

Quali sono i passi da compiere per sviluppare un'applicazione?

1. Creazione struttura del database.
2. Definizione oggetti *table*, *mapper* e *model* di SQLAlchemy.
3. Definizione interfacce zope.schema.
4. Creazione delle forms con Qt Designer.
5. Creazione delle classi delle form dove codificare gli *event handlers*.
6. Definizione del binding ai dati nel file di configurazione.

Stato

- La libreria quasi completa per quanto riguarda le features:
  - alcuni dettagli sono ancora grezzi;
  - da implementare una più fine gestione delle sessioni;
  - migliorare la funzionalità delle ricerce;
  - la sicurezza: per la maggior parte verrà delegata al database tramite l'utilizzo di *views*.
- L'applicazione è in via di completamento nel prossimo periodo:
  - la base dati sembra appropriata ma non tutti gli use cases sono stati verificati.
- Attualmente siamo in tre a contribuire allo sviluppo oltre a me Tiziano Lattisi ed Emanuele Gaifas.

Get in touch

- Il sito è un'istanza di Trac all'indirizzo
  `http://www.pypapi.org`
- Il repository è gestito con Darcs. Per scaricarlo:
  darcs –partial http://darcs.arstecnica.it/our/pypapi
- Esiste una mailing list su SF
  `http://sourceforge.net/projects/pypapi`
- Il mio indirizzo email: **alberto at metapensiero.it**