# IGSuite - Integrated Groupware Suite

Una delle mode del momento, è senza dubbio quella delle soluzioni web-based. Le aziende cercano costantemente di rimanere il più possibile libere da infrastrutture, piattaforme proprietarie, o problemi dovuti a logistica e accessibilità delle informazioni. Dal suo avvento il Web è stato senza dubbio la risposta ad una moltitudine di problemi, si può dire che è stato finalmente lo 'standard' a cui ogni produttore di software ha dovuto adeguarsi. Ma cosa succede quando un progetto web-based orientato al groupware aziendale è in sviluppo con un anticipo di almeno 5 anni rispetto alla moda attuale?

IGSuite o meglio IsoGest è nato nel '98 all'interno di un'attività lavorativa che presentava diverse problematiche di logistica e cooperazione, l'obiettivo da subito fu unico nel suo genere: 'trattare informazioni aziendali doveva essere tanto facile quanto navigare'. E fù questo l'orientamento dato allo sviluppo del progetto. Nel tempo IG è divenuto una vera e propria groupware suite, integrata da decine di strumenti utili al normale lavoro quotidiano.

Il nome IsoGest porta molto spesso ad essere frainteso da chi non conosce il progetto. E' vero deriva da 'Iso', come le Iso standard e 'Gest' come gestione, ed in effetti è nato come strumento di gestione di un sistema di qualità, ma nel tempo ha perso questa funzione primaria ereditando però alcune caratteristiche. In IG infatti, ogni cosa' è gestita da un punto di vista: dell'Emissione, Distribuzione, Archiviazione e Rintracciabilità; proprio come in genere si usa fare per documenti che riguardano un sistema di qualità. La particolarità sta nel fatto che per 'ogni cosa' si intende ad esempio: procedure di qualità, offerte, lettere, ordini, contratti ma anche documenti wiki, e-mail, fax ricevuti, fax inviati, eventi, todo. Ogni cosa in IG ha le stesse tipologie di proprietà e metodi di gestione.

E' chiaro che tutto questo non poteva essere realizzato se non attraverso lo sviluppo di un Framework ben definito. Il Framework di IG definisce uno pseudo-linguaggio con funzioni specifiche ad esempio per il rendering della GUI o per l'invio di query al DataBase o ancora per la creazione di widget utili alla gestione di informazioni in un ambiente web-based. Sviluppare il Framework all'interno di IG è stato come costruirsi gli strumenti necessari a raggiungere l'obiettivo. Ad oggi si può tranquillamente dire che il Framework è maturo al punto di poter essere utilizzato per la creazione di altre applicazioni web-based, tra breve infatti sarà documentato e reso disponibile come modulo Perl.

Nello sviluppo di IG non si è sempre e solo riflettuto sulle utilità che via via si andavano definendo, ma piuttosto si sono cercate soluzioni architetturali che rendessero il prodotto il più possibile flessibile e libero da qualsiasi componente a lui necessario. IG oggi gira perfettamente sia su piattaforma Linux che Windows e puo' essere utilizzato sia con Postgres che Mysql, inoltre c'è da aggiungere una localizzazione in Inglese Italiano e Spagnolo, ed un installer che in pochi click mette su l'intero sistema.
Decine le features del progetto, la maggior parte delle quali ben definite e funzionali, altre invece ancora in sviluppo o perfezionamento.

In termini usati nel gergo comune, si può asserire che IG integra strumenti rintracciabili in software definiti oggi da sigle quali: ERP, CRM, CMS, PIM. In realtà la definizione più vera anche se limitativa resta quella di Groupware, e in quanto groupware tutti gli strumenti che compongono la suite hanno come scopo comune quello di coordinare e avvicinare gli utenti del sistema al fine di aumentarne la collaborazione l'interazione e la condivisione delle risorse.

In questo articolo ci soffermeremo sul framework di IG al fine di poter creare una piccola applicazione d'esempio. Lasciamo al lettore la possibilità di appronfondire la conoscenza delle features presenti all'interno del progetto, applicazioni quali: IGFax; IGTodo; IGFilemanager IGWebmail; IGCalendar; IGWiki; e molte altre.

## Anatomia di un'applicazione che utilizza IG.pm

Dopo i primi mesi di sviluppo di IGSuite (IsoGest al tempo), ci si rese conto che tanto del lavoro fatto era ripetitivo e ridondante, avevamo bisogno di potenziare e "affilare" i nostri strumenti. Fu posto allora un obiettivo, ogni applicativo sviluppato, doveva potenzialmente essere uno strumento da poter riutilizzare all'interno della suite.

In questo modo nel tempo l'intero progetto si è mosso verso un unico grande obiettivo, la crescita di un framework flessibile, veloce e pratico da utilizzare.

In questo articolo analizzeremo una semplice applicazione web based che utilizza il framework di IGSuite per creare un form con lo scopo di effettuare l'upload di un file, verificarne il contenuto e successivamente o respingerlo qualora non sia in formato 'text', o salvarlo in una directory predefinita in caso contrario.

## IG.pm e le Dispatch Table

Il framework di IGSuite ruota tutto sul suo modulo principale IG.pm che da solo importa nell'applicazione: l'ambiente CGI, numerosi widget grafici per il rendering di form, task, tabelle ecc, piccole utilità per il coding dei dati e per la manipolazione di date e l'interfaccia ad un RDBMS predefinito. Molti metodi implementati sono il risultato di wrap di altri moduli Perl ben conosciuti come CGI.pm, HTML::Entities, DBI e molti altri.

La cosa che più deve destare attenzione nel primo pezzo di codice riportato, è la funzione DTable (Dispatch Table) l'argomento è affrontato in molta documentazione on-line, e non ultimo all'interno di un recente libro di Mark Jason Dominus "Higher Older Perl" quindi ci limiteremo a spiegare cosa fa il dispatcher di IGSuite.

DTable sfrutta semplicemente il valore del parametro cgi "action" per lanciare l'omonima procedura all'interno dello script. Nel caso action non sia definita, per default viene eseguita 'default_action()' procedura che deve sempre essere presente all'interno di un'applicazione di IGSuite. Nel nostro esempio 'default_action()' altro non farà che visualizzare il form per la richiesta del file di cui fare l'upload.

I parametri passati a DTable riguardano oltre all'action che abbiamo visto rappresentare il nome della relativa procedura; anche un valore uguale a '1' o '0' che abilita o disabilità la possibilità di eseguire la procedura specificata. In alternativa al valore binario si può indicare il riferimento ad una sub che dovrà restituire ovviamente gli stessi valori '1' o '0'.

E' grazie a questa particolarità di DTable che si possono scrivere apposite funzioni che individuano se avviare o meno una determinata azione dello script. Nel caso di IGSuite molto spesso questo compito è affidato ad apposite funzioni come CheckSecur() o CheckPrivilege() ma non ne parleremo in questo documento.

Nell'esempio richiameremo anche il modulo File::MMagic che utilizzeremo più avanti nell'applicazione.

```
#!/usr/bin/perl
use strict;
use IG;
use File::MMagic;

IG::DTable ( file_upload    => \&mychecks,
             default_action => 1 );

sub mychecks
 {
  1; ## .... my stuff....
 }
```

## L'interfaccia grafica e i Form

Il codice sotto riportato è abbastanza intuitivo e lascia comprendere il funzionamento degli widget richiamati per la definizione sia dell'interfaccia grafica che del form per l'upload del file.

Sono tantissimi gli attributi messi a disposizione da ogni widget di IGSuite, di questi ne analizzeremo a titolo dimostrativo soltanto alcuni riferiti al metodo "Input":

**type**

Il metodo "Input" comprende tutti quei campi che normalmente è possibile definire all'interno di un Form Html più molti altri derivanti da essi. Grazie all'attributo "type" si definisce quale tipo di campo vogliamo inserire nel form. Oltre ai campi standard ne esistono alcuni specifici, ne è un ottimo esempio il campo di tipo "date" che definisce un campo "text" ma con un controllo automatico del valore inserito, e un piccolo calendario che appare a richiesta dell'utente.

**pattern**

Sfruttando il fatto che JavaScript supporta le regular expression è possible grazie a questo attributo definire una RE da passare ad un JavaScript che effettuerà il matching lato client del valore inserito avvisando l'utente di eventuali difformità del tipo di dato inserito.

**quickhelp**

In modo molto pratico con questo attributo si definisce un riquadro a scomparsa che appare qualora l'utente si sposti con il mouse sulla "label" del campo del form. Attributo molto utile per informare l'utente sul tipo di dato che viene richiesto.

**float**

I form in IGSuite sono allineati esclusivamente grazie a CSS e non a tabelle html come di consueto si fa. Con l'attributo "float" si definisce in quale posizione visualizzare un determinato campo rispetto al suo contenuto. Sarà possibile ad esempio visualizzare più campi sulla stessa riga specificando un float uguale a 'left' (come nel nostro esempio) il che posizionerà il campo precedente alla sinistra del campo in oggetto oppure visualizzare un campo per ogni riga (valore di default) con un float nullo.

```
sub default_action
 {
  HtmlHead();
  TaskHead( width => 500,
            title => 'IG Framework Demo - Input Form');

  FormHead( cgiaction => 'file_convert',
            enctype   => 'multipart/form-data',
            method    => 'post');

  TaskMsg( Input( type      => 'text',
                  label     => 'New file name',
                  pattern   => '\w\.\w',
                  quickhelp => 'You can leave it blank',
                  name      => 'newfilename').

           Input( name      => 'myfile',
                  label     => 'Select your file',
                  type      => 'file' ).

           Input( type      => 'submit',
                  float     => 'left',
                  label     => 'Ok' )
         ,4);

  FormFoot();
```

```
 TaskFoot();
 HtmlFoot();
}
```

## File Upload

La procedura "file_upload" è un concentrato di attività svolte dal framework. Iniziamo subito dicendo che i parametri passati dai form html a IG.pm vengono gestiti tramite due hash '%on' e '%in' rispettivamente il primo contiene il valore passato tale e quale, il secondo per comodità in alcune applicazioni, riporta il valore 'quotato' e pronto per essere passato ad una query verso l'RDBMS. Qualcuno non condividerà tale scelta a vantaggio di metodi che restituiscono i valori dei parametri (tipo 'param()' utilizzato da CGI.pm) ma per motivi storici, ed anche per motivi a parer nostro di efficenza, IGSuite nel tempo ha continuato ad adottare tale sistema.

Il metodo che più desta interesse nel codice sotto riportato è 'FileUpload()' che offre un alto grado di flessibilità sulla gestione dell'upload di file da form html. Ecco spiegati tutti gli attributi riportati nell'esempio:

**param_name**
: E' semplicemente il nome del campo specificato nel form da cui prelevare il file. Chiaramente il campo dovrà essere di tipo 'file' e all'interno di FormHead() dovrà essere specificato l'attributo enctype con valore uguale a 'multipart/form-data'.

**target_dir**
: E' il nome della directory nella quale vogliamo salvare il file. Nell'esempio viene specificata $IG::temp_dir che è una directory temporanea messa a disposizione dal framework di IGSuite per ogni tipo di utilizzo. L'uso di tale directory è comoda quando non sappiamo su quale sistema operativo verrà eseguita la nostra applicazione.

**target_file**
: E' il nome che vogliamo dare al file. Nell'esempio è specificato il valore del campo 'newfilename', ma nel caso non venga specificato nessun valore FileUpload() assegnerà al file il nome originario.

**overwrite**
: E' un flag che permette di evitare la sovrascrittura del file qualora ne esista già una copia all'interno della directory specificata.

**deny_pattern**
: Gemello di allow_pattern è un filtro sul nome del file. E' possibile specificare una regular expression che dovrà "matchare" il nome del file.

**filter**
: Questo è forse l'attributo più interessante in quanto è possibile passare ad esso il riferimento ad una funzione che filtri il contenuto del file prima di scriverlo nella nostra directory. Nel nostro esempio dopo aver utilizzato File::MMagic per prelevare il contenttype del file, ritorniamo l'interno contenuto nel caso sia di tipo "text" oppure una stringa vuota (filtriamo quindi l'interno contenuto) nel caso contrario.

```
sub file_upload
 {
  default_action() && return if !$on{myfile};

  HtmlHead();
  TaskHead( width => 500,
            title => 'IG Framework Demo - Result page');
```

```
 TaskMsg( IG::FileUpload
            ( param_name   => 'myfile',
              target_dir   => $IG::temp_dir,
              target_file  => $on{newfilename},
              overwrite    => 'true',
              deny_pattern => '\.exe$',
              filter => sub { my $contents = shift;
                              my $ref = new File::MMagic;
                              my $content_type = $ref->checktype_contents($contents);
                              return $content_type =~ /text/ ? $contents : '';
                            }
            )
        );
 TaskFoot();
 HtmlFoot();
}
```

Per vedere lo script in esecuzione http://www.igsuite.org/cgi-bin/demo2

Come si è potuto vedere, grazie al framework di IGSuite si è arrivati a creare uno pseudo linguaggio che semplifica la vita di uno sviluppatore di applicazioni web based aumentandone la produttività.

E' doveroso in ogni modo aggiungere che il framework di IGSuite si trova ancora in uno stato di "dipendenza" dall'intero progetto e non è ancora possibile utilizzarlo in un contesto differente. Gli sforzi odierni comunque sono concentrati tanto sulla stabilità ed efficenza del framework, quanto ad un suo inevitabile distacco dalla suite.

LucaS