Workflow in Geoweb

```
Questa guida è da ritenersi valida per le versioni di Geoweb dalla 4.2.7
compresa in avanti.
```

Geoweb permette di costruire ed integrare processi nel framework grazie alla integrazione del motore di workflow esterno: Activiti.

Che cos'è Activiti?

https://www.activiti.org/ è una **piattaforma di workflow** e **Business Process Management (BPM)** leggera ed orientata a contesti business, sofware development e system administration. Mette a disposizione un veloce e scalabile **motore di processi BPMN 2** per Java. E' open-source e viene distribuita sotto licenza Apache. Activiti è leggera, basata su standard open e disegnata per ben integrarsi con applicazioni basate su Spring, come GeoWeb.

Che cos'è il BPMN 2?

BPMN 2.0 (Business Process Modeling Notation) è uno **standard di sviluppo open** sviluppato dallo Object Management Group (OMG) per fornire una notazione che sia facilmente comprensibile da tutti gli utenti business: business analyst che disegnano processi, sviluppatori che implementano la tecnologia per eseguire tali processi e, in generale, chiunque debba gestire e monitorare tali processi.

La prima versione delle specifiche BPMN definiva solo una **notazione grafica**, e divenne presto popolare presso l'utenza business analyst. Essa definiva alcuni concetti di base, quali **task utente**, **procedure eseguibili** e **decisioni automatiche** (le quali potevano essere visualizzate in modo standard e trasversale ai vari contesti proprietari).

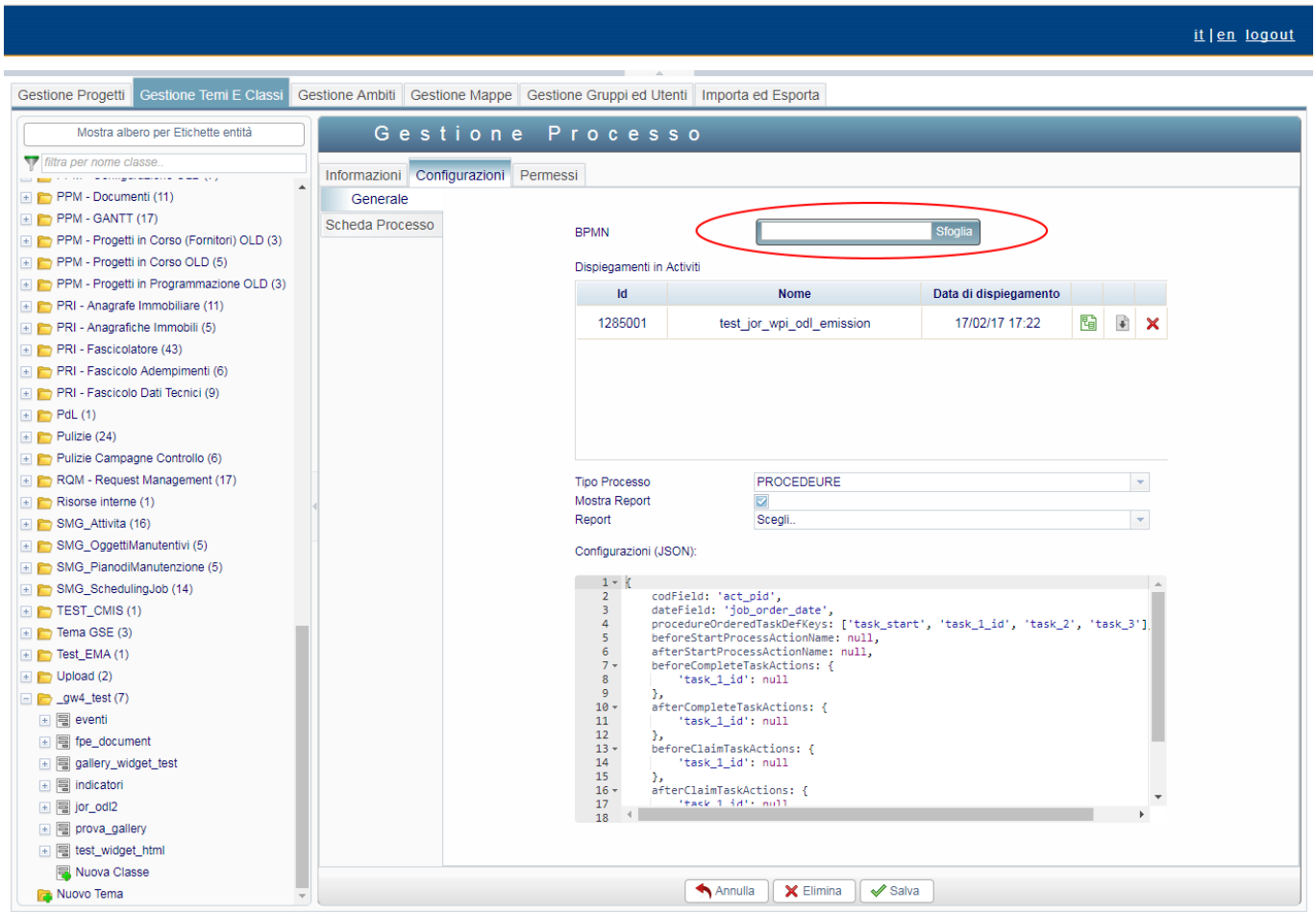
La seconda versione estendeva tale standard per includere **semantiche di esecuzione** e **formati di scambio comuni**. Questo significa che i modelli delle definizioni di processo BPMN 2.0 possono essere interscambiate fra vari editor grafici, ed **eseguiti su ogni motore di workflow aderente alle specifiche BPMN 2.0 come Activiti**. Puoi sapere di più riguardo allo standard BPMN sul sito dell'OMG.

Creare un file .bpmn

Geoweb necessita solo del file .bpmn (aderente allo standard BPMN 2.0) per sapere tutto il necessario riguardo alla definizione di un dato processo. Tale file ha una struttura xml e può essere generato con un qualsiasi editor grafico/testuale. Nel proseguo faremo riferimento all'editor Activiti Eclipse Designer, disponibile come plugin in Eclipse (istruzioni per l'installazione qui https://www.activiti.org/userguide/)

Caricare un file .bpmn

Tale file può essere caricato nel framework dall'apposita interfaccia Geoweb Admin, nell'apposita scheda 'Configurazioni' del menu accessibile da 'Gestione Temi e Classi' ⇒ [Tema] ⇒ [Classe] ⇒ 'Processi' ⇒ [Processo].



Nella scheda 'Configurazioni' ⇒ 'Generale' del menu accessibile da 'Gestione Temi e Classi' ⇒ [Tema] ⇒ [Classe] ⇒ 'Processi' ⇒ [Processo] è possibile caricare **più definizioni di processo** per **lo stesso GwProcess**, ognuna legata ad uno specifico file .bpmn.

Ad essere utilizzata in Geoweb per avviare una nuova istanza di processo è sempre **l'ultima definizione** di processo caricata. Caricare una nuova definizione però non comporta problemi ai processi già in corso: essi verranno terminati secondo il flusso di processo con il quale sono stati iniziati.

L'interfaccia per ogni dispiegamento di processo permette di:

- aprire una **rappresentazione grafica** del flusso di processo
- scaricare il file .bpmn
- eliminare il dispiegamento

Alla cancellazione di un dispiegamento, il sistema richiederà se l'utente vuole procedere anche alla contestuale cancellazione di tutte le istanze di processo avviate secondo quella definizione. Rispondendo 'no' le istanze di processo avviate con quella definizione di processo termineranno

normalmente, altrimenti verranno cancellate.

Eliminare l'ultimo dispiegamento caricato, comporta l'utilizzo del nuovo ultimo disponibile in lista, il quale verrà usato per avviare le nuove istanze di processo

Integrazione Activiti in Geoweb

Activiti è integrato in Geoweb tramite un apposito **plugin**. In questo è presente tutto il necessario per poter configurare, eseguire e monitore i vari processi.

In Geoweb un Processo (anche detto gwProcess) è sempre subordinato alla presenza di una Classe. Una Classe (anche detta gwClass) è un metadato di Geoweb ed in generale è un'entità astratta che modellizza oggetti le cui caratteristiche sono presenti nelle colonne di una tabella su un DataBase. Nel contesto del workflow si può dire che una classe modellizza entità che detengono **lo stato corrente** di tutte **le variabili** di un processo delle quali vogliamo tenere traccia . Ogni variabile, o caratteristica, è nota al framework in quanto sopra vi è definito un metadato di Geoweb chiamato Attributo (anche detto gwAttribute) che lo mette in relazione con essa (relazione in generale non sempre biunivoca: in Geoweb un Attributo è spesso totalmente svincolato dalla colonna di una qualche tabella).

Una gwClass detiene in generale diversi gwAttribute, le cui definizioni vengono utilizzate per poter rappresentare nei modi più consoni le caratteristiche degli oggetti modellizzati dalle Classi.

In linea di massima un gwAttribute dovrà essere creato ogni qualvolta si desideri persistere il valore di una variabile di processo anche nella tabella che sta dietro la Classe, così da rendere tale valore utilizzabile secondo tutte le modalità standard di Geoweb. Nulla vieta che alcune variabili di processo dichiarate nella definizione di processo .bpmn, magari necessarie solo alle logiche di workflow, possano **esistere solo in Activiti**, senza avere necessariamente una controparte in Geoweb.

Quindi, una Classe che sottostà ad un Processo conterrà tutti gli Attributi necessari alla coesistenza e all'interscambio delle variabili di processo presenti nel motore di Activiti e quelle di Geoweb (cioè su DB, mediate dagli Attributi).

Tale interscambio è bidirezionale. In ogni punto del workflow dove è necessario avviene una **sincronizzazione delle variabili** da Activiti a Geoweb, e viceversa. A seconda dei casi:

- sincronizzazione da Activiti ⇒ a Geoweb. Essa avviene:
  - alla **creazione di ogni istanza di processo** in activiti (creando un record nella tabella della Classe di Processo).
  - prima e dopo **l'esecuzione dei groovy** di un **ServiceTask** (aggiornando il record dell'istanza di processo nella tabella della Classe di Processo).
- sincronizzazione da Geoweb ⇒ a Activiti. Essa avviene:
  - contestualmente al **completamento** di un qualsiasi **UserTask**, passando al metodo esposto da Activiti un set di variabili, fra cui sono presenti anche quelle provenienti da Geoweb.
  - più in generale anche nel **salvataggio di modifiche sulla form del task** (senza completarlo). Ciò è fatto per tenere sincronizzate le variabili in Activiti con quelle di Geoweb. Utile per i sistemi di Filtri.

Per far sì che questa sincronizzazione possa avvenire correttamente ci sono una serie di convenzioni

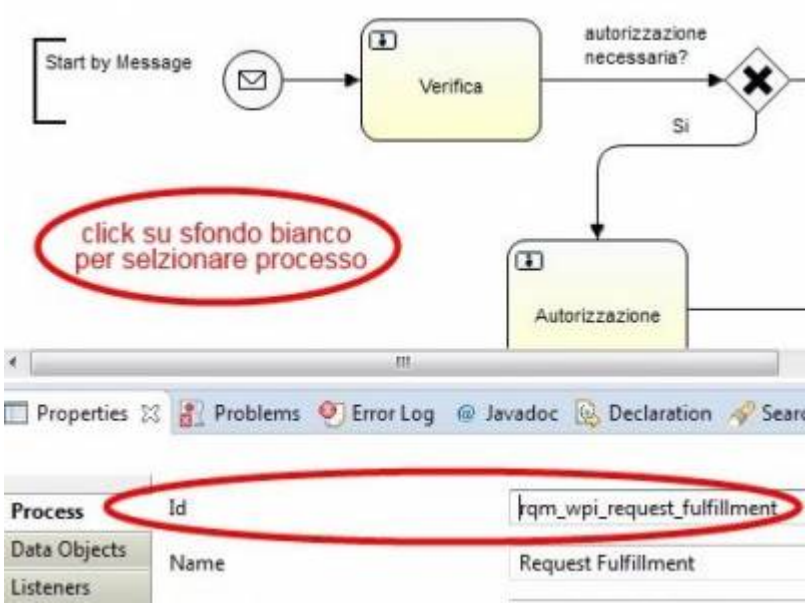
da rispettare nell'implementazione del workflow lato .bpmn e lato Geoweb.

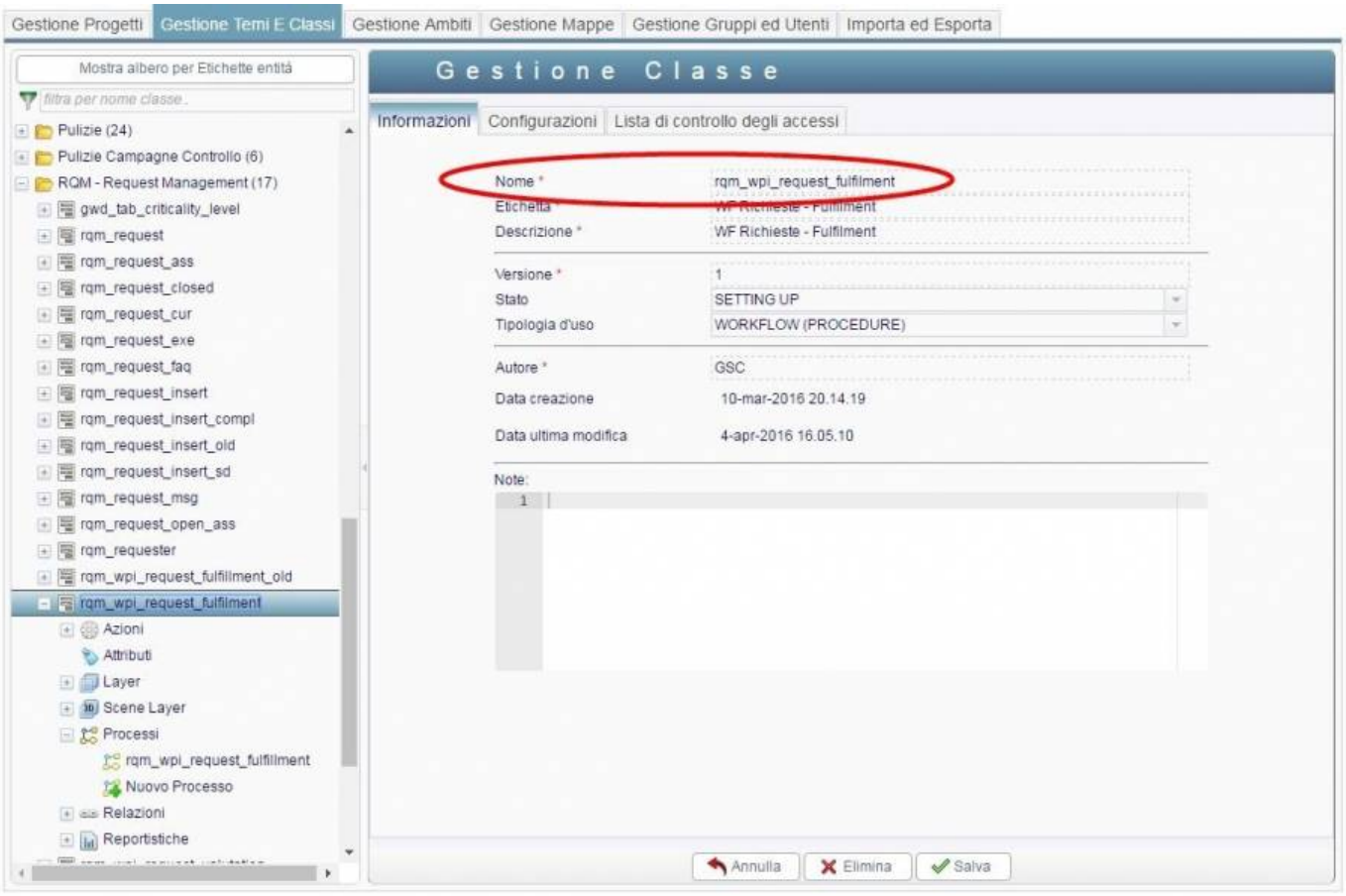
Convenzioni Implementazione

(verrà sempre mostrato prima lo screen della configurazione dell'editor grafico .bpmn, seguito da quello del Geoweb Admin)

- **Id process BPMN == Nome GwProcess**

Il nome di processo è univoco in Geoweb ed, essendo legato ad una sola Classe, è usato per esempio per recuperare le informazioni necessarie per costruire (grazie agli Attributi della Classe) le varie form utente (corrispettivo delle UserTask nel BPMN).






```
<process id="rqm_wpi_request_fulfillment" name="Request Fulfillment"
isExecutable="true">
```

- **Activiti formProperty Id == Nome Attributo**

Si può creare una form property per uno specifico UserTask, mappandola con uno specifico attributo di Geoweb (che deve avere columnName non nullo e dataType compatibile con quelli gestiti da Activiti (STRING, NUMBER ,INTEGER, DATE, BOOLEAN). Questo è necessario per:

- poter sovrascrivere, localmente al task, alcune proprietà generali dell'attributo definite in Geoweb.

La mappatura delle variabili fra Geoweb ed Activiti è valutata da un processo di sincronizzazione che avviene in automatico:

- da Geoweb ⇒ a Activiti
  - al completamento di ogni UserTask, al metodo esposto da Activiti vengono passati i valori correnti delle variabili di Geoweb mappate
- da Activiti⇒ a Geoweb
  - alla creazione di ogni istanza di processo in activiti
  - prima e dopo l'esecuzione dei groovy di un ServiceTask

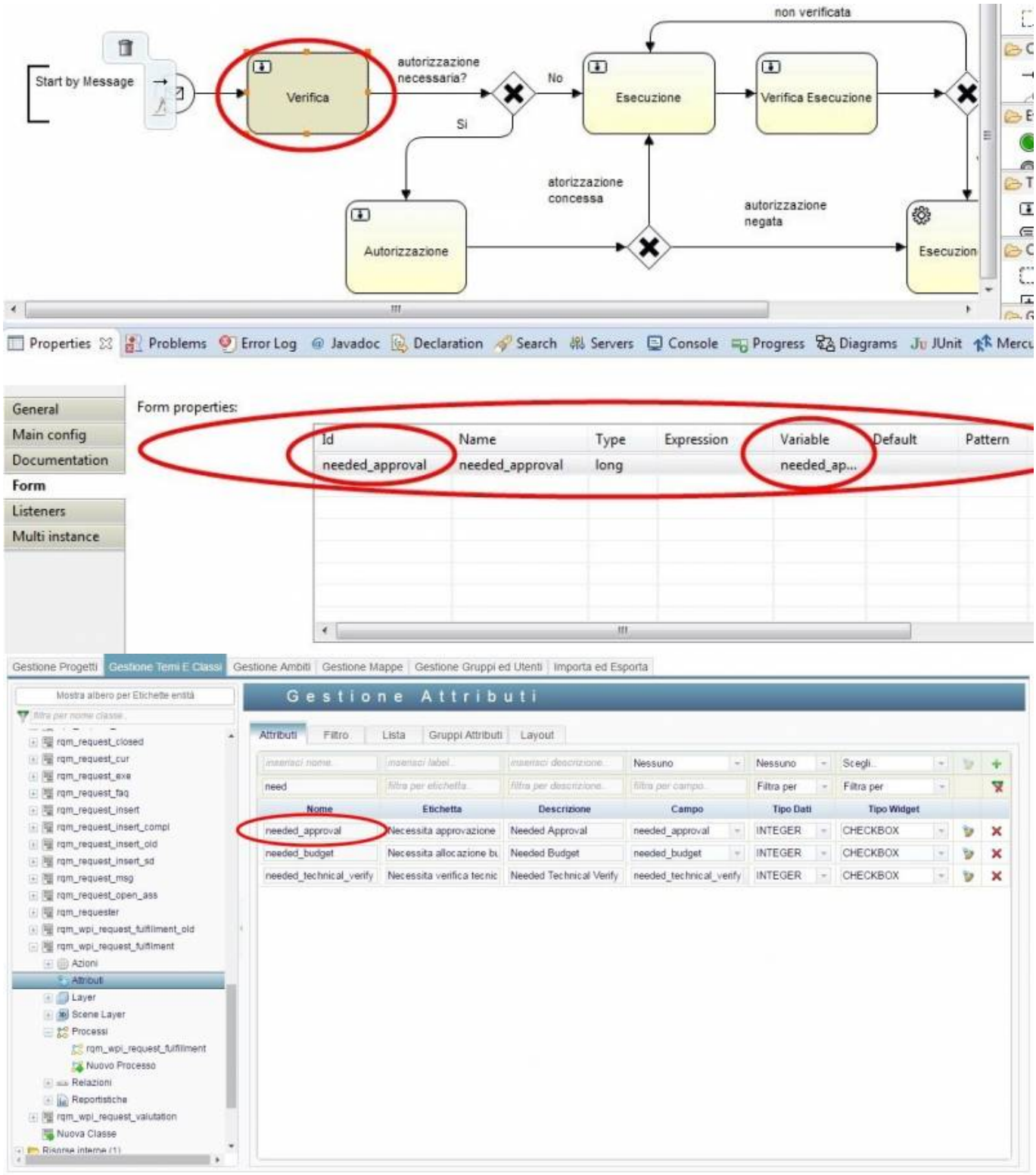
Ciò non esclude la possibilità di creare form property con id a piacere, di fatto creando variabili di processo con visibilità e ciclo di vita limitati all processo stesso, gestibili esclusivamente al suo interno, e comunque sia mai allineate in alcun modo in Geoweb.

Qui sotto stiamo dicendo ad Activiti che il valore del Widget CheckBox (costruito in base all'Attributo di columnName 'needed_approval' in Geoweb) mappato tramite id con uguale nome, dovrà essere

impostato sulla variabile di processo 'needed_approval'.

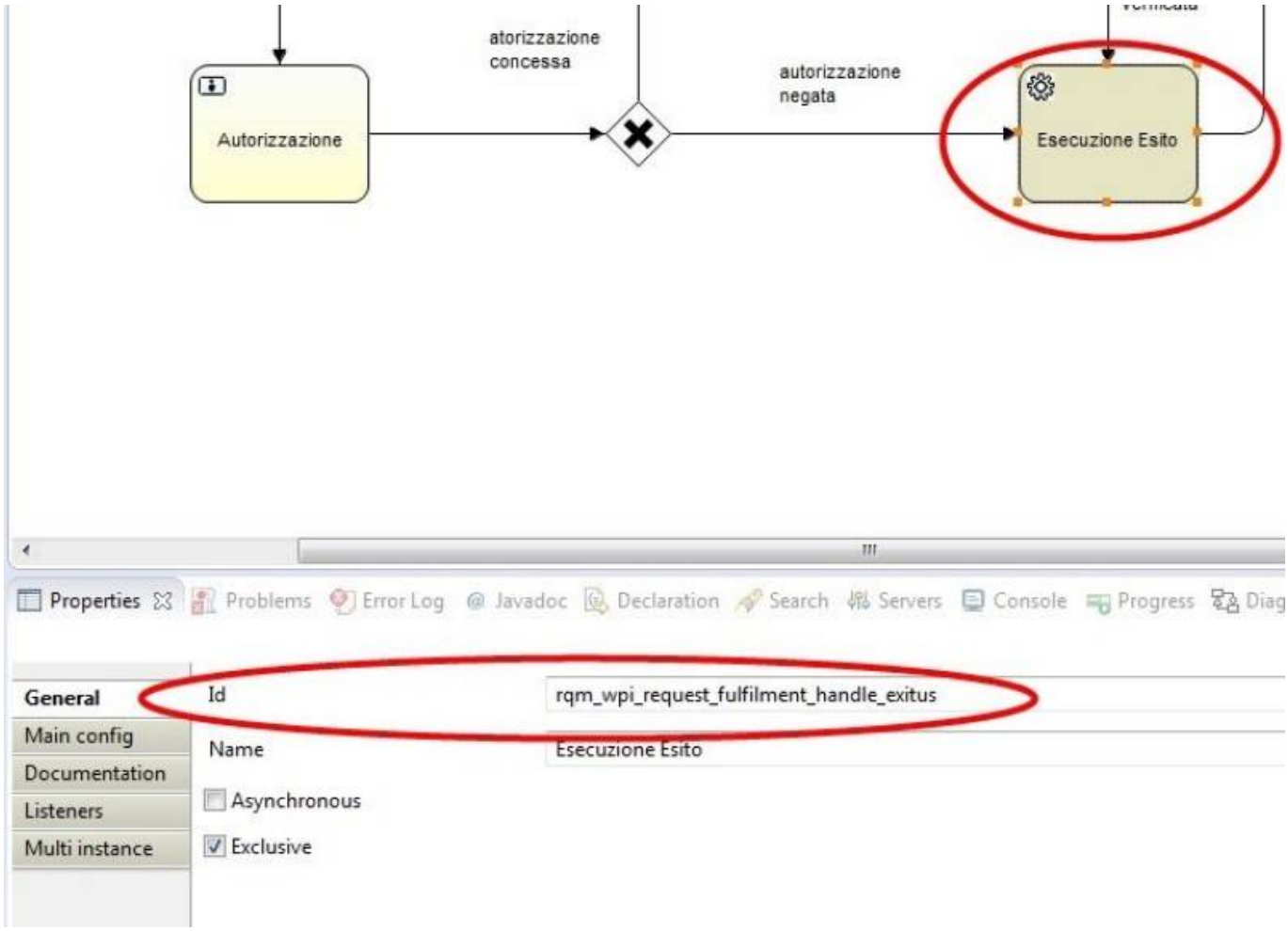
In questo caso essendo la variabile di processo anche uguale al columnName dell'Attributo questo valore non esisterà solo in Activiti, ma verrà sincronizzato nei modi previsti anche in Geoweb (su DB).

Per approfondimenti si rimanda alla apposita sezione UserTask.



- **Id ServiceTask == nome file .groovy (nei contenuti statici)**

Nell'implementazione di ServiceTask (vedi sotto), l'id del service task è usato per conoscere il nome file .groovy (senza estensione .groovy) contente le istruzioni da eseguire. Tale file viene cercato nella cartella dei contenuti statici di Geoweb, sotto il path **'WEB\groovy\'** .



UserTask

Di base Geoweb passa alla funzione che esegue il completamento di ogni UserTask, come variabili di processo, tutti gli attributi di Geoweb che soddisfano queste condizioni:

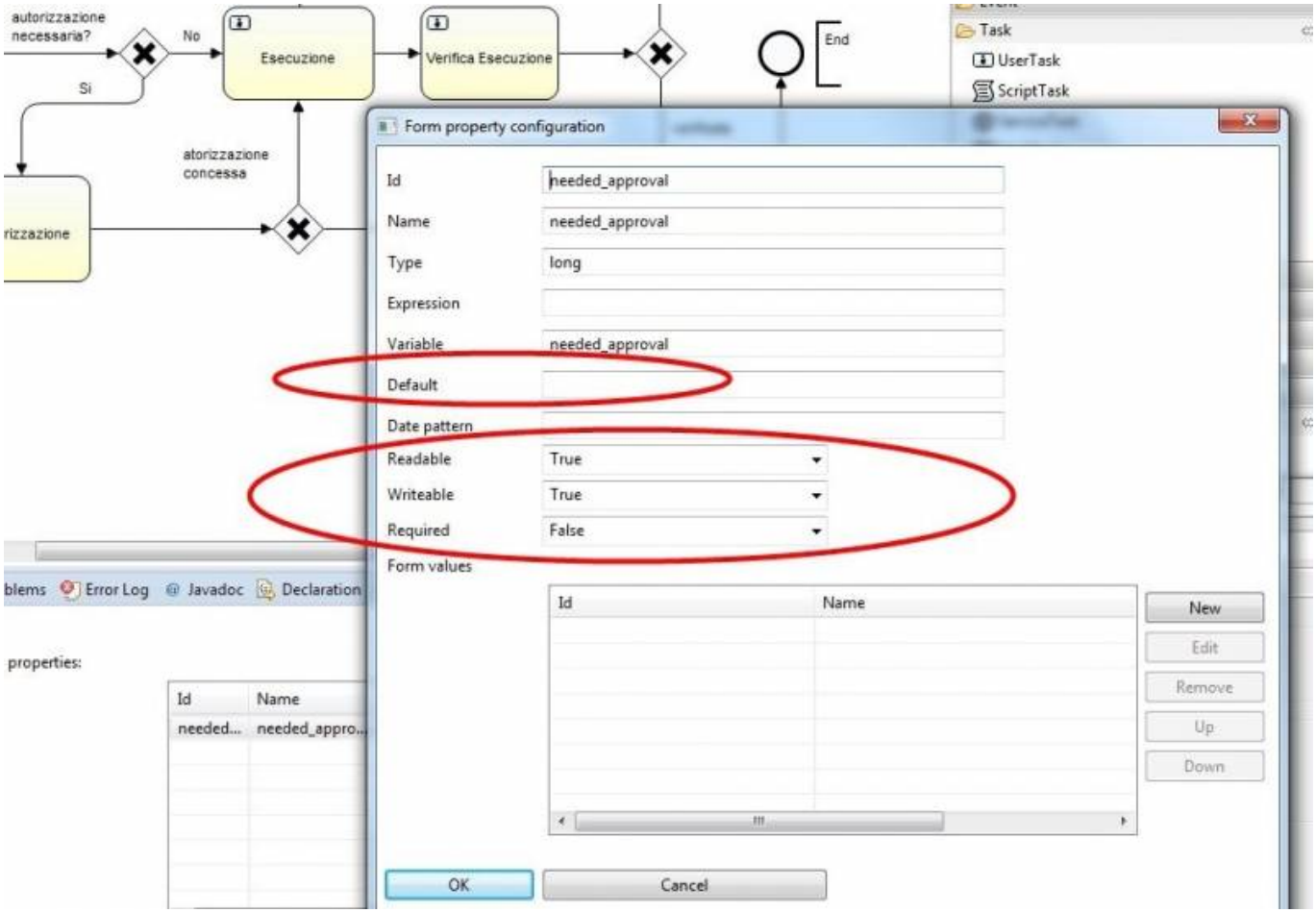
- columnName attributo Geoweb non nullo (dietro l'attributo c'è un campo reale sulla tabella del DB)
- dataType attributo Geoweb fra quelli compatibili per le variabili di processo di Activiti (*STRING, NUMBER, INTEGER, DATE, BOOLEAN*)

Nella sezione di configurazione '*Form*' si possono creare molteplici 'form properties'. Fermo restando quanto sopra questo può essere fatto per::

- poter sovrascrivere, localmente al task, alcune proprietà generali dell'attributo definite in Geoweb.
- aggiungere variabili (di task e di processo) che hanno vita solo in Activiti (in quanto non vengono in alcun modo persistite in Geoweb, su DB), che possono essere modificate nei .groovy di processo e che possono venire valutate nelle logiche di processo (per esempio nelle *condition* dei *Flow* dei *GateWay*).

Da notare che queste form properties da sole non aggiungono alcun widget di input alla form del task: a comandare sono sempre gli attributi legati (tramite i gruppi attributi) al *DetailLayout* di Geoweb, che è associato a sua volta (tramite *formKey*) allo specifico *UserTask* di Activiti.

In particolare all'interno dell'interfaccia per editare queste form property sono presenti le voci **Readable**, **Writeable**, **Required** e **Default**.



Queste sono usate per sovrascrivere per una data form di un dato UserTask le proprietà **required**, **readonly** e **defaultValue** degli Attributi di Geoweb, che di loro varrebbero per tutti i Layout in cui sono presenti quegli attributi.

Questo è utile per esempio per mostrare a due utenti, che hanno privilegi diversi di editare un widget (leggi variabile di processo), due form basate sullo stesso Layout, uguali in tutto e per tutto tranne che per, magari, la presentazione su una delle due form di una data proprietà in sola lettura .

Required e *Default* sono i corrispettivi di Geoweb e sovrascrivono, localmente ad una specifica form, gli equivalenti imposti nell'Attributo di Geoweb.

Readable e *Writeable* hanno invece significati particolari in Activiti. *Readable* se a true permette ad Activiti di poter usare tali variabili in lettura (per esempio nelle espressioni ${value1}). *Writeable* a true ne permette inoltre anche la scrittura.

Readable e *Writeable* dovrebbero essere generalmente lasciate sempre a true (lo sono di default).

Il *Writeable* di Activiti, nelle prime implementazione di processi, veniva usato per poter sovrascrivere

localmente il *readonly* di Geoweb (se *Writable* è true, *readonly* vale false, e viceversa). Ciò poteva generare un certo numero di problemi, quando incautamente impostato a false. Infatti quando ciò accade bisogna essere certi che nessuno tenti di modificare quella variabile di processo, fino a quando *Writeable* non torni a true.

Per queste ragioni l'uso di *Writeable* in logica inversa per sovrascrivere il readonly di Geoweb è da considerarsi deprecato, e ne viene scoraggiato l'uso.

Dalla versione di Geoweb 4.2.5 è infatti possibile aggiungere a mano nel file di testo .bpmn, per un dato tag activiti:formProperty, l'attributo **readonly** (con true/false come valori ammessi) cosi:

```
<activiti:formProperty id=“needed_approval” name=“needed_approval”
type=“long” variable=“needed_approval”
readonly=“true”></activiti:formProperty>
```

L'attributo readonly non è uno standard ne di Activiti ne delle specifiche .BPMN, ma verrà cercato da Geoweb e, se presente, verrà applicato per ultimo.

ServiceTask

A differenza degli *UserTask*, i *ServiceTask* sono eseguiti dal motore di workflow, che sostanzialmente invoca l'esecuzione di una classe Java esterna ad Activiti.

Ci sono varie (4) modalità esposte da Activiti per invocare l'esecuzione di codice Java.

Nell'integrazione di Activiti in Geoweb viene usata la modalità che prevede la valutazione di un'espressione UEL, allo scopo di far eseguire delle procedure di scripting scritte in Groovy. Ciò viene fatto usando l'attributo **activiti:expression** nel file .bpmn (vedi sotto).

Con queste procedure si possono eseguire un'ampia gamma di operazioni, in quanto si hanno anche a disposizione tutta una serie di servizi (services) esposti da Geoweb.

A seconda della complessità delle azioni richieste, queste procedure possono essere scritte designatore del processo o da uno sviluppatore software.

Dall'editor grafico 'Task Type' va impostato su **Expression**. E su 'Expression' bisogna mettere la seguente espressione fissa:

${gwServiceTask.executeGroovy(execution)}

Questa espressione farà si che venga eseguito il codice contenuto dentro il file .groovy presente nei contenuti statici di Geoweb tale per cui **nome_file_groovy == id_service_task**, cosi come mostrato prima nella sezione convenzioni.

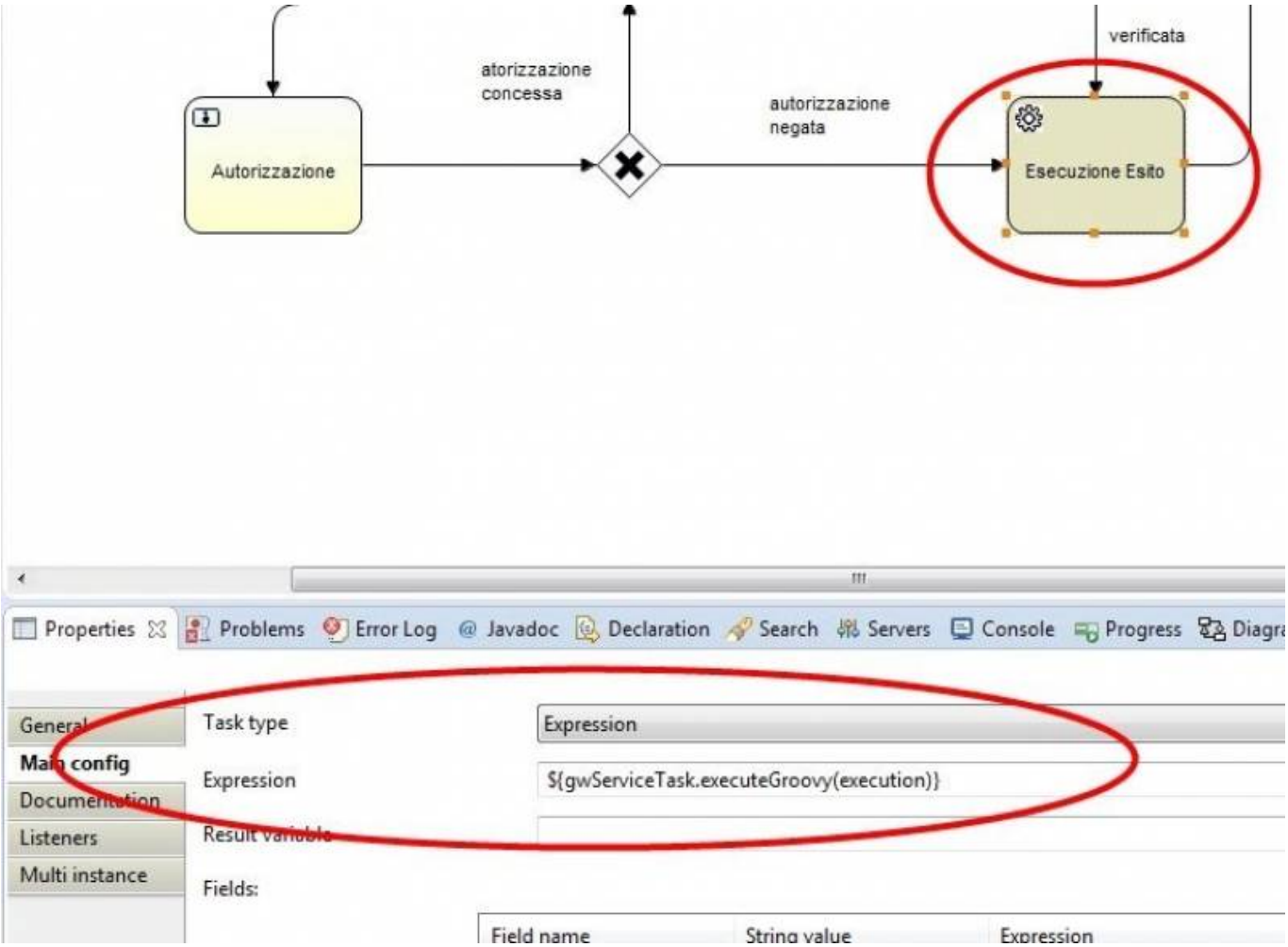
Senza soffermaci troppo sul perchè l'espressione funziona, basti sapere che *gwServiceTask* è un servizio di Geoweb in qualche modo esposto e reso utilizzabile dal motore di workflow di Activiti. Questo servizio espone un metodo *executeGroovy()*, a cui viene passata in ingresso la variabile (predefinita in Activiti) *execution* che, tra l'altro, contiene una mappa chiave-valore contente tutte le variabili di processo (già riallineate con i valori di quelle di Geoweb).

Tutte le variabili di processo contenute in *execution* sono direttamente rese visibili al codice del

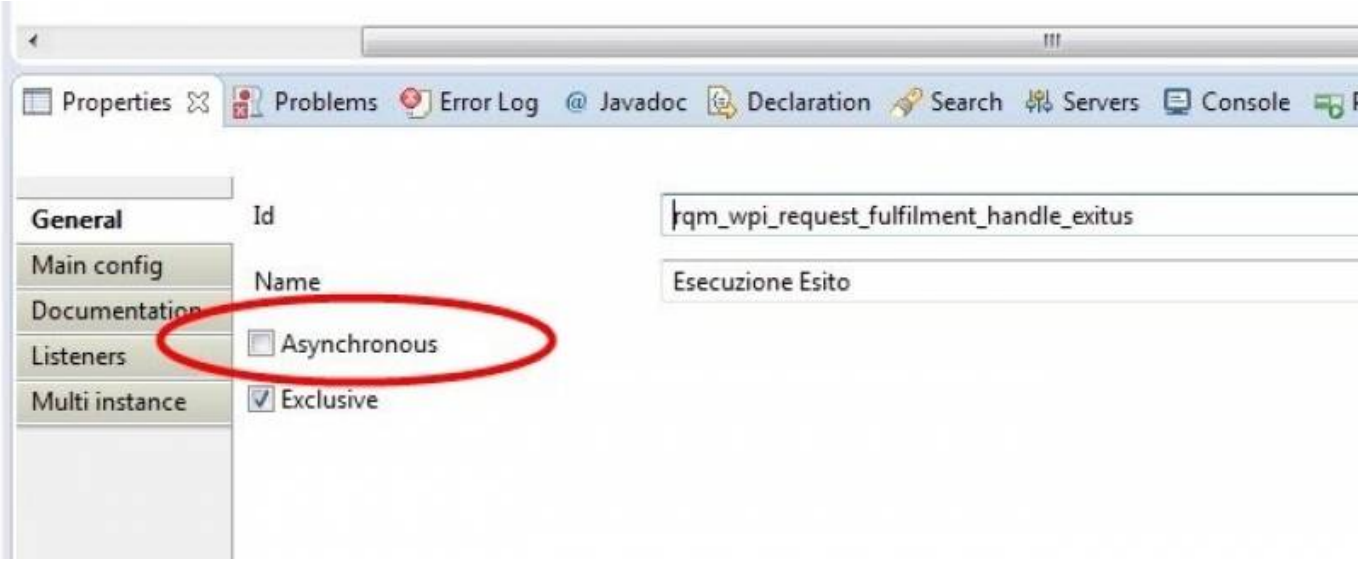
.groovy.

Vengono inoltre rese visibili al .groovy le seguenti variabili:

- [geowebService], tutti i servizi di Geoweb
- *log*, un servizio usabile dal .groovy per i log
- *gw_activeUser*, String utente in sessione, rimpiazza il deprecato activeUser
- *gw_activeGroup*, String utente in sessione, rimpiazza il deprecato activeGroup
- *activeScopes*, List<SessionScopeDefinitionValue>, ognuna con le proprietà activeScopeDefinition e activeScopeValue
- *mgSession*, String, sessione MapGuide
- *projectName*, String
- *execution*, variabile di Activiti



Una cosa importante, spesso fonte di comportamenti indesiderati, bisogna assicurarsi di avere la spunta della voce **Asynchronous** settata a **false**. Questo a meno di necessità particolari e comunque sia sempre sapendo cosa si sta facendo.



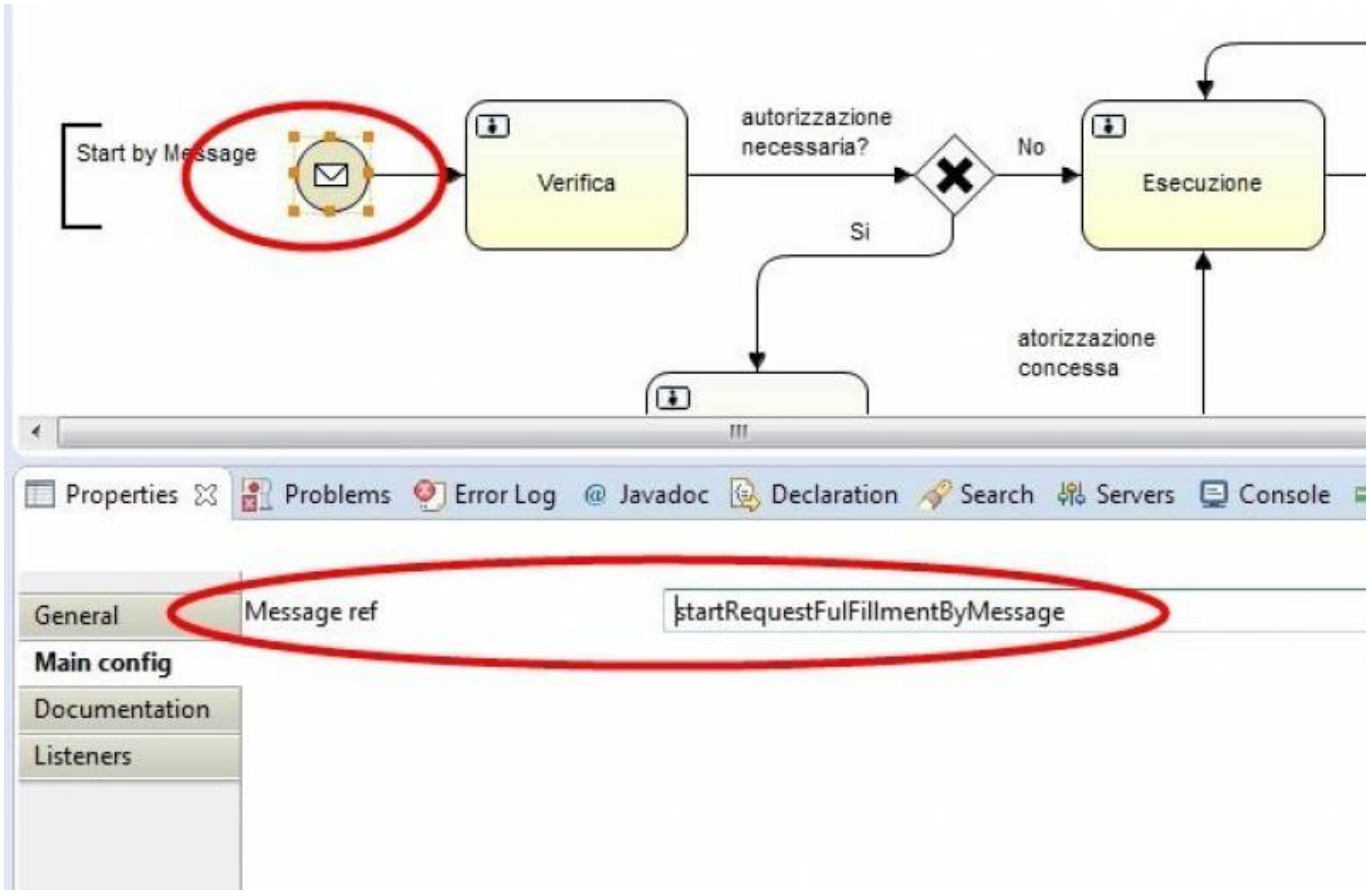
In file .bpmn:

```
<serviceTask id="rqm_wpi_request_fulfilment_handle_exitus" name="Esecuzione
Esito"
activiti:expression="${gwServiceTask.executeGroovy(execution)}"></serviceTask
>
```

MessageStartEvent

Permette di avviare un'istanza di processo usando un messaggio con uno specifico identificativo. Facendo il deploy di una data definizione di processo (leggi file .bpmn) con uno o più message start event valgono le seguenti considerazioni:

- Il **name** del message deve essere univoco all'interno di una data definizione di processo. Activiti lancia un'eccezione deploy di una definizione di processo contenente più message con lo stesso name.
- Il **messageRef** deve essere univoco all'interno di tutte le definizioni di processo. Activiti lancia un'eccezione se il messaggio era già stato precedentemente usato per il dispiegamento di un'altra definizione di processo
- Dispiegando un nuova versione della stessa definizione di processo, tutti le sottoscrizioni ai messaggi delle precedenti versioni sono annullate



In file .bpmn:

```
<startEvent id="messagestartevent1" name="Message start">
<messageEventDefinition messageRef="startRequestFulFillmentByMessage">
</messageEventDefinition> </startEvent>
```

Un processo può essere avviato da messaggio in un qualsiasi file .groovy:

```
def processVariables = [:];
...
workflowService.startProcessByMessage(
    "startOrderProposalByMessage",
     processVariables
);
```

Un processo può essere avviato da messaggio anche tramite più azioni javascript Geoweb.

Azione generica per avviare un processo da messaggio, ed aprire il primo UserTask per l'utente corrente, se presente.

```
   var parameterMap = {};
   //.. populating parameterMap
   var groovyName = 'order_proposal_startbymessage';
   groovyActionGeneric( groovyName, parameterMap, function(responseHashMap){
```

```
       var taskNextStepInfosHM = responseHashMap.result;
       handleTaskNextStep(taskNextStepInfosHM);
   });
```

E relativa controparte groovy:

Invoca il metodo startProcessByMessageForTaskNextStepInfos() su workflowService. La mappa che restituisce viene fatta ritornare a sua volta dal groovy, cosi che sarà accedibile in seguito sull'azione js (tramite 'responseHashMap.result;')

def processVariables = [:];

```
//populating processVariables
processVariables.ids = ids;

processVariables.iniziator = activeUser; //activeUser is a predefinited
passed variabled

processVariables.proposal_status = "In fase di compilazione";

def taskNextStepInfosHM =
workflowService.startProcessByMessageForTaskNextStepInfos(
    "startOrderProposalByMessage",
    processVariables
);
return taskNextStepInfosHM;
```

Azione JS in caso di avvio di un processo da messaggio, passando il record, da **dettaglio o da lista Geoweb**.

L'oggetto parameterQueryList è passato in automatico in caso di Azione GeoWeb di tipo 'List'su item selezionati (gwClassList). In caso azioni di tipo 'Detail' va ricreato cosi da poter usare lo stesso script .groovy sia per azioni 'List' che 'Detail'. In alternativa usare il metodo groovyActionDetail(..), adattando eventualmente il groovy.

```
     var parameterQueryList = {
       filters: [
           {
               attributeGwid: null,
               columnName: 'pk_planned_activity',
               condition: 'OR',
               filterType: 'INTEGER',
               operator: '=',
               value: [data.itemDB.pk_planned_activity]
             }
        ]
    };
    var groovyName = 'order_proposal_startbymessage';
    groovyActionList(groovyName, parameterQueryList, grid,
function(responseHashMap){
        var taskNextStepInfosHM = responseHashMap.result;
```

```
        handleTaskNextStep(taskNextStepInfosHM);
    });
```

Da notare che handleTaskNextStep esegue le stesse identiche azioni che verrebbero eseguiti dai tasti avvia processo/completa task, quindi, avviando un processo, possiamo avere casi differenti dall'apertura del task:

- se il primo task è assegnato all'utente corrente viene aperta la form gli è attivata la possibilità di completare il task.
- se esso termina senza passare per alcun task utente, verrà mostrato un messaggio.
- se il primo task è assegnato ad un altro utente verrà mostrato un messaggio.
- se il primo task è assegnato ad una lista di utenti o gruppi a cui è associato l'utente corrente viene aperta la form in consultazione con la possibilità di richiedere (to claim) l'attività.

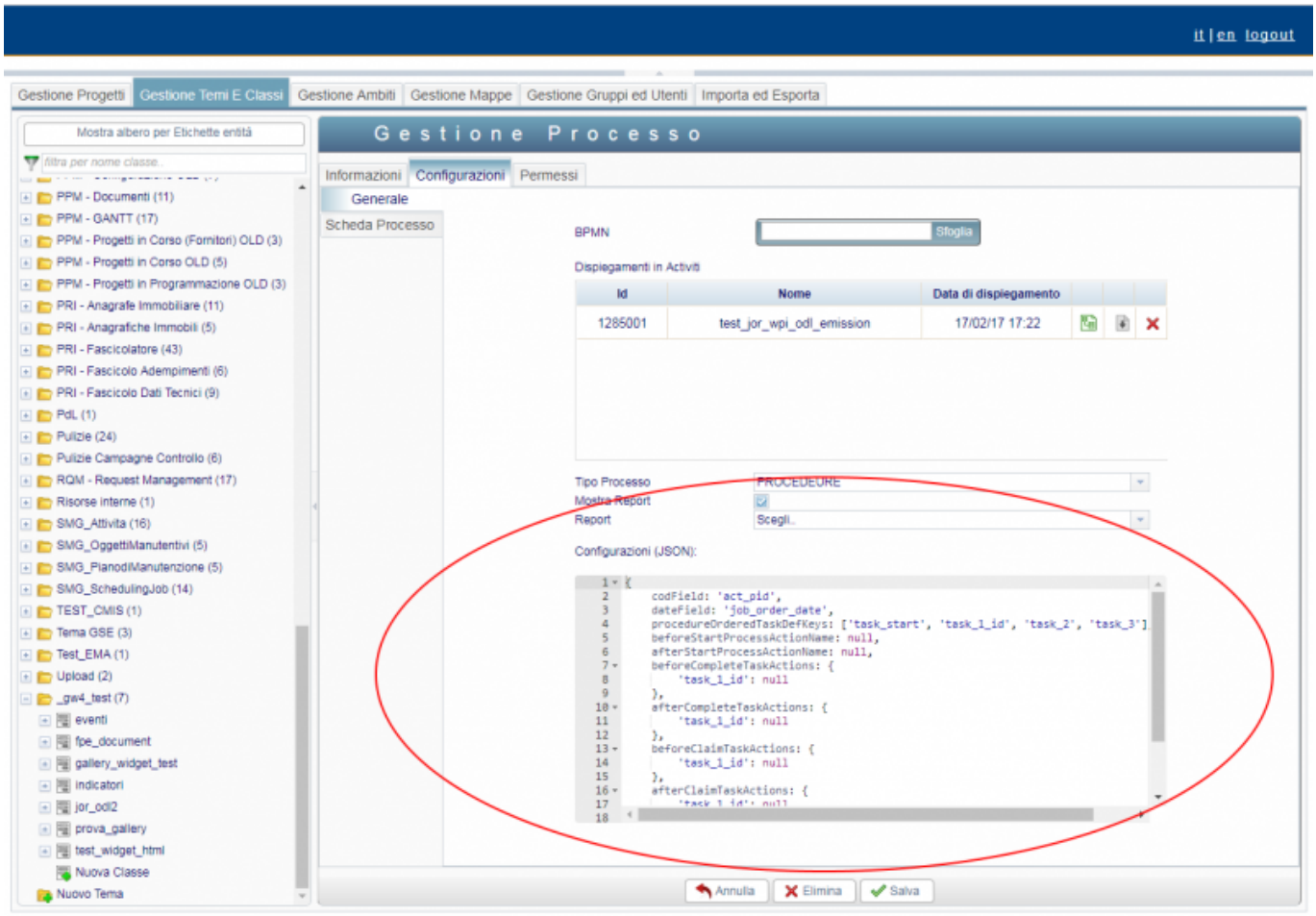
E relativa controparte groovy:

Invoca il metodo startProcessByMessageForTaskNextStepInfos() su workflowService. La mappa che restituisce viene fatta ritornare a sua volta dal groovy, cosi che sarà accedibile in seguito sull'azione js (tramite 'responseHashMap.result;')

```
   def processVariables = [:];
   def ids = [];
   for(item in items)
   ids.add(item.pk_planned_activity);
   processVariables.ids = ids;
   processVariables.iniziator = activeUser; //activeUser is a predefinited
passed variabled
   processVariables.proposal_status = "In fase di compilazione";
   def taskNextStepInfosHM =
workflowService.startProcessByMessageForTaskNextStepInfos(
       "startOrderProposalByMessage",
       processVariables
   );
   return taskNextStepInfosHM;
```

Customizzazione Processo

Nella sezione 'configJSON' della scheda 'Configurazioni' ⇒ 'Generale' del menu accessibile da 'Gestione Temi e Classi' ⇒ [Tema] ⇒ [Classe] ⇒ 'Processi' ⇒ [Processo] è possibile configurare tutta una serie di parametri.



Azioni Javascript di Classe

Queste azioni javascript di classe che vengono eseguite in determinati hook point del workflow.

Queste in genere sono eseguite con i seguenti parametri come argomenti:

- ***params***, contenente parametri tra cui processDefinitionKey, gwClassName, taskFormContainerId, openingDetailMode, etc..
- ***dojo/Deferred*** grazie al quale sarà configurabile la prosecuzione o meno del normale flusso di processo:
  - deferred.resolve(); causerà soddisfacimento della promessa e la prosecuzione del flusso di processo
  - deferred.reject("Error Message"); causerà il rifiuto del soddisfacimento della promessa di risoluzione, causa eccezione.
  - deferred.cancel(); causerà la cancellazione del deferred, rendendo impossibile il soddisfacimento della promessa di risoluzione (come reject, ma in questo caso la promessa non va semplicemente gestita, in quanto irrilevante)

01 beforeStartProcessActionName

type: String

default: null

required: false

E' il name della gwAction di Classe che verrà eseguita prima dell'avvio del processo. Essa verrà eseguita con i seguenti parametri:

- ***params***, contenente parametri tra cui processDefinitionKey, gwClassName, taskFormContainerId, openingDetailMode, etc..
- ***dojo/Deferred*** grazie al quale sarà configurabile la prosecuzione o meno del normale flusso di processo:
  - deferred.resolve(); causerà soddisfacimento della promessa e la prosecuzione del flusso di processo
  - deferred.reject("Error Message"); causerà il rifiuto del soddisfacimento della promessa di risoluzione, causa eccezione.
  - deferred.cancel(); causerà la cancellazione del deferred, rendendo impossibile il soddisfacimento della promessa di risoluzione (come reject, ma in questo caso la promessa non va semplicemente gestita, in quanto irrilevante)

Esempio Supponiamo di voler controllare, prima di avviare il processo, lato .groovy delle condizioni che potrebbero eventualmente bloccarne l'avvio, o come in questo caso far scegliere l'utente se proseguire comunque.

Nel configJSON del processo aggiungere:

```
''...beforeStartProcessActionName: 'test_beforeStartProcessActionName'
,...''
```

Creare quindi l'azione test_beforeStartProcessActionName nella gwClass che ospita il gwProcess.

```
//params and deferred are function arguments
try{
     var taskFormContainerId = params.taskFormContainerId;
     var taskFormContainer =
dijit.registry.getEnclosingWidget(dojo.byId(taskFormContainerId));
     var scriptName = 'script_name';
     var parameterMap = taskFormContainer.getFormValues();
     var callback = function(/Object/ data){
          var result = data.result;
          var success = result.success;
          var description = result.description;
          if(success){
               deferred.resolve();
          }else{
               invisibleStandbyWidget.hide();
               var title = confirmRequestLabel; //optional
               var message = description+'. Creare comunque una nuova
istanza di processo?';
               //optional
               var yesCallback = function(){
                    invisibleStandbyWidget.show();
                    deferred.resolve();
```

```
                };
                var noCallback = function(){
                     deferred.cancel('stopped by user');
                };
                var cancelCallback = function(){
                     deferred.cancel('stopped by user');
                };
                var params = {
                     title: title,
                     message: message,
                     yesCallback: yesCallback,
                     noCallback: noCallback,
                     cancelCallback: cancelCallback
                };
                showYesNoCancelDialog(params);
           }
     };

     groovyActionGeneric(scriptName, parameterMap, callback);

}catch(e){
     console.log('test_beforeCompleteTaskAction error');
     console.log(e);
     deferred.reject(e.toString());
}
```

Il groovy dovrebbe essere cosi strutturato:

```
def result = [:];
def success = true;
def description = null;
try{
     //fare tutti i check necessari ed eventualmente lanciare eccezioni
     if(condition==true){
          throw new RuntimeException("ATTENZIONE! Errore!");
     }
      description ='ok';
}catch(Exception e){
     log.error(e.getMessage(), e);
     success = false;
     description = e.getMessage();
}finally{
     result.success = success;
     result.description = description;
}
return result;
```

02 afterStartProcessActionName

type: String

default: null

required: false

E' il name della gwAction di Classe che verrà eseguita **dopo** l'avvio del processo. Essa verrà eseguita con i seguenti parametri:

params, contenente processDefinitionKey, processInstanceId, gwClassName, userId, taskFormContainerId, openingDetailMode

Esempio Supponiamo di voler forzare il refresh di tutti i componenti basati sulla classe 'gw_class_to_refresh', dopo che è stato avviato il processo.

Nel configJSON del processo aggiungere:

```
...
afterStartProcessActionName: 'test_afterStartProcessActionName',
...
```

Creare quindi l'azione test_afterStartProcessActionName nella gwClass che ospita il gwProcess.

```
//params are function arguments
var gwClassName = 'gw_class_name';
publishGwClassUpdate(gwClassName);
```

03 beforeCompleteTaskActions

type: Map<String,String>

default: null

required: false

E' una mappa con chiave la Task Definition Key del singolo e per valore il name della gwAction di Classe che verrà eseguita prima del completamento del Task. Nella mappa posso impostare un gwAction diversa per ogni Task Definition. La gwAction verrà eseguita con i seguenti parametri:

- ***params***, contenente processDefinitionKey, gwClassName, taskFormContainerId, openingDetailMode, buttonWidget
- ***dojo/Deferred*** grazie al quale sarà configurabile la prosecuzione o meno del normale flusso di processo:
  - deferred.resolve(); causerà soddisfacimento della promessa e la prosecuzione del flusso di processo
  - deferred.reject("Error Message"); causerà il rifiuto del soddisfacimento della promessa di risoluzione, causa eccezione.
  - deferred.cancel(); causerà la cancellazione del deferred, rendendo impossibile il soddisfacimento della promessa di risoluzione (come reject, ma in questo caso la

promessa non va semplicemente gestita, in quanto irrilevante)

Esempio Supponiamo di voler controllare, prima di completare il task con taskDefinitionKey 'task_definition_key', lato .groovy delle condizioni che potrebbero eventualmente bloccarne il completamento, o come in questo caso far scegliere l'utente se proseguire comunque tramite un dialogo modale.

Nel configJSON del processo aggiungere:

```
...
beforeCompleteTaskActions: {
    'task_definition_key':  'test_beforeCompleteTaskAction'
},
...
```

Creare quindi l'azione test_beforeCompleteTaskAction nella gwClass che ospita il gwProcess.

```
    //params and deferred are function arguments
    try{
        var taskFormContainerId = params.taskFormContainerId;
        var taskFormContainer =
    dijit.registry.getEnclosingWidget(dojo.byId(taskFormContainerId));
        var scriptName = 'script_name';
        var parameterMap = taskFormContainer.getFormValues();
        var callback = function(/Object/ data){
            var result = data.result;
            var success = result.success;
            var description = result.description;
            if(success){
               deferred.resolve();
            }else{
                invisibleStandbyWidget.hide();
                var title = confirmRequestLabel; //optional
                var message = description+'. Proseguire Comunque?';
//optional
                var yesCallback = function(){
                    invisibleStandbyWidget.show();
                    deferred.resolve();
                };
                var noCallback = function(){
                    deferred.cancel('stopped by user');
                };
                var cancelCallback = function(){
                    deferred.cancel('stopped by user');
                };
                var params = {
                    title: title,
                    message: message,
                    yesCallback: yesCallback,
                    noCallback: noCallback,
                    cancelCallback: cancelCallback
```

```
                };
                showYesNoCancelDialog(params);
            }
        };
        groovyActionGeneric(scriptName, parameterMap, callback);
       }catch(e){
        console.log('test_beforeCompleteTaskAction error');
        console.log(e);
        deferred.reject(e.toString());
      }
```

Il groovy dovrebbe essere cosi strutturato:

```
    def result = [:];
    def success = true;
    def description = null;
    try{
        //fare tutti i check necessari ed eventualmente lanciare eccezioni
        if(condition==true){
            throw new RuntimeException("ATTENZIONE! Errore!");
        }
        description ='ok';
    }catch(Exception e){
        log.error(e.getMessage(), e);
        success = false;
        description = e.getMessage();
    }finally{
        result.success = success;
        result.description = description;
    }
    return result;
```

04 afterCompleteTaskActions

type: String default: null required: false

E' una mappa con chiave la Task Definition Key del singolo e per valore il name della gwAction di Classe che verrà eseguita **dopo** il completamento del Task. Nella mappa posso impostare un gwAction diversa per ogni Task Definition. La gwAction verrà eseguita con i seguenti parametri:

- ***params***, contenente taskId, processDefinitionKey, processInstanceId, taskVariables, gwClassName, userId, itemId

Esempio Supponiamo di voler forzare il refresh di tutti i componenti basati sulla classe 'gw_class_to_refresh', dopo che è stato completato il task con Task Definition Key 'task_definition_key'.

Nel configJSON del processo aggiungere:

```
...
afterCompleteTaskActions: {
    'task_definition_key':  'test_afterCompleteTaskAction'
},
...
```

Creare quindi l'azione test_afterStartProcessActionName nella gwClass che ospita il gwProcess.

```
 //params are function arguments
  var gwClassName = 'gw_class_name';
  publishGwClassUpdate(gwClassName);
```

05 beforeClaimTaskActions

type: Map<String,String>

default: null

required: false

E' una mappa con chiave la Task Definition Key del singolo e per valore il name della gwAction di Classe che verrà eseguita **prima** del claim del Task. Nella mappa posso impostare un gwAction diversa per ogni Task Definition. La gwAction verrà eseguita con i seguenti parametri:

- **params**, contenente processDefinitionKey, gwClassName, taskFormContainerId, openingDetailMode, buttonWidget
- **dojo/Deferred** grazie al quale sarà configurabile la prosecuzione o meno del normale flusso di processo:
  - deferred.resolve(); causerà soddisfacimento della promessa e la prosecuzione del flusso di processo
  - deferred.reject("Error Message"); causerà il rifiuto del soddisfacimento della promessa di risoluzione, causa eccezione.
  - deferred.cancel(); causerà la cancellazione del deferred, rendendo impossibile il soddisfacimento della promessa di risoluzione (come reject, ma in questo caso la promessa non va semplicemente gestita, in quanto irrilevante)

Esempio

Supponiamo di voler controllare, prima di completare il task con taskDefinitionKey 'task_definition_key', lato .groovy delle condizioni che potrebbero eventualmente bloccarne il completamento, o come in questo caso far scegliere l'utente se proseguire comunque tramite un dialogo modale.

Nel configJSON del processo aggiungere:

```
   ...
beforeClaimTaskActions: {
    'task_definition_key':  'test_beforeClaimTaskAction'
},
...
```

Creare quindi l'azione test_beforeClaimTaskAction nella gwClass che ospita il gwProcess.

```
        //params and deferred are function arguments
    try{
        var taskFormContainerId = params.taskFormContainerId;
        var taskFormContainer =
    dijit.registry.getEnclosingWidget(dojo.byId(taskFormContainerId));
        var scriptName = 'script_name';
        var parameterMap = taskFormContainer.getFormValues();
        var callback = function(/Object/ data){
            var result = data.result;
            var success = result.success;
            var description = result.description;
            if(success){
                deferred.resolve();
            }else{
                invisibleStandbyWidget.hide();
                var title = confirmRequestLabel; //optional
                var message = description+'. Proseguire Comunque?';
//optional
                var yesCallback = function(){
                    invisibleStandbyWidget.show();
                    deferred.resolve();
                };
                var noCallback = function(){
                    deferred.cancel('stopped by user');
                };
                var cancelCallback = function(){
                    deferred.cancel('stopped by user');
                };
                var params = {
                    title: title,
                    message: message,
                    yesCallback: yesCallback,
                    noCallback: noCallback,
                    cancelCallback: cancelCallback
                };
                showYesNoCancelDialog(params);
            }
        };
        groovyActionGeneric(scriptName, parameterMap, callback);
    }catch(e){
        console.log('test_beforeCompleteTaskAction error');
        console.log(e);
        deferred.reject(e.toString());
    }
```

Il groovy dovrebbe essere cosi strutturato:

```
    def result = [:];
    def success = true;
    def description = null;
    try{
        //fare tutti i check necessari ed eventualmente lanciare eccezioni
        if(condition==true){
            throw new RuntimeException("ATTENZIONE! Errore!");
        }
        description ='ok'
    }catch(Exception e){
        log.error(e.getMessage(), e)
        success = false;
        description = e.getMessage();
    }finally{
        result.success = success;
        result.description = description;
    }
    return result;
```

06 afterClaimTaskActions

type: String

default: null

required: false

E' una mappa con chiave la Task Definition Key del singolo e per valore il name della gwAction di Classe che verrà eseguita **dopo** il claim del Task. Nella mappa posso impostare un gwAction diversa per ogni Task Definition. La gwAction verrà eseguita con i seguenti parametri:

- ***params***, contenente taskId, processDefinitionKey, processInstanceId, taskVariables, gwClassName, userId, itemId

Esempio Supponiamo di voler forzare il refresh di tutti i componenti basati sulla classe 'gw_class_to_refresh', dopo che è stato fatto il claim del task con Task Definition Key 'task_definition_key'.

Nel configJSON del processo aggiungere:

```
 ...
 afterClaimTaskActions: {
'task_definition_key':  'test_afterClaimTaskAction'
 },
 ...
```

Creare quindi l'azione test_afterClaimTaskAction nella gwClass che ospita il gwProcess.

```
 //params are function arguments
```

```
var gwClassName = 'gw_class_name';
publishGwClassUpdate(gwClassName);
```

07 processEndedGwActionName

type: String

default: null

required: false

E' il name della gwAction di Classe che verrà eseguita alla fine del processo. Essa verrà eseguita con i seguenti parametri:

- ***params***, contenente processDefinitionKey, processInstanceId, userId, formChangedValues, taskVariables

Esempio Supponiamo di voler notificare Geoweb lato client del fatto che, nell'esecuzione del workflow, sono stati modificati alcuni record sulla GwClass 'secondary_class'.

Nel configJSON del processo aggiungere:

```
...
processEndedGwActionName: 'test_processEndedGwActionName' ,
...
```

Creare quindi l'azione test_afterClaimTaskAction nella gwClass che ospita il gwProcess.

```
publishGwClassUpdate('secondary_class');
```

Bottoni e Label workflow

E' possibile modificare in dettaglio bottoni e relative label di tutte le form. Ciò può essere fatto task per task, od a livello dell'intero processo.

Proprietà globali:

- **allTaskForm_hideDocumentation**, boolean, default false, se true nasconde il contenuto descrittivo *Documentation* del .BPMN ed il bottone che mostra il diagramma di processo
- **allTaskForm_hideCloseButton**, boolean, default false, se true nasconde bottone di chiusura dialog
- **allTaskForm_hideRestoreButton**, boolean, default false, se true nasconde bottone di annulla modifche
- **allTaskForm_hideSaveButton**, boolean, default false, se true nasconde bottone di salvataggio
- **allTaskForm_completeTaskLabel**, string, se omesso viene usato un default (label del tasto

che serve a completare il task: tipicamente 'Avanza')

- **allTaskForm_claimTaskLabel**, string, se omesso viene usato un default (label del tasto che serve a richiedere il task in caso di task assegnato a più Utenti/Gruppi)
- **allTaskForm_saveTaskLabel**, string, se omesso viene usato un default (label del tasto che serve a persistere in geoweb lo stato della form del task, ma senza notificare il motore di workflow che l'utente ha completato il processo)
- **allTaskForm_restoreLabel**, string, se omesso viene usato un default (label del tasto che serve a riportare la form del task nello stato in cui era stata aperta)

Proprietà per i bottoni della form del task specifico (dove al posto del **?** va messa la **formKey** del task)

- **taskForm_?_hideDocumentation**, boolean, default false, se true nasconde il contenuto descrittivo *Documentation* del .BPMN ed il bottone che mostra il diagramma di processo
- **taskForm_?_hideCloseButton**, boolean, default false, se true nasconde bottone di chiusura dialog
- **taskForm_?_hideRestoreButton**, boolean, default false, se true nasconde bottone di annulla modifche
- **taskForm_?_hideSaveButton**, boolean, default false, se true nasconde bottone di salvataggio
- **taskForm_?_completeTaskLabel**, string, se omesso viene usato un default
- **taskForm_?_claimTaskLabel**, string, se omesso viene usato un default
- **taskForm_?_saveTaskLabel**, string, se omesso viene usato un default
- **taskForm_?_restoreLabel**, string, se omesso viene usato un default
- **taskForm_?_hideReportButton**, boolean, default false, se true nasconde l'eventuale bottone che apre la report di processo

Esempio: se la formKey vale *emission_odl_msg, taskForm_emission_odl_msg_completeTaskLabel* è un nome di proprietà valido ed il suo valore sarà la label mostrata sul tasto completeTask della tas Form con formKey = *emission_odl_msg*

Altre Configurazioni

- **codField**, String, default null, se configurato l'attributo con name codField, a prescindere che sia o meno inserito nella lista di Classe, viene aggiunto/spostato in prima posizione nella gwTaskList
- **dateField**, String, default null, se configurato l'attributo con name dateField, a prescindere che sia o meno inserito nella lista, viene aggiunto/spostato in prima posizione nella gwTaskList di Classe, viene aggiunto in prima posizione nella gwTaskList (ma dopo codField se configurato)
- **orderListByGwAttributes**, boolean, default false, se abilitato adotta il generale sistema di ordinamento delle liste di Geoweb, configurato sugli xml dei singoli widget, tramite gli appositi tag (da 4.4.12)
- **hideActivitiFilters**, boolean, default false, se abilitato nasconde dentro la UI di filtro della gwTaskList, i criteri di filtro fissi e legati alle proprietà di Activiti (taskName, assegnee, taskDocumentation, taskPriority, taskCreated, taskDueDate) (da 4.4.12)

- **startProcessInstanceLabel**, string, se omesso viene usato un default (label tasto avanza del form preliminare all'avvio del processo)
- **allTaskForm_startProcessInstanceLabel**, string, se omesso viene usato un default (label tasto avanza del form preliminare all'avvio del processo) **@Deprecated, funziona nel .BPMN, rimpiazzato da startProcessInstanceLabel nel configJSON**
- **processEndedNotificationTitle**, string, se omesso viene usato un default (titolo del dialog che notifica la fine di tutte le attività di un utente su una data istanza di processo)
- **processEndedNotificationMessage**, string, se omesso viene usato un default (messaggio del dialog che notifica la fine di tutte le attività di un utente su una data istanza di processo)

****Note****

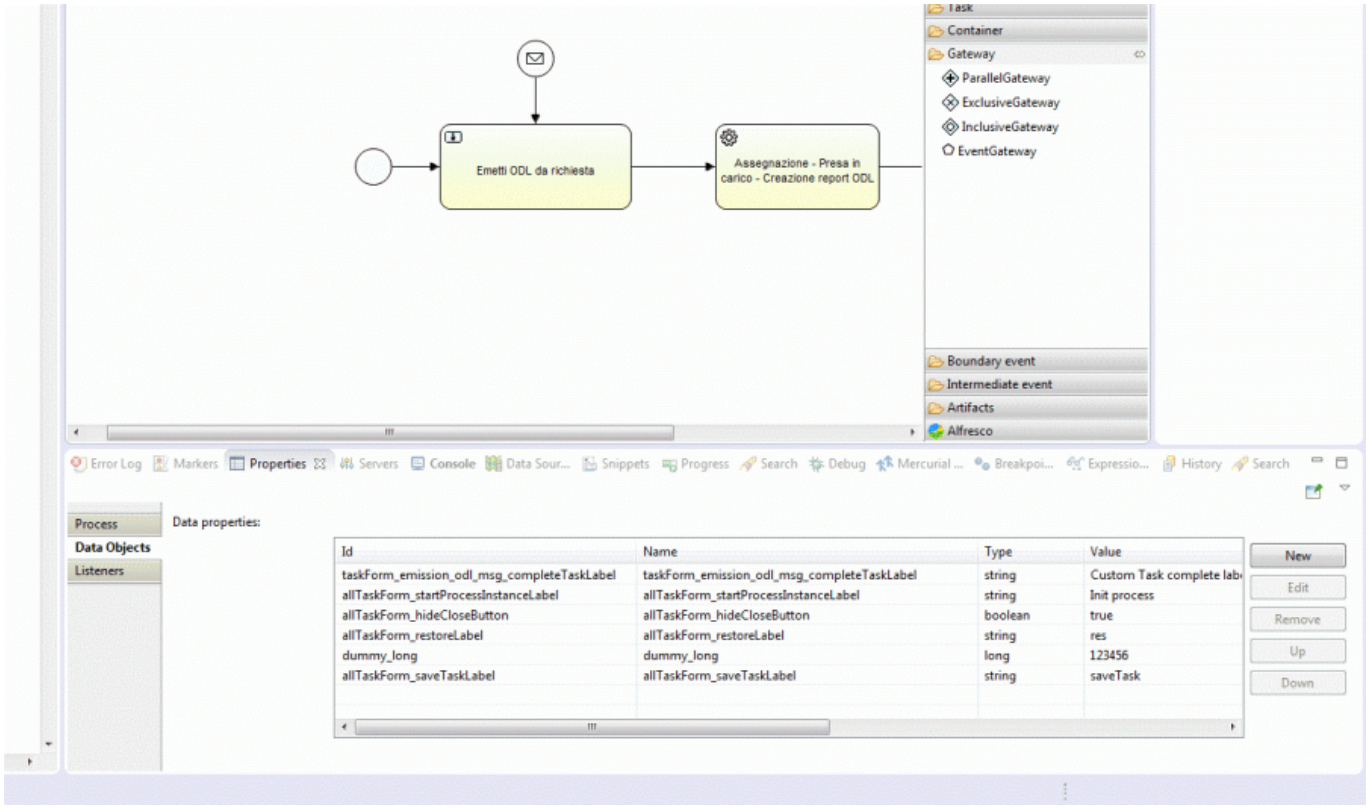
Le proprietà del task specifico hanno priorità su quelle globali.

Le proprietà delle serie allTaskForm_*, taskForm_* e processEndedNotification* possono essere impostate anche nel file .BPMN di processo.

Questo era il modo con cui venivano impostate prima dell'aggiunta del metadato **configJSON**.

Tale modalità di configurazione, seppur supportata è da considerarsi **DEPRECATA**.

Con il diagramma di processo aperto in *Eclipse*, cliccando su una zona bianca, il tab '*Properties*' mostra le proprietà generali del processo. Nella sezione '*Data Object*' è possibile gestire variabili di processo custom.



Ad essere valutato ad ogni proprietà è l'attributo name. (per comodità mettiamo id=name)

Il type è importante ed è specifico della proprietà che vogliamo usare.

Una volta modificato il file .BPMN, esso va ricaricato in Geoweb admin, per poter vedere le modifiche.

Note:

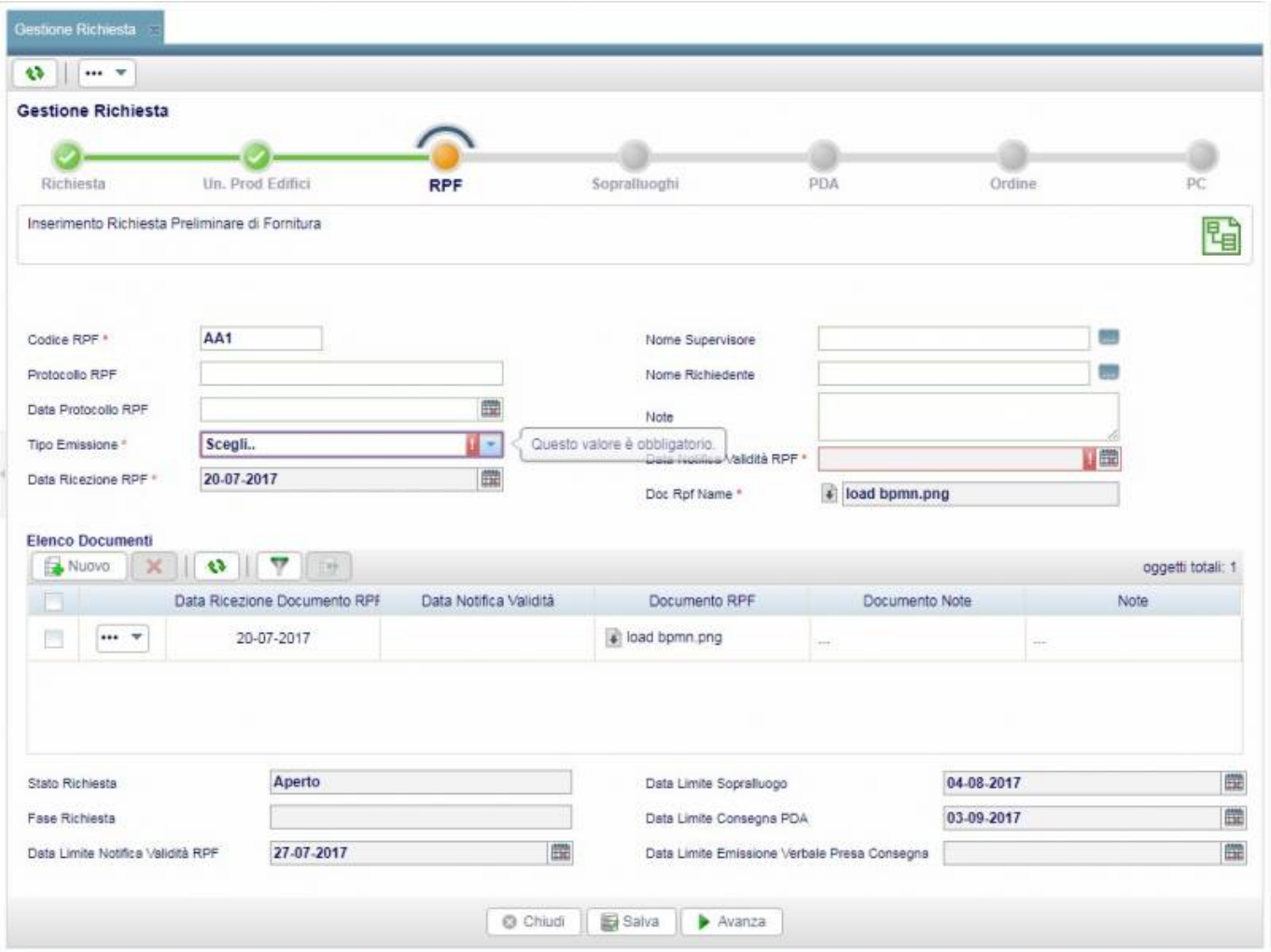
- i processi che erano stati avviati con le precedenti definizioni di processo, non saranno effetti dalle nuove modifiche
- le variabili definite in Data Objects, saranno considerate da Activiti a tutti gli effetti variabili di processo, e saranno quindi accessibili e modificabili dal codice contenuto nei vari .groovy delle TaskService

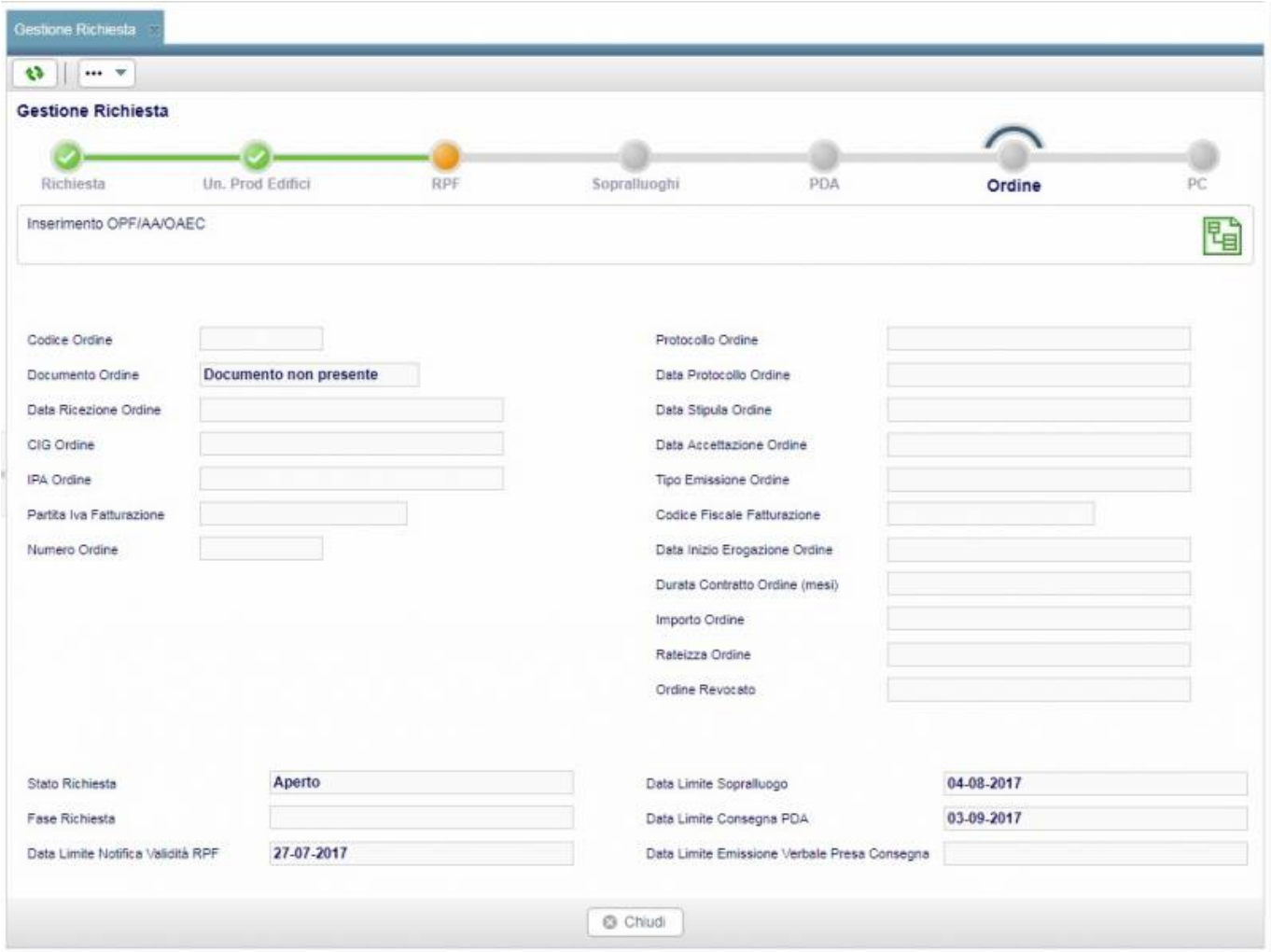
Procedure

Nell'ambito del workflow, una ***procedura*** è un **particolare processo** dove il flusso di lavoro è lineare, costituito da una serie di ***fasi*** predefinite che si susseguono in ordine temporale. Una procedura comporta che il flusso di un workflow possa essere diviso in step,

Impostando tipo processo 'PROCEDURE' ad un gwProcess comporta:

- all'utente viene presentato un dettaglio task che mostra la sequenza delle fasi del flusso del processo, evidenziando quella corrente. La fase corrente è preceduta da tutte le eventuali fasi già completate, e seguita dalle fasi relative agli step successivi. Al click sulla singola fase viene mostrata la form corrispondente, in edit per la fase corrente, in visualizzazione per le fasi precedenti e successive.
- Il dettaglio task presenta in primo piano l'informazione del codice della procedura, se è configurato il campo 'codeField'
- nella schede di tipo gwClassList, se sono configurati i campi 'codeField', 'dateField', essi vengono posti in determinate posizioni, a prescindere dal loro posizionamento di base nella lista della gwClass di processo


Gestione Richiesta
Gestione Richiesta
Richiesta
Un. Prod Edifici
RPF
Sopralluoghi
PDA
Ordine
PC
Inserimento Richiesta Preliminare di Fornitura
Codice RPF *
AA1
Protocollo RPF
Data Protocollo RPF
Tipo Emissione *
Scegli..
Questo valore è obbligatorio.
Data Ricezione RPF *
20-07-2017
Nome Supervisore
Nome Richiedente
Note
Data Notifica Validità RPF *
Doc Rpf Name *
load bpmn.png
Elenco Documenti
Nuovo
oggetti totali: 1
Data Ricezione Documento RPF
Data Notifica Validità
Documento RPF
Documento Note
Note
20-07-2017
load bpmn.png
Stato Richiesta
Aperto
Fase Richiesta
Data Limite Notifica Validità RPF
27-07-2017
Data Limite Sopralluogo
04-08-2017
Data Limite Consegna PDA
03-09-2017
Data Limite Emissione Verbale Presa Consegna
Chiudi
Salva
Avanza



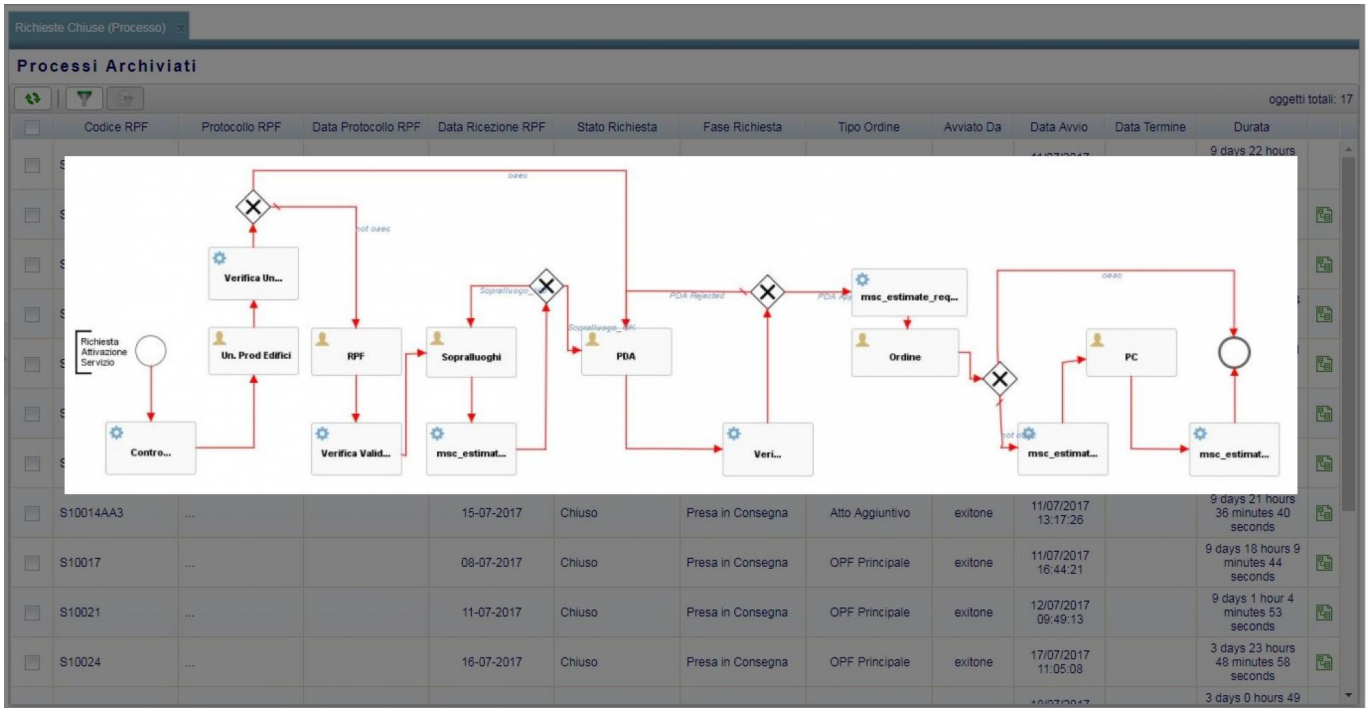
Wizard

Questa modalità va usata tipicamente per processi mono-utente che non hanno un formalismo (dato da un protocollo, etc..) dietro. In questa modalità su ogni scheda di ogni attività del processo saranno disponibili i soli tasti 'Close' (,'Abort' ? eventualmente da decidere), 'Advance', mentre verranno sempre esclusi 'Restore', 'Save', 'Claim'. Continua ad essere supportato il meccanismo che permette di nascondere il tasto 'Close' e modificare la label di 'Advance' sia per tutti i task o per i soli taskcon specifici taskForm.

Tipi di schede

gwArchivedProcessList

Scheda per la visualizzazione delle informazioni delle istanze di processo terminate relative al processo con processDefinitionKey (uguale al parametro gwProcessName). Al click sulle righe della lista verranno aperte le classiche form del gwClassDetail. In caso di gwProcess di Tipo PROCEDURE , verrà invece aperta l'apposita interfaccia riepilogativa. Al click sull'ultima colonna si aprirà il diagramma del flusso seguito dalla particolare istanza di processo.



Si possono imporre particolari set di filtri. Si possono imporre particolari set di filtri statici. Il parametro *openingDetailMode* dentro *options* influisce sul tipo di contenitore che verrà utilizzato per aprire le form. I possibili valori sono: '*floatingPane*', '*dialog*' e '*tab*'.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

gwClassProcesses

In Geoweb è presente una scheda per la visualizzazione e la gestione di task e istanze di processo.

Questa scheda racchiude tanti componenti e ne è scoraggiato l'uso per le implementazioni in quanto non risulterebbe chiara per l'utente. Tutte le schede visibili qui sono apribili separatamente tramite API js o tramite appositi leafItem.

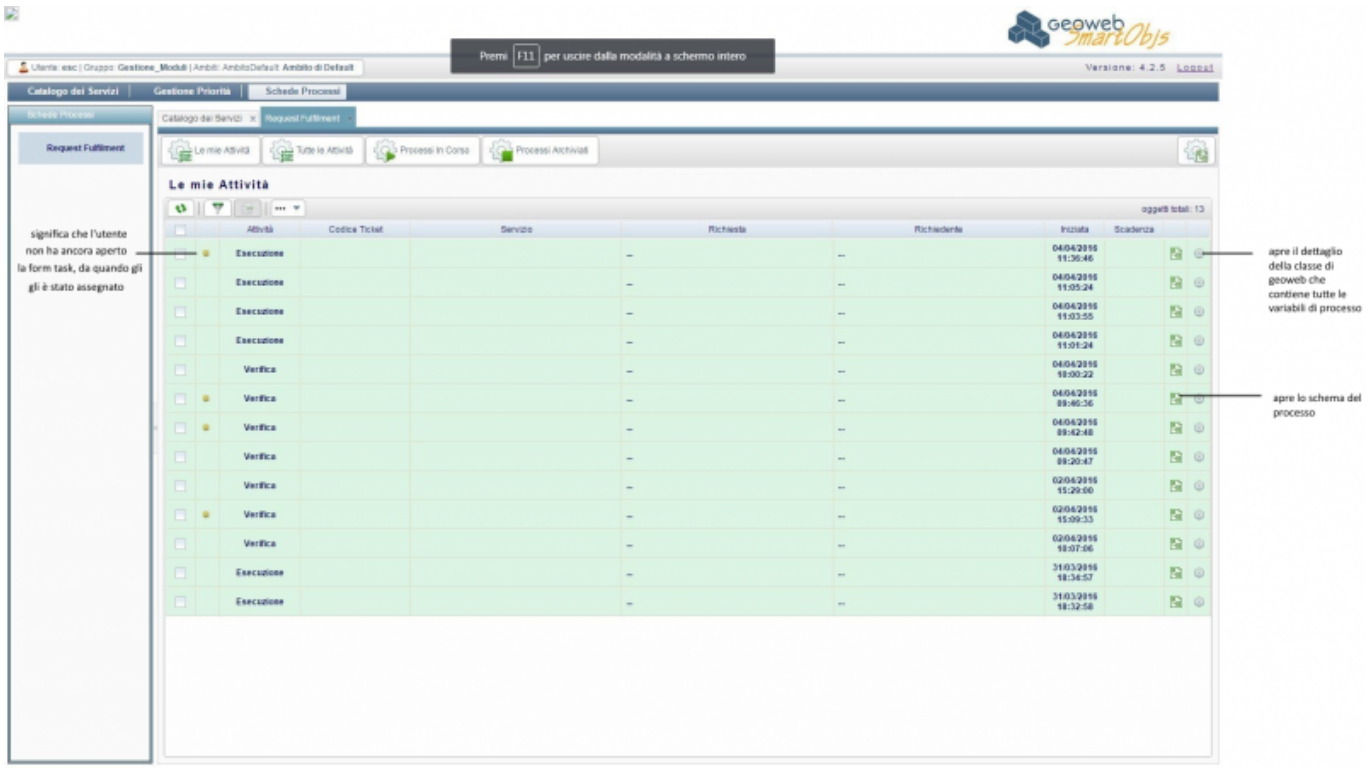
In questa scheda fra le altre cose, è possibile:

- avviare una nuova istanza di processo
- vedere tutte le proprie attività
- completare le proprie attività
- vedere le attività degli altri utenti coinvolti nel processo (in sola lettura).
- richiedere per se (Claim) lo svolgimento di tutti quei Task che non sono stati ancora assegnati ma prevedono una lista di utenti candidati od una lista di gruppi candidati a svolgere quella attività (fra i quale si è presenti).
- consultare la lista dei processi in corso
- consultare la lista dei processi terminati (in sola lettura)
- consultare un'immagine grafica del modello .bpmn del processo
- consultare un'immagine grafica del modello .bpmn per i singoli task, processi in corso, e processi terminati

Liste Attività ('Mie' e 'Tutte')

I record di tali liste hanno:

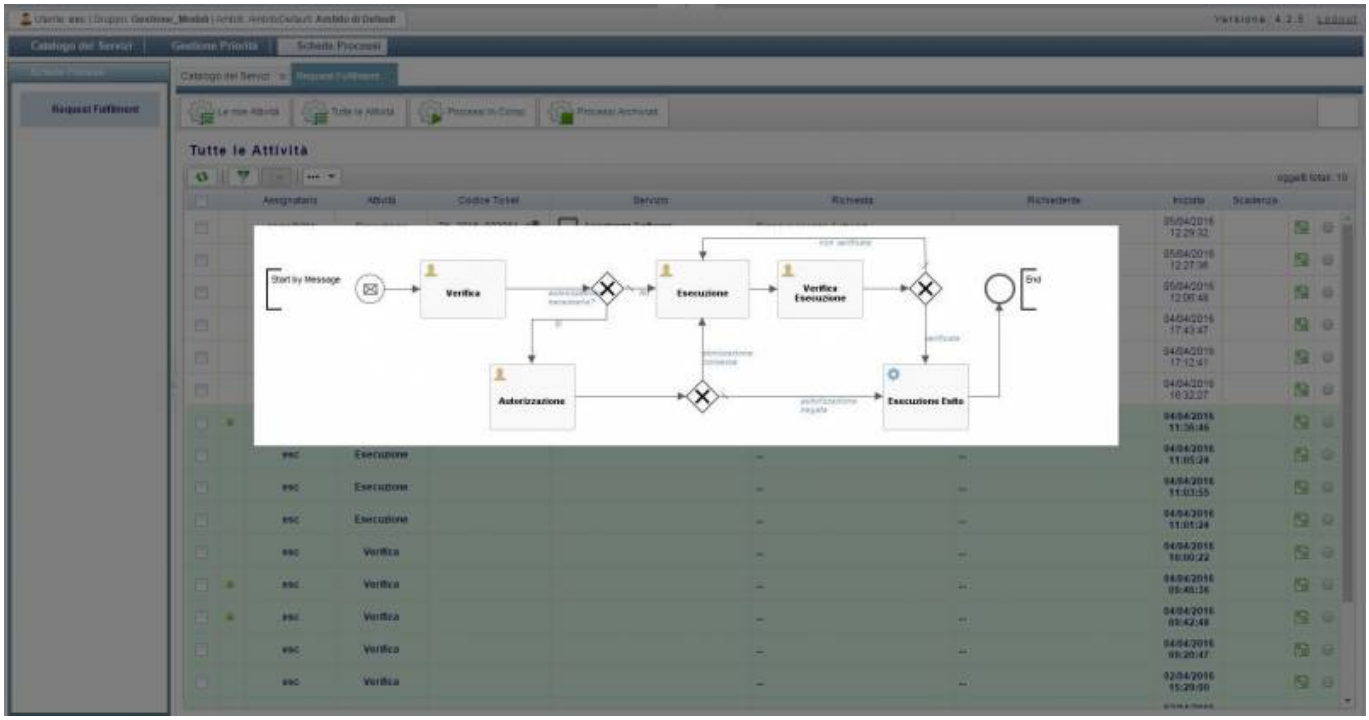
- Sfondo verde: se il task è assegnato da me.
- Sfondo Giallo: se il task è assegnato a una specifica lista di utenti, o ad una specifica lista di gruppi, nei quali è invcluso l'utente corrente, od il gruppo dell'utente corrente.
- Sfondo Bianco: se il task è assegnato ad un altro utente oppure non l'utente corrente non risulta nella lista degli utenti candidati all'esecuzione.



Se è presente il marker nella prima colonna significa che la form del task non è mai stata aperta dall'utente dopo che gli è stata assegnata tale attività dal motore di workflow.

In queste schede ci sono colonne fisse (nome attività, assegnatario, data inizio, scadenza, tasto apertura diagramma, tasto apertura record istanza processo) e colonne variabili (individuate in base agli attributi della classe di processo messi in lista)

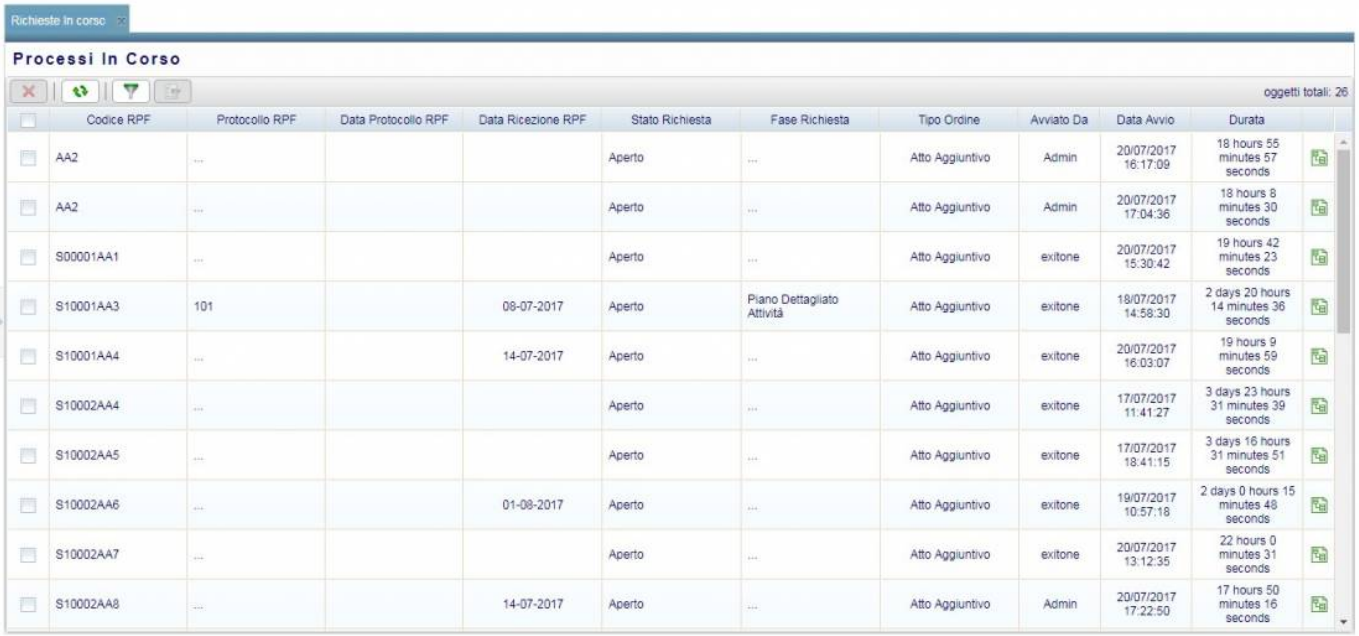
Qui sotto un esempio di diagramma di processo.



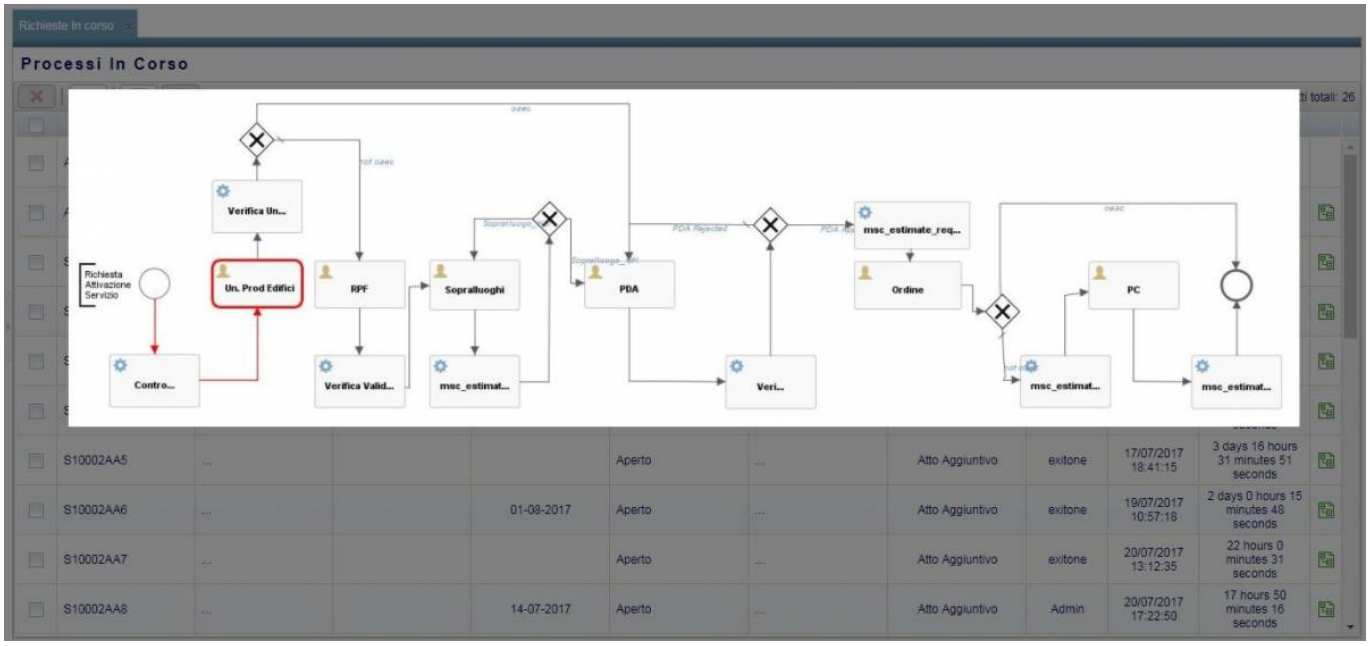
Liste Istanze di processo ('In Corso' e 'Archiviate')TODO

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

gwRunningProcessList



Scheda per la visualizzazione delle informazioni delle **istanze di processo attive** relative al processo con processDefinitionKey (uguale al parametro gwProcessName). Al click sulle righe della lista verranno aperte le classiche form del gwClassDetail. In caso di gwProcess di Tipo PROCEDURE verrà invece aperta l'apposita interfaccia riepilogativa. Al click sull'ultima colonna si aprirà il diagramma del flusso seguito dalla particolare istanza di processo.



Si possono imporre particolari set di filtri. Si possono imporre particolari set di filtri statici. Il parametro *openingDetailMode* dentro options influisce sul tipo di contenitore che verrà utilizzato per aprire le form. I possibili valori sono: '*floatingPane*', '*dialog*' e '*tab*'.

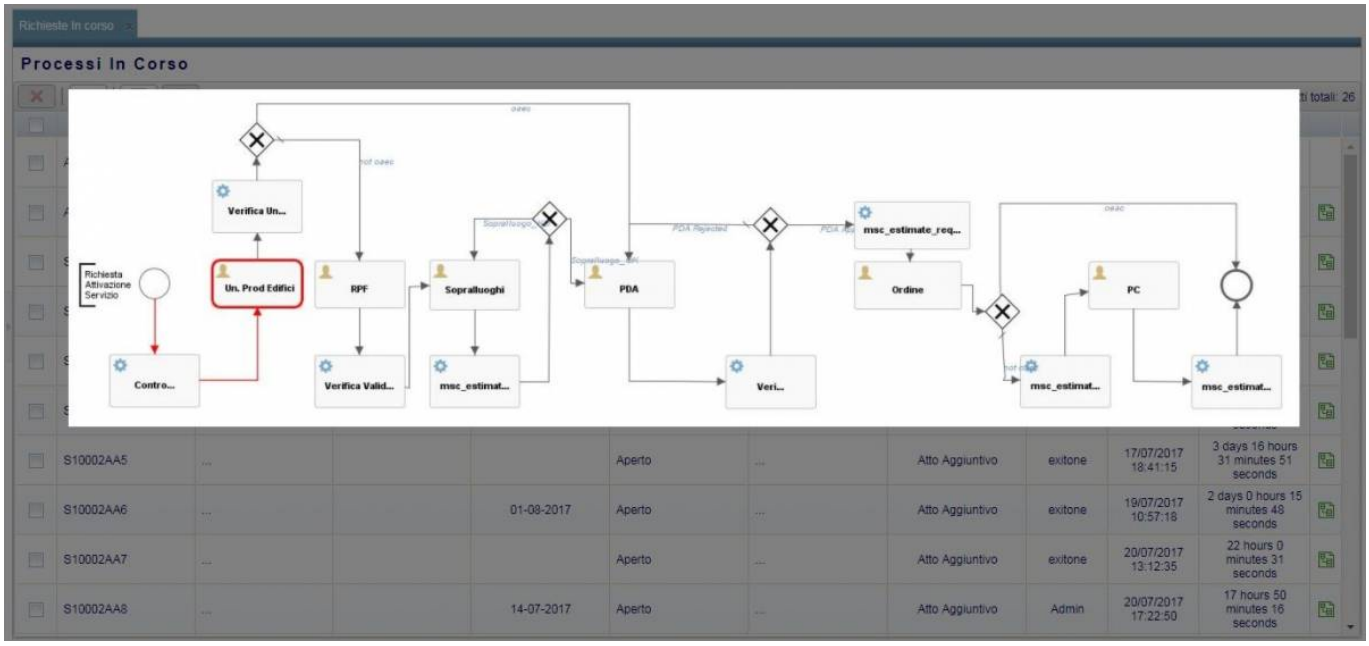
Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

gwArchivedProcessList

Scheda per la visualizzazione delle informazioni delle **istanze di processo terminate** relative al processo con processDefinitionKey (uguale al parametro gwProcessName). Al click sulle righe della lista verranno aperte le classiche form del gwClassDetail. In caso di gwProcess di Tipo PROCEDURE verrà invece aperta l'apposita interfaccia riepilogativa. Al click sull'ultima colonna si aprirà il diagramma del flusso seguito dalla particolare istanza di processo.



Si possono imporre particolari set di filtri. Si possono imporre particolari set di filtri statici. Il parametro *openingDetailMode* dentro options influisce sul tipo di contenitore che verrà utilizzato per aprire le form. I possibili valori sono: '*floatingPane*', '*dialog*' e '*tab*'.

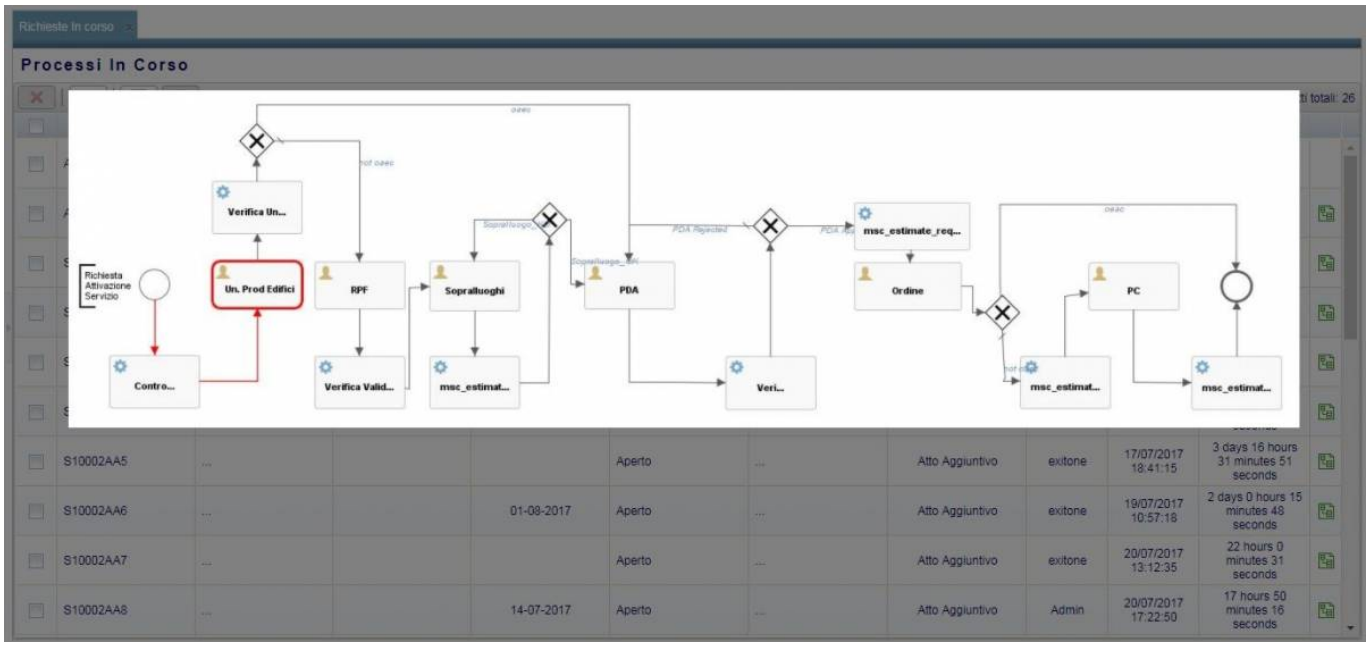
Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

gwSuspendedProcessList

Scheda per la visualizzazione delle informazioni delle **istanze di processo sospese** (non terminate, e non attive, secondo l'accezione Activiti) relative al processo con processDefinitionKey (uguale al parametro gwProcessName). Al click sulle righe della lista verranno aperte le classiche form del gwClassDetail. In caso di gwProcess di Tipo PROCEDUREverrà invece aperta l'apposita interfaccia riepilogativa. Al click sull'ultima colonna si aprirà il diagramma del flusso seguito dalla particolare istanza di processo.



Si possono imporre particolari set di filtri. Si possono imporre particolari set di filtri statici. Il parametro *openingDetailMode* dentro options influisce sul tipo di contenitore che verrà utilizzato per aprire le form. I possibili valori sono: '*floatingPane*', '*dialog*' e '*tab*'.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

gwTaskDetail

Questa scheda permette di aprire una form contenente le informazioni relative ad un task. Questa form a seconda dei casi verrà aperta:

- in edit: se il task è assegnato all'utente corrente
- in visualizzazione, con la possibilità di richiederlo: se il task non è stato ancora assegnato, e l'utente corrente può richiedere l'esecuzione del task.
- in visualizzazione: se il task è assegnato ad un altro utente oppure l'utente corrente non può

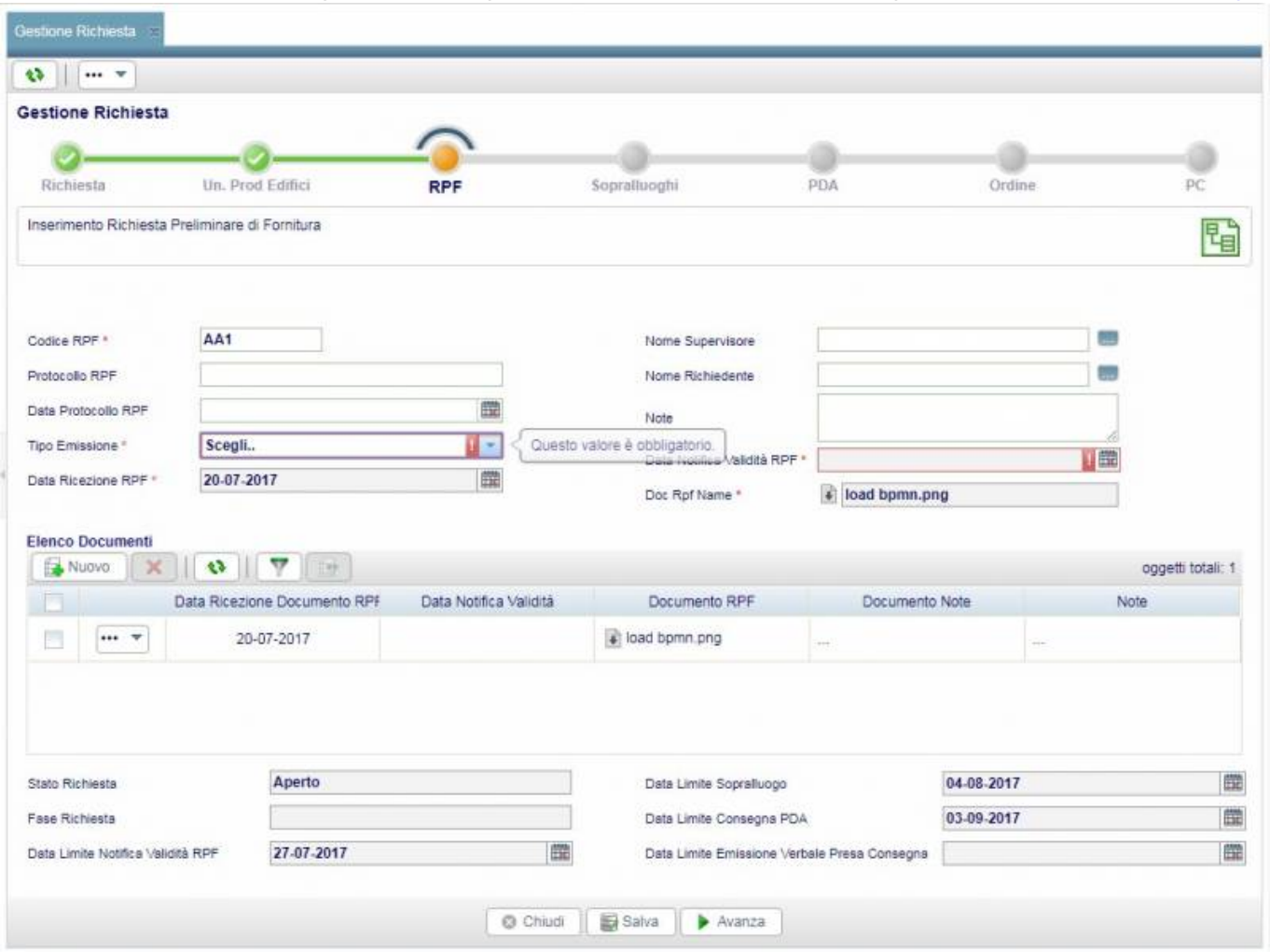
richiederlo in quanto non è nella lista degli utenti candidati ad eseguire il task

Il claim task viene fatta attraverso il tasto '*Richiedi*'.

Il completamento del task viene fatto attraverso il tasto '*Avanza*'.

Il gwTaskDetail, oltre che in Tab, può essere aperto anche su Floating Pane e su Dialog.

Se il task è relativo ad un processo di tipo PROCEDURE, la scheda verrà presentata come descritto qui



gwTaskList

Scheda per la visualizzazione dei task relativi alle istanze di processo generate dalla definizione di processo con processDefinitionKey (uguale al parametro gwProcessName). Le righe vengono visualizzate con colori differenti in base all'assegnatario del task:

- verdi: task assegnato all'utente corrente
- gialle: task non assegnato a nessuno e richiedibile da parte dell'utente corrente
- bianco: task assegnato ad un altro utente, o comunque sia non richiedibile dall'utente corrente

Si possono eventualmente visualizzare solo i task assegnati all'utente corrente (showOnlyUserTasks posto a true).

Si possono imporre particolari set di filtri statici.

Al click sulle righe della lista verranno aperte form come all'esecuzione di openGwTaskDetail(). Il parametro *openingDetailMode* dentro options influisce sul tipo di contenitore che verrà utilizzato per aprire le form. I possibili valori sono: '*floatingPane*', '*dialog*' e '*tab*'.

Parametri:

- gwProcessName, String
- title, String, optional
- parametersMap, Object, optional
  - openingDetailMode, String, optional evaluated only if handleNextStep==true
  - showOnlyUserTasks, Boolean, optional, default false
  - hideTaskCreatedDateColumn, Boolean, optional, default true
  - hideTaskDueDateColumn, Boolean, optional, default true
  - hideTaskPriorityColumn, Boolean, optional, default true
  - hideProcessDiagramColumn, Boolean, optional, default false
  - hideProcessDetailColumn, Boolean, optional, default false (default true when gwProcess is a PROCEDURE)
  - staticFilters, Object[], optional
  - insertIndex, Integer, optional

Questa scheda può essere aperta tramite API JS o tramite apposito leafItem.

Tipologie di LeafItem

LeafItem gwClassProcesses

Apre una scheda di tipo gwClassProcesses.

Parametri:

- className, String, required
- gwProcessName, String, required

Esempio:

```
  <leafItem name="request_fulfilment" label="Request Fulfilment"
image="process.png" type="gwClassProcesses">
     <parameter name="className" value="rqm_wpi_request_fulfilment"
hideToClient="false"></parameter>
     <parameter name="gwProcessName" value="rqm_wpi_request_fulfilment"
hideToClient="false"></parameter>
  </leafItem>
```

LeafItem gwArchivedProcessList

Apre una scheda di tipo gwArchivedProcessList.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Esempio:

```
<leafItem name="list_rpf_closed" label="Richieste Chiuse" image="icon.png"
type="gwArchivedProcessList">
    <parameter name="gwProcessName" value="msc_estimate_request"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
    <parameter name="staticFilters" value="[{condition:'AND', columnName:
'status', operator:'=', filterType:'STRING', value: ['OPN']}]"
hideToClient="false"></parameter>
</leafItem>
```

LeafItem gwRunningProcessList

Apre una scheda di tipo gwRunningProcessList.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Esempio:

```
<leafItem name="list_rpf_closed" label="Richieste Chiuse" image="icon.png"
type="gwRunningProcessList">
    <parameter name="gwProcessName" value="msc_estimate_request"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
    <parameter name="staticFilters" value="[{condition:'AND', columnName:
'status', operator:'=', filterType:'STRING', value: ['OPN']}]"
hideToClient="false"></parameter>
</leafItem>
```

LeafItem gwSuspendedProcessList

Apre una scheda di tipo gwSuspendedProcessList.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- filters, Object[], optional
- staticFilters, Object[], optional

Esempio:

```
<leafItem name="list_rpf_closed" label="Richieste Chiuse" image="icon.png"
type="gwSuspendedProcessList">
    <parameter name="gwProcessName" value="msc_estimate_request"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
    <parameter name="staticFilters" value="[{condition:'AND', columnName:
'status', operator:'=', filterType:'STRING', value: ['OPN']}]"
hideToClient="false"></parameter>
</leafItem>
```

LeafItem gwStartProcessInstanceAction

Azione eseguibile da leafItem che al click apre una scheda che permette di avviare un'istanza di processo, mostrando la form di raccolta dati iniziale.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- startFormKey , String, optional, default null. If omitted is retrieved

Esempio:

```
<leafItem name="list_rpf" label="Richieste in Fase RPF" image="icon.png"
type="gwStartProcessInstanceAction">
    <parameter name="gwProcessName" value="msc_estimate_request"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
    <parameter name="startFormKey" value="start_form"
hideToClient="false"></parameter>
</leafItem>''
```

LeafItem gwStartProcessInstanceByMessageAction

Azione eseguibile da leafItem che al click avvia un'istanza di processo con un set di dati iniziale. Quando handleNextStep vale true, apre eventualmente anche il primo task disponibile eseguibile o richiedibile dall'utente corrente.

Parametri:

- message, String, required
- processVariables , Object, optional.
- handleNextStep, Boolean, optional, default true.
- openingDetailMode, String, optional evaluated only if handleNextStep==true

Esempio:

```
<leafItem name="list_rpf" label="Richieste in Fase RPF" image="icon.png"
type="gwStartProcessInstanceAction">
    <parameter name="message" value="message"
hideToClient="false"></parameter>
    <parameter name="processVariables " value="{var1: 'var1', var2: 2}"
hideToClient="false"></parameter>
    <parameter name="handleNextStep" value="true"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
```

```
</leafItem>
```

LeafItem gwTaskList

Apre una scheda di tipo gwTaskList.

Parametri:

- gwProcessName, String, required
- openingDetailMode, String, optional evaluated only if handleNextStep==true
- showOnlyUserTasks, Boolean, optional, default false
- hideTaskCreatedDateColumn, Boolean, optional, default true
- hideTaskDueDateColumn, Boolean, optional, default true
- hideTaskPriorityColumn, Boolean, optional, default true
- hideProcessDiagramColumn, Boolean, optional, default false
- hideProcessDetailColumn, Boolean, optional, default false (default true when gwProcess is a PROCEDURE)
- staticFilters, Object[], optional

Esempio:

```
<leafItem name="list_rpf" label="Richieste in Fase RPF" image="icon.png"
type="gwTaskList">
    <parameter name="gwProcessName" value="msc_estimate_request"
hideToClient="false"></parameter>
    <parameter name="openingDetailMode" value="tab"
hideToClient="false"></parameter>
    <parameter name="showOnlyUserTasks" value="false"
hideToClient="false"></parameter>
    <parameter name="staticFilters" value="[{condition:'AND', columnName:
'phase_estimate', operator:'=', filterType:'STRING', value: ['RPF']}]"
hideToClient="false"></parameter>
</leafItem>
```

Workflow javascript API

Sezione dedicata qui

Workflow Java API

Sezione dedicata qui