Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

1

WORKFLOW EDITOR PER IL MIDDLEWARE MUSA

A. Cavaleri L. Sabatucci

Rapporto Tecnico N.: ***Data:***

RT-ICAR-PA-15-05 **Novembre 2015**

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Viale delle Scienze edificio 11, 90128 Palermo.

Realizzazione di un Editor di Workflow Web Based

Antonella Cavaleri and Luca Sabatucci ICAR-CNR, Palermo
Email: {a.cavaleri,sabatucci}@pa.icar.cnr.it

ABSTRACT

Il presente lavoro presenta l'attività di sviluppo di un editor di workflow realizzato per la fruizione remota attraverso l'utilizzo di tecnologie web. A seguito di uno studio dello stato dell'arte dei principali tool commerciali ed open source esistenti sono stati definite le funzionalità di base ed avanzate delle quali dovrebbe essere dotato tale tool. Quindi è stato possibile avviare la fase di progettazione e la fase di sviluppo dell'editor di workflow "web based".

INTRODUZIONE

All'interno di un sistema informativo i principali elementi costitutivi che possono essere individuati sono senza dubbio i Processi di Business che rappresentano l'insieme delle singole attività che è necessario correlare opportunamente per raggiungere dei ben precisi obiettivi. Infatti definire un processo di business equivale a illustrare e descrivere il workflow che lo rappresenta al fine di poter individuare in modo precisa e non ambiguo quali sono gli elementi costitutivi che lo compongono, in modo che ciò sia ben chiaro e utile a tutte le figure coinvolte nel processo stesso. Attraverso la corretta definizione dei processi di business di un sistema informativo vengono quindi definiti chiaramente quali sono i requisisti che il sistema stesso deve possedere. In letteratura si enfatizza infatti molto in merito alla necessità di focalizzare l'attenzione sul perfezionamento dei singoli processi di business al fine di raggiungere un miglioramento continuo delle prestazioni aziendali.

Le attività che costituiscono il business process management possono essere raggruppate in 5 fasi principali: Design, Modeling, Execution, Monitoring, Optimization.
Attraverso la modellazione del processo necessita dell' utilizzo di modelli di processo che siano dotati di un soddisfacente grado di rigore nella definizione del loro significato in modo da renderne univoca l'interpretazione considerato che attraverso la modellazione si realizza spesso la comunicazione e la condivisione delle informazioni relative al processo in considerazione tra i vari attori coinvolti. Per tale motivo gli standard esistenti per la modellazione dei business process hanno un obiettivo di alto livello puramente descrittivo che favorisce la corretta interpretazione e comunicazione human-to –human ma anche e soprattutto una precisa esecuzione in quanto favoriscono una rappresentazione dettagliata e rigorosa nella comunicazione human-to-computer.

La definizione della struttura di un business process necessita quindi della modellazione si degli elementi costitutivi che lo compongono, nonché delle loro reciproche relazioni ed interconnessioni ma soprattutto di tutte le vari tipologie di entità coinvolte nel processo.
La "Business Process Management Initiative" e l'"Object Management Group" (http://www.omg.org), hanno sviluppato la notazione BPMN (Business Process Management Notation) che si attualmente affermato come il principale standard per la modellazione dei processi aziendali. Tale notazione si pone come obiettivo principale quello di fornire uno modello uniforme di rappresentazione efficace, facile da utilizzare e da comprendere da parte degli utenti di business interessati al problema della modellazione, progettazione ed eventuale informatizzazione dei processi aziendali. Infatti attraverso l'uso della Business

Process Management Notation è possibile costruire dei diagrammi di processo (BPD – Business Process Diagram) che rappresentano in pratica dei grafi o reti costituiti da "oggetti" rappresentati dalle attività di processo, collegati da flussi di controllo che definiscono la relazione logica, le dipendenze e l'ordine di esecuzione delle attività stesse. La versione 1.0 dello standard è stata rilasciata nel 2004, mentre la versione 2.0 è oggi quella ufficialmente utilizzata e la cui release risale alla fine del 2010.
Grazie al BPMN è possibile spesso realizzare una stretta integrazione con i sistemi di sviluppo software. Negli ultimi anni infatti numerose sono state le applicazioni sviluppate e rese disponibili alle aziende che permettono al modellista di un business process di rappresentare i dettagli di un processo di business tramite BPMN e tradurre in seguito tale modello in un programma software che sia in grado di gestire l'esecuzione del processo stesso.
Un BPD(Business Process Diagram) si costruisce utilizzando alcuni elementi grafici la cui caratteristica principale è la semplicità interpretativa per gli analisti di business e dei processi organizzativi.
Lo standard BPMN definisce alcuni elementi grafici di base, generalmente sufficienti per modellizzare una vasta casistica di processi, e ai quali comunque si possono aggiungere integrazioni ed elementi addizionali per dare maggiore efficacia rappresentativa nei casi di processi molto complessi ma senza modificare l'impostazione base della notazione usata.
Le quattro categorie fondamentali di elementi grafici sono le seguenti:
- Elementi di flusso (flow object)
- Connettori (connecting object)
- Corsie (swimlane)
- Artefatti (artifact)

Tali premesse fanno facilmente intuire come sia quindi necessario per le aziende, per le quali i processi sono il motore di realizzazione delle loro attività, avere degli strumenti che permettano al definizione di un editor di workflow basato sullo standard BPMN, tuttavia in un epoca in cui internet è divenuto pervasivo la principale esigenza diviene quella di realizzare tale sistema capace di inter-operare in un ambiente web based. Tale caratteristica rende quindi l'editor BPMN uno strumento innovativo rivolto e usufruibile da diversi tipi di utenti i quali accedendo tramite un comune browser all'interfaccia dell'Editor possono progettare in maniera visuale il proprio workflow che può essere poi convertito su richiesta in codice xml compatibile alle specifiche OGM per BPMN

INDICE

ANALISI COMPARATIVA EDITOR DI WORKFLOW WEB BASED

Al fine di poter meglio definire le funzionalità e le caratteristiche implementative si è resa utile una fase iniziale di analisi degli editor web già esistenti in modo da metterne in rilievo le

loro potenzialità, nonché i limiti. I principali tools con caratteristiche affini agli obiettivi realizzativi prefissati verranno analizzati dettagliatamente nel corso di questo paragrafo. Nella tabella sottostante so no stati raggruppati i principale sistemi presi in considerazione per i quali sono stati effettuati delle comparazioni che riguardano le principali caratteristiche di interesse per individuare quale tra di essi abbia le migliori caratteristiche che permettano di sfruttarli per la realizzazione delle funzionalità di cui debba essere dotato il sistema che ci si prefigge di realizzare.

EDITOR	LICENZA	TECNOLOGIE	DOCUMENTAZIONE
LUCIDCHART	COMMERCIALE	HTML5-JAVASCRIPT	FORUM-BLOG
JOINTJS	OPEN SOUCE(LIBRERIA)	HTML 5 - JAVASCRIPT	TUTORIAL-FORUM
DIAGRAMO	OPEN SOURCE	JAVASCRIPT-PHP	BLOG- TUTORIAL-JSDOC
DRAW IO	COMMERCIALE	JGRAPHX(libreria open source)	BLOG- USER MANUAL-GOOGLE COMMUNITY
Gliffy	COMMERCIALE	HTML 5	USER MANUAL-BLOG
Cacoo	COMMERCIALE	FLASH PLAYER	BLOG
BPMN.IO	OPEN SOURCE	JAVASCRIPT	BLOG- FORUM
SIGNAVIO	OPEN SOURCE	JAVASCRIPT	BLOG- USER MANUAL-GOOGLE COMMUNITY
JALAVA	OPEN SOURCE	JAVA	BLOG-

Figura 1 -Tabella comparativa dei tools analizzati

A. DIAGRAMO

Il tool Diagramo permette di creare diagrammi on line è free ed open source, è basato su Html5, permette di personalizzare l'applicazione utilizzando una licenza di tipo GPL, è possibile creare un istallazione per il proprio server. Le principali tecnologie utilizzate sono javascript e php. Gli unici tipi di diagrammi attualmente supportati dall'editor sono quelli UML e macchine a stati, da agosto 2014 si è passati da una licenza GPL ad una licenza di tipo Apache 2.0. Le funzionalità messe a disposizione dell'utente sono molto limitate e comprendono pochi elementi raggruppati in delle limitate categorie. Permette di condividere i diagrammi con un team ed è possibile esportare i diagrammi n SVG, Gf o JPEG. È a disposizione degli utenti un blog attivo all'anno corrente. La versione trial premette di testare le funzionalità di base anche senza registrazione , necessaria invece per avere abilitare ulteriori funzionalità quali il salvataggio , la condivisione e la modifica di diagrammi precedentemente salvati. Gli elementi possono essere inseriti agevolmente nel proprio diagramma e connessi usando varie tipologie di collegamenti messi a disposizione, inoltre è possibile inserire una scritta personalizzata, nonché impostare la dimensione e la collocazione di essi. I diagrammi realizzati vengono salvati all'interno del servizio e sono consultabili dall'utente che li ha generati per successive modifiche, esiste la possibilità di condividerli con altri ma non è invece supportata la collaborazione sincrona tra vari utenti.

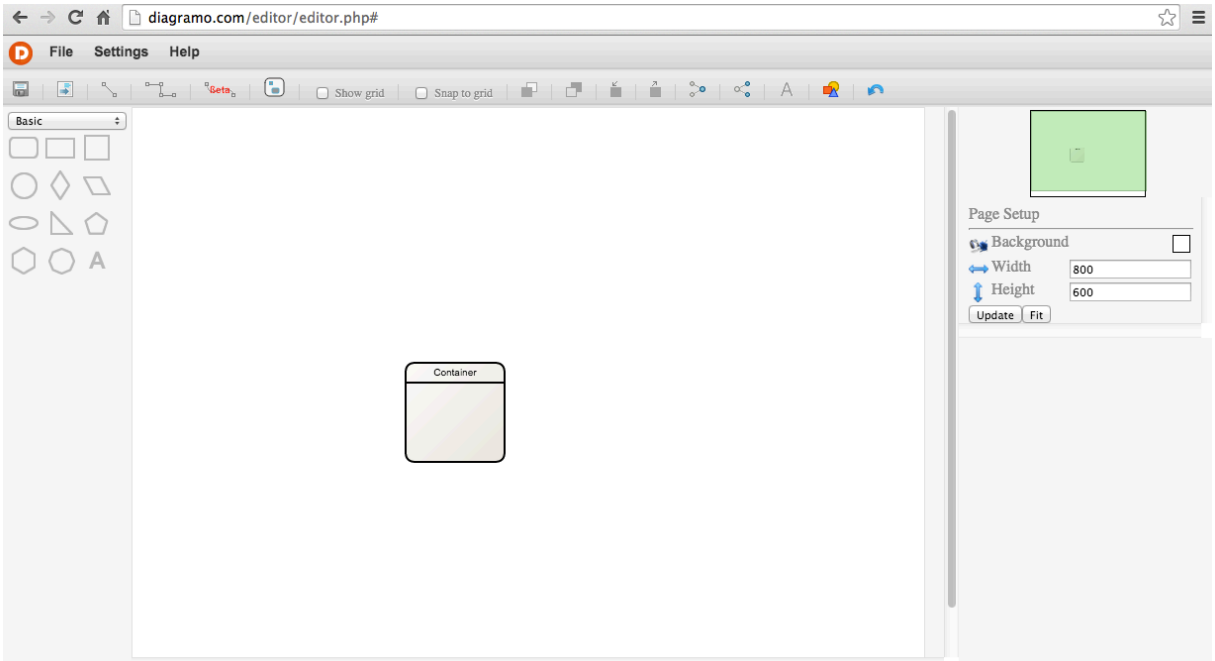


Figura 2 - Diagramo Editor [1]

B. LUCIDCHART

LucidChart è un software web-based per l'editing di diagrammi che consente agli utenti la definizione di svariate tipologie di grafici visuali (organigrammi, wireframe di siti web, disegni UML, mappe mentali, prototipi software, e molti altri tipi di diagrammi) nonché , la condivisione e lo sviluppo collaborativo tra gli utenti che possono cooperare nella stesura degli stessi in tempo reale e online. Il servizio web è utilizzabile anche in modalità free permette la definizione di diagrammi di flusso, mappe mentali mockups, wireframe e altre tipologie di grafici tutti online, mane viene inibita la condivisione È necessaria una registrazione al servizio web per poterne accedere alle funzionalità. Il tool mette a disposizione dell'utente per ogni tipologia di grafico che si intende realizzare le forme o oggetti appositi da inserirvi, vengono inoltre forniti dei template di esempio modificabili che semplificano la creazione di diagrammi.

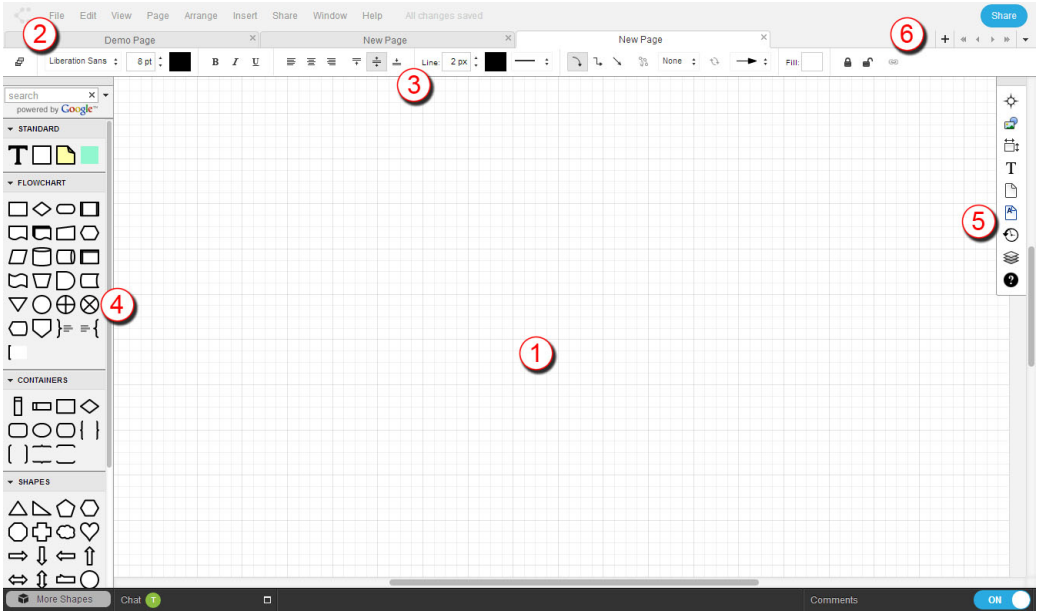


Figura 3- Editor LucidChart [2]

L'utilizzo è abbastanza intuitivo , a partire dalla scelta della tipologia di grafico che si intende realizzare attraverso il pulsante "Start Draving" si accede all'interfaccia di definizione del diagramma messa a disposizione dall'editor. Come si evince dalla Figura 3 l'interfaccia web mostrata dall'editor è possibile individuare le parti principali che la compongono:

1→ Area di disegno: è possibile personalizzarne le dimensioni , orientamento e margini per adattarla alle esigenze del diagramma in essa definito; attraverso il clik con il tasto destro è possibile spostare l'area di disegno trascinandola a destra o a sinistra.

2→ Barra del Menu: posizionata sotto il titolo del documento, fornisce l'accesso alle comuni azioni permesse all'interno dell'editor, quali l'importazione, l'esportazione e la condivisione, ; fornisce inoltre altre funzionalità all'interno del menù a tendina, inoltre attraverso il tasto "Help" è possibile accedere ai Tutorial forniti dagli sviluppatori e collegarsi con il centro assistenza, accedere al forum e ad altri servizi di supporto.

3→ Barra delle Proprietà: consente di modificare rapidamente le caratteristiche più semplici relative al tema, al testo, e agli oggetti quando sono selezionati. Le personalizzazioni riguardano ad esempio le dimensioni, il colore lo stile, il tipo del testo, delle linee , delle forme ecc…

4→ Toolbox: posto alla sinistra dell'area di disegno, contiene le librerie di forme messe a disposizione dell'utente, che può inoltre caricare delle immagini personali da includere nel diagramma.
5→ Pannello laterale: posto sul alto destro dell'editor. Si compone dei seguenti pannelli: di navigazione, grafico, metrico, di testo, di pagina principale, della cronologia, dei layers.
I vari pannelli vengono attivati cliccando sulle relative icone, ciò permette di accedere alla singole proprietà dei vari pannelli.
6→ Controlli di Pagina: collocato sul lato superiore destro dell'editor, permette di gestire facilmente le pagine di editing. È possibile aggiungere o rimuovere ulteriori pagine cliccando sui relativi pulsanti, ed effettuare il drag an drop delle page tabs per modificare l'ordine in cui sono mostrate

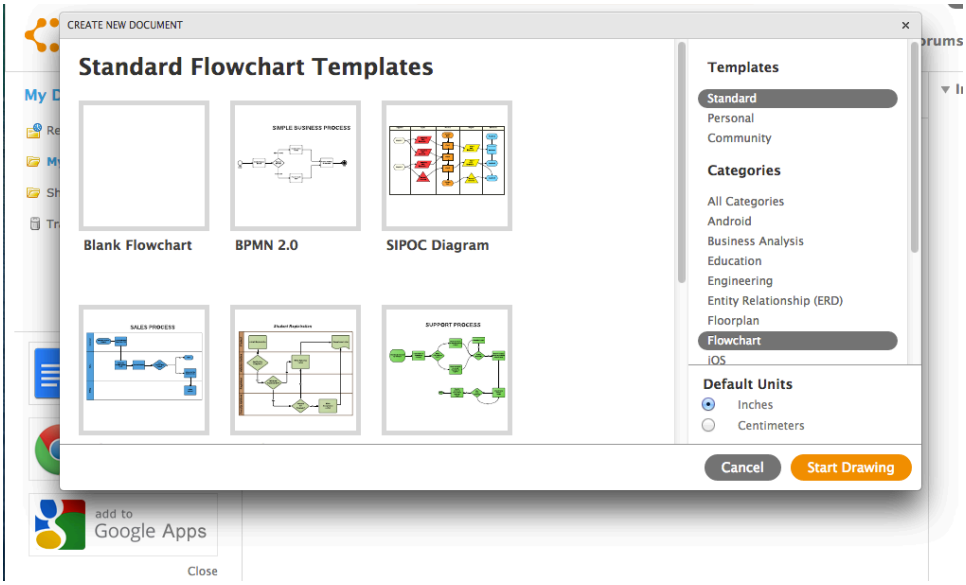


Figura 4 - Templates Editor [2]

Nella nuova scheda del browser che il servizio crea l'utente può modificare gli elementi del template, eliminarli, aggiungerli e compiere altre operazioni. La versione free di Lucidchart non ha limiti temporali ma consente di inserire massimo 60 oggetti e di salvare file, sotto forma di diagrammi, mappe e altro per una capacità massima di 25MB; inoltre non viene consentita la realizzazione di diagrammi collaborativi, mentre per avere maggiori servizi è necessario acquistare delle versioni Basic, Pro o Team. L'applicazione si basa sugli standard web quali HTML5 e Javascript e garantisce il supporto per tutti i principali browser web (Google Chrome, Firefox, Safari e Internet Explorer 8 +). È facilmente integrabile con Google Drive, Google Apps, JIRA, Confluence, Jive, and Box. Il software è di tipo proprietario quindi tutte le funzionalità complete possono essere utilizzate pienamente solo con il rilascio di versioni a pagamento. Sono messe a disposizione degli sviluppatori delle API che permettono di interagire con l'editor e ne permettono quindi l'integrabilità in altre applicazioni, a supporto di ciò è fornita una accurata documentazione.

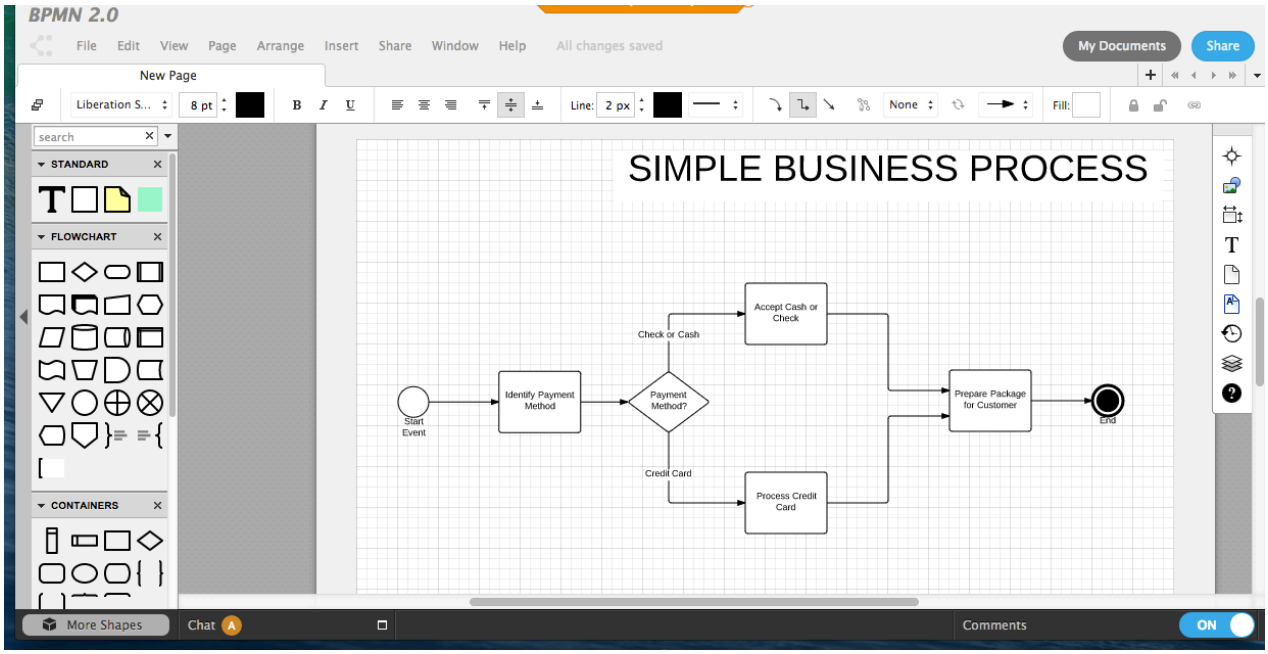


Figura 5 -Esempio diagramma Editor Lucidchart [2]

C. JALAVA

Si tratta di un semplice editor di diagrammi in Javascript che permette l'interazione online con i diagrammi creati, è stato progettato per poter essere facilmente personalizzabile in modo da poter progettare un proprio editor; è un'applicazione che viene seseguita da un browser web, tutttavia ad oggi esiste una versione beta dell'editor non recentemente aggiornata non è stato possibile testarne pienamente le funzionalità poiché non esiste una versione dimostrativa, inoltre la documentazione a supporto è carente e poco esaustiva.

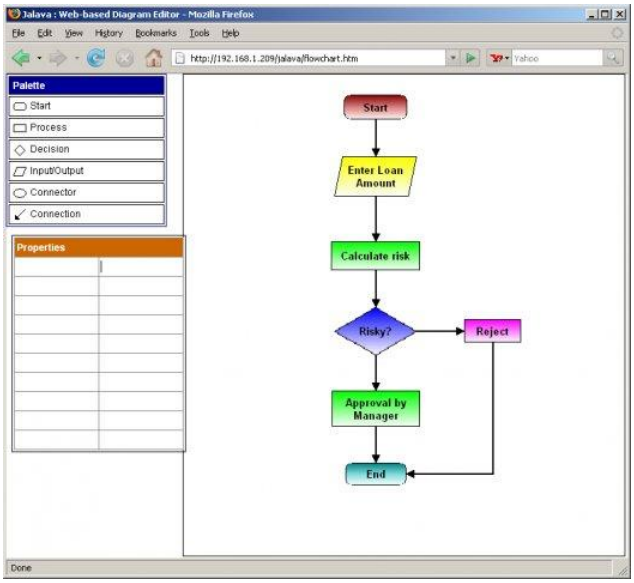


Figura 6 - Jalava Editor [9]

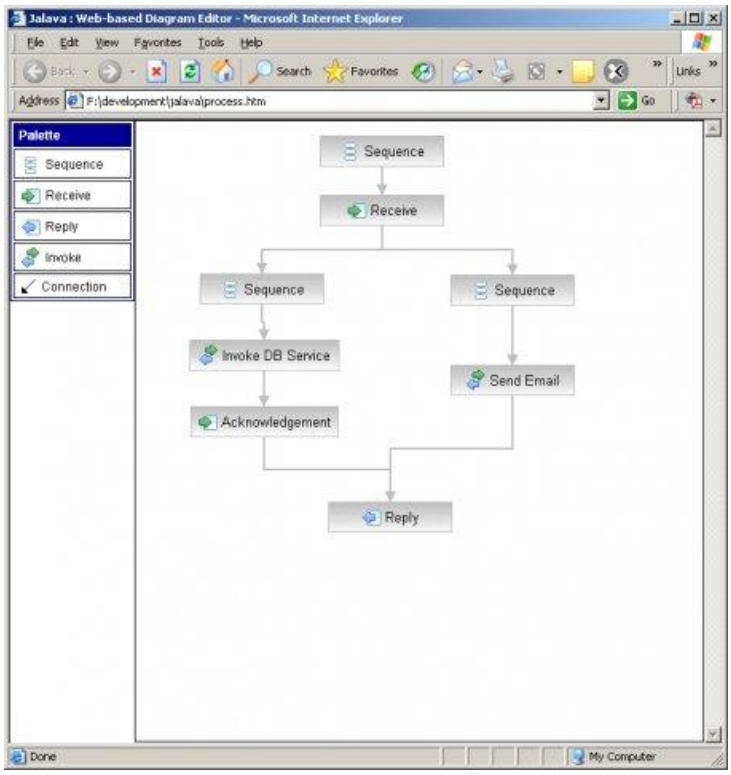


Figura 7 - Esempio Diagramma Jalava Editor [9]

D. DRAW IO

Draw.io è un editor grafico che permette di creare diagrammi di varie tipologie online (workflow, BPMN,UML,ER, diagrammi di reti…ecc) Non è richiesto il login né la registrazione al primo accesso all'interfaccia web vien data la possibilità all'utente di scegliere la modalità di archiviazione per il diagramma creato, selezionando una tra le quattro opzioni disponibili: GoogleDrive, Dropbox, Device, Browser.

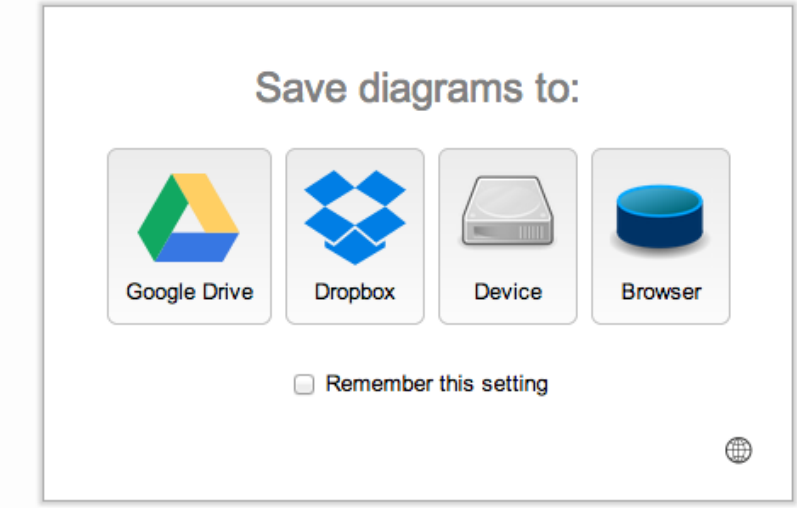


Figura 8 DrawIo

Se si selezionano le prime due opzioni (GoogleDrive e Dropbox) viene richiesto all'utente di confermare l'autorizzazione per accedere agli account personali, mentre è più semplificato il

salvataggio nel file system locale , viene infatti generato semplicemente un file xml contenente la definizione del diagramma. Tuttavia il salvataggio nel file system ha alcune limitazioni innanzitutto non disponibile per tablet e cellulari, non è permesso il salvataggio automatico e può essere definita la directory sulla quale effettuare il salvataggio. Infine è possibile salvare il diagramma direttamente nel browser per ogni singola sessione di lavoro. È possibile inoltre recuperare i diagrammi precedentemente salvati o eliminarli cliccando sulle apposite icone.

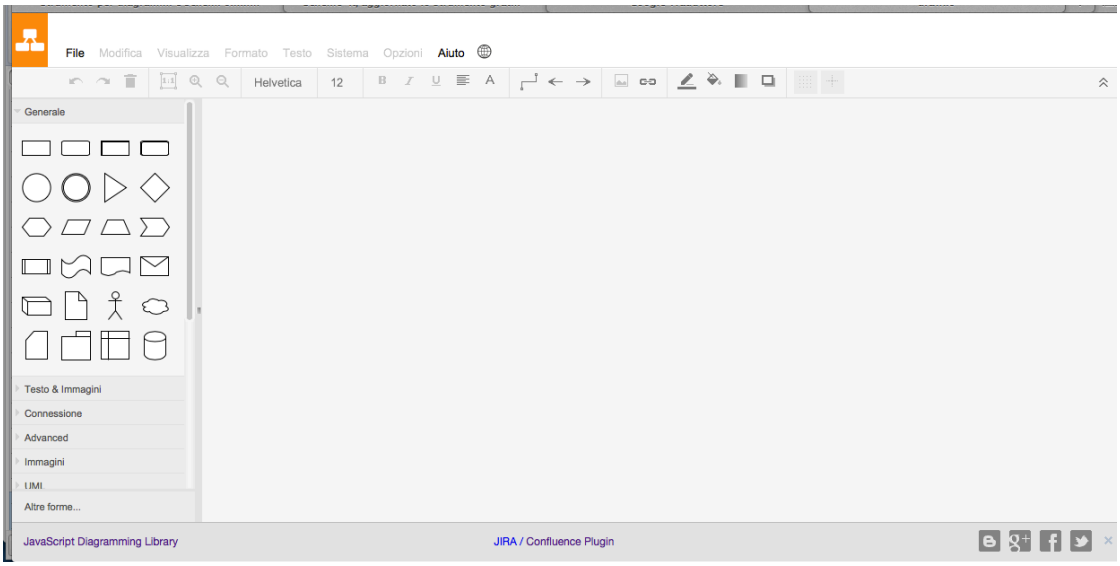


Figura 9- Editor Draw Io [3]

Gli strumenti messi a disposizione dell'utente sono molteplici , il tool è ben funzionante e performante. Esistono numerosi strumenti a supporto dell'utente (FAQs, Forum Supporto Online, Video Tutorial). Inoltre è garantita l'integrabilità con molti device(Android, Linux, Mac, Mobile Web App, Web-based, Windows). La vasta libreria di elementi messi a disposizione dell'utente permettono una gestione di essi attraverso un'interfaccia intuitiva, è possibile inoltre esportare i diagrammi in vari formati e integrali in blog o wiki. Draw.io mette a disposizione un'intera area entro la quale lavorare utilizzando un ampio set di risorse quali connettori, linee, forme, testo, clipart e molto altro ancora. È possibile impostare la lingua dell'interfaccia grafica ed è contemplata anche la possibilità di scelta dell' italiano, lo strumento è ricco dei features, strumenti ed opzioni, Draw.io offre la possibilità di salvare i propri lavori in differenti formati PNG, PDF, JPG, GIF, SVG, XML, con possibilità di successiva di reimportazione per modifiche dei formati XML. Permette la personalizzazione di ogni elemento sia nella definizione delle proprietà che nelle caratteristiche grafiche. Tuttavia relativamente agli elementi BPMN contemplati, pur mettendo a disposizione dell'utente tutta l'ampia gamma degli elementi compliant alla specifica BPMN , non viene ad esempio permessa l'assegnazione delle risorse all'interno di un workflow.

Il tool è sottoposto a costanti aggiornamenti e si sta pensando di rilasciare quanto prima una versione stan-alone scaricabile da Github che abbia tutte le funzionalità dell'applicazione on-line attuale, ad eccezione di quelle relative alle funzionalità lato client. Recentemente inoltre l'editor è stato dotato di funzionalità che garantiscono l'import di file da Gliffy e Lucidchart, infatti È ora possibile salvare il diagramma come un formato Gliffy JSON L'editor è stato realizzato usando una libreria commerciale mxGraph basata su javascript, della quale esistono versioni in costante aggiornamento.

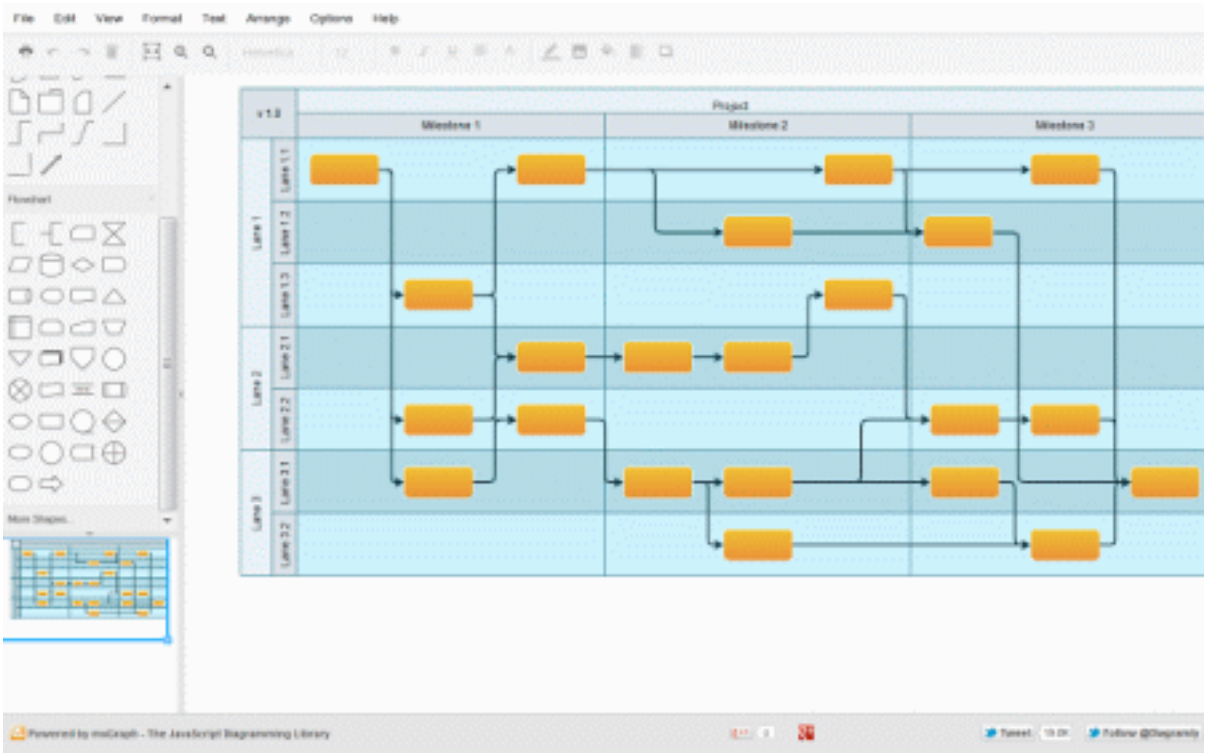


Figura 10 - Esempio Diagramm Draw IO [3]

E. JOINTJS

Si tratta di una libreria javascript open source basata su HTML5 che permette di definire elementi grafici , creare diagrammi e grafici e interagirne con gli elementi che lo compongono. Si basa sull'architettura MVC, mette a disposizione elementi di base per i diagrammi. La libreria può essere ottenuta attraverso una Mozilla Public License, che permette di accedere e ottenere il codice sorgente. È stata inoltre creata una estensione commerciale della libreria ,ovvero Rappid, che fornisce funzionalità aggiuntive. Rappid mette a disposizione un kit di strumenti completo per la definizione di diagrammi di vari tipologie (ERD;FSA;UML,Petri..)., infatti gli elementi a disposizione dell'utente sono raggruppati in basilari categorie. Attraverso la libreria JointJS è possibile gestire esclusivamente gli elementi di base per la creazione di diagrammi(rettangolo, cerchi , testo , immagini, ecc) , i diagrammi possono inoltre essere serializzati in JSON per favorirne lo scambio in applicazioni web.

```
var graph = new joint.dia.Graph;

var paper = new joint.dia.Paper({
    el: $('#myholder'),
    width: 600,
    height: 200,
    model: graph,
    gridSize: 1
});

var rect = new joint.shapes.basic.Rect({
    position: { x: 100, y: 30 },
    size: { width: 100, height: 30 },
    attrs: { rect: { fill: 'blue' }, text: { text: 'my box', fill: 'white' } }
});

var rect2 = rect.clone();
rect2.translate(300);

var link = new joint.dia.Link({
    source: { id: rect.id },
    target: { id: rect2.id }
});

graph.addCells([rect, rect2, link]);
```

Figura 11 - Esempio definizione elementi JointJS [13]

Rappid implementa inoltre una completa gamma di componenti per la gestione dell'interfaccia utente, per l'esportazione, la definizione del layout , la gestione dello storage, la collaborazione in tempo reale nella definizione di un diagramma, vengono inoltre forniti degli strumenti di programmazione per la gestione delle funzionalità di undo/redo, la manipolazione e la validazione del grafico; tutto ciò è gestito da appositi moduli che sono raggruppati in plugins.
La libreria è supportata da tutti i principali moderni browser : Google Crome(inclusa la versione mobile), Firefox, Safari, IE9+, Opera 15+. Non si stratta semplicemente di una librerai grafica ma grazie all'architettura MVC(più precisamente MV) su cui si basa viene separato l'aspetto grafico dal modello al quale un singolo elemento (grafico, elementi e link) corrisponde. Ciò facilita l'integrazione della libreria JointJS con e applicazioni di back-end con le quali si intende integrarla. La libreria è stata costruita a partire dalla libreria Backbone MVC, si basa inoltre su JQuery , Underscore e SVG.

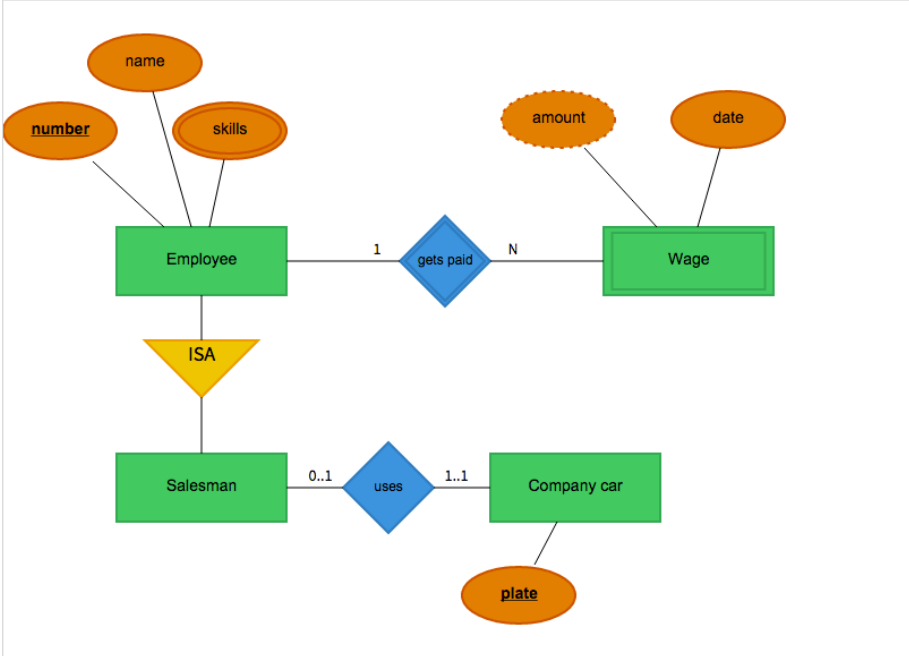


Figura 12 - Esempio diagramma JointJS [13]

I Diagrammi creati attraverso la libreria consistono di elementi collegati da link. Un diagramma in JointJS è rappresentato attraverso un modello di tipo joint.dia.Graph. Tale modello è capace di contenere celle (elementi e link), a partire dalla versione 0.6 di JointJs è possibile manipolare modelli relativi agli elementi e non le viste. La documentazione a supporto è esaustiva e chiara, esistono tutorial e documentati Api , inoltre si può usufruire degli strumenti di community e far parte del relativo google group attivato dagli sviluppatori per poter avere chiarimenti e risoluzione di eventuali bug.
Attraverso la libreria Rappid sono stati creati degli editor con funzionalità essenziali per al definizione di diagrammi di vari tipologie (Reti di Petri, Diagrammi Entità-Relazioni, Macchine a stati, Diagrammi UML). L'utente ha la possibilità di interagire con gli elementi grafici e di impostarne tutte le principali proprietà.

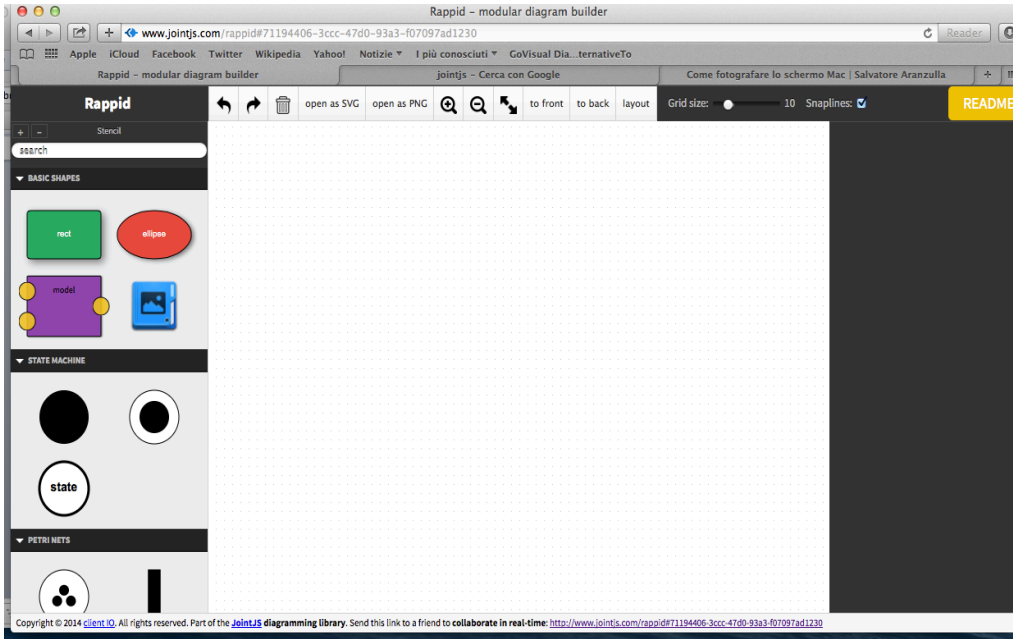


Figura 13 - Editor Rappid [13]

Interessante si è inoltre rivelato l'editor BPMN creato con la libreria Rappid che fornisce tutti i principali elementi costitutivi di un diagramma BPMN, l'interazione con gli elementi è intuitiva e non artificiosa.
È possibile personalizzare le proprietà di tutte le entità messe a disposizione, attraverso una barra laterale che si attiva per ogni elemento selezionato nella quale sono settate le caratteristiche associate a ciascuna entità. Vengono inoltre forniti le funzionalità di base per il caricamento di un diagramma precedentemente salvato in google drive e il relativo salvataggio. Infine è possibile visualizzare il formato Json relativo al diagramma che si sta realizzando.

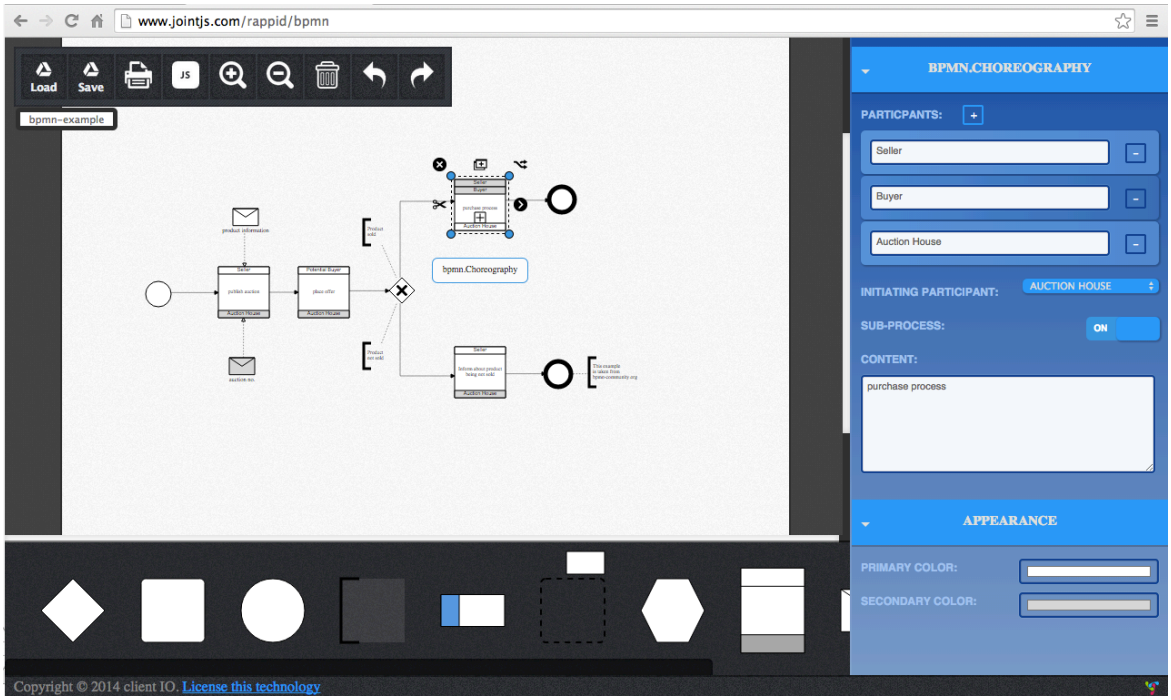


Figura 14 Editor BPMN Rappid [13]

F. Gliffy

Gliffy è un editor web ovvero un prodotto SaaS(software as a service) realizzato in HTML5 che permette all'utente di creare numerose tipologie di diagrammi attraverso un interfaccia abbastanza intuitiva . I punti di forza di Gliffy riguardano la possibilità, per un team di colleghi, di collaborare contemporaneamente alla stesura di un progetto. L'applicazione è essenzialmente "web based" può essere impiegata per realizzare diagrammi "flow chart", UML, schemi di reti, grafici entità-relazione e così via. L'editor permette all'utente di personalizzare stili e colori ,inserire immagini , aggiungere testo e di poter usufruire di una guida interattiva a supporto delle azioni che è possibile compiere con gli strumenti principali messi a disposizione, la versione Pro permette di inserire delle entità personalizzate attraverso l'aggiunta di una libreria integrabile nell'editor. Esistono numerosi strumenti a supporto dell'utente, forum attivi, tutorial, manuali.
Una volta avviata l'interfaccia grafica verrà mostrata all'utente una guida che spiega l 'utilizzo degli strumenti principali ciò rende facile la creazione di diagrammi che possono inoltre essere condivisi sui principali social network.

Oltre alla versione online esistono delle versioni dell'editor come Gliffy Confluence Plugin e Gliffy JIRA Plugin. La versione on-line permette di creare gratuitamente fino a 5 diagrammi e si ha la possibilità di salvare direttamente sul server Gliffy (condividendo quindi i diagrammi con altre postazioni) o di esportare il proprio lavoro in vari formati(da semplici immagini ad immagini di tipo vettoriale). Anche se l'interfaccia è abbastanza complessa la sua lettura è facile e immediata grazie alla possibilità che ha l'utente di visualizzare un pop-up di testo di aiuto che viene visualizzato se si esegue il mouse sopra i diversi strumenti e pulsanti presenti nella zona di lavoro.
È garantito il supporto per tutti i principali browser, non esistono quindi problemi di compatibilità. Per poter utilizzare il servizio è necessari una fase di registrazione che premette di utilizzare il software completo per 30 giorni, scaduti i quali l'account verrà convertito nel piano Basic che prevede alcune limitazioni d'uso, quali il numero massimo di diagrammi. Inoltre l'account Premium consente la creazione di più utenti al fine realizzare diagrammi collaborativi. Un ulteriore funzionalità interessante messa a disposizione dal tool riguarda la pubblicazione del diagramma creato sul sito Gliffy: attraverso una specifica URL è possibile così condividere i file con clienti e colleghi, infine è garantito il supporto per il controllo di revisione dei file, una funzionalità che diventa fondamentale quando si lavora in gruppo ed è necessario tenere traccia di modifiche o mantenere divere versioni di un diagramma.

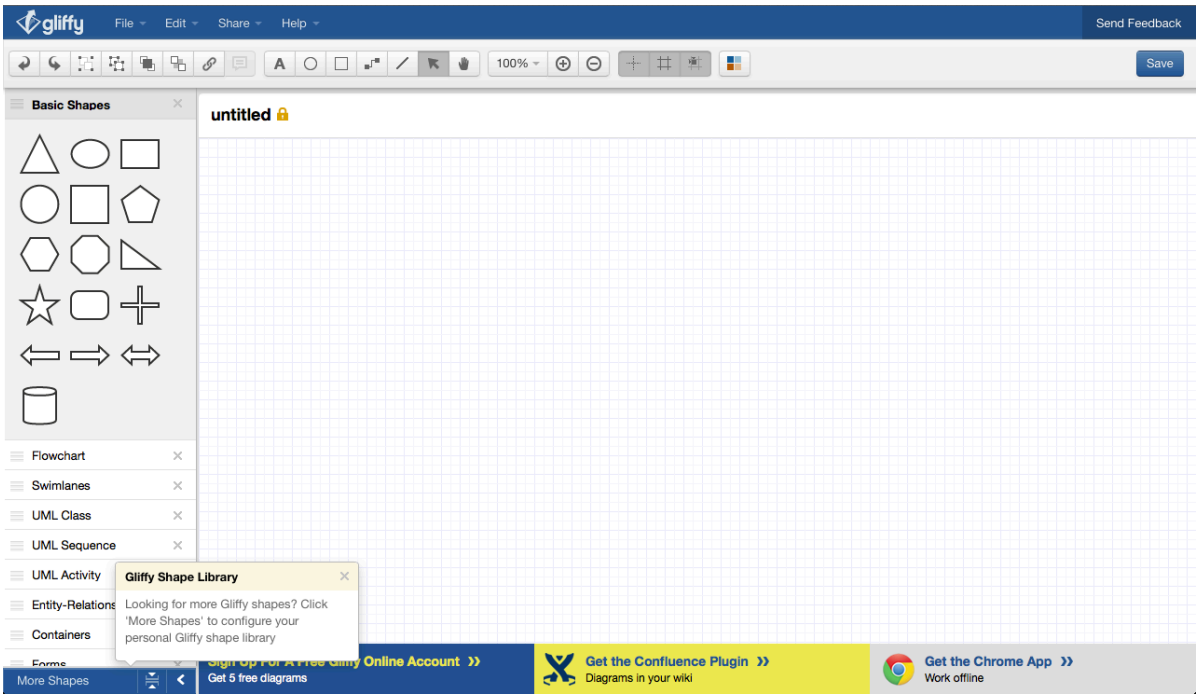


Figura 15 - Editor Gliffy[4]

G. Cacoo

Editor user friendly permette la realizzazione una gran varietà di diagrammi online, a questo scopo sono a disposizione dell'utente diverse forme predefinite che permette la facile creazione di grafici, raggruppati in svariate categorie, può essere utilizzato on line gratuitamente ma con delle limitazioni sulle funzionalità nonché sul numero dei diagrammi realizzabili(25), per ottenere un estensione illimitata delle funzionalità è necessario acquisire

una versione a pagamento; tuttavia adesso , probabilmente per un limitato periodo ,è stata messa a disposizione del mondo accademico il livello gratuito Team Plan.
Le forme predefinite a disposizione dell'utente sono raggruppate in : Sitemap,Wireframe (Layout di un sito/applicazione),Wireframe a mano libera,Web Service, Diagrammi di flusso,Diagrammi ,UML, Diagrammi ER, Figure base, Fumetti, Persone, Faccine, Biglietti di auguri, Linee dimensionali, Apparecchiature per ufficio, Reti, Disposizione ufficio, Icone semplici, File, iPhone, iPad, Android, Circuiti elettronici. Gli utenti possono aggiungere propri stencil e templait. Con Cacoo inoltre è disponibile la collaborazione in tempo reale , infatti un singolo diagramma può essere editato contemporaneamente da più persone rendendo le modifiche visibili in tempo reale. Gli elementi possono essere facilmente inseriti nel diagramma e collegati attraverso i connettori messi a disposizione, è possibile usare dei text box per inserire del testo, è possibile inserire delle immagini personali o importarle da in sito web attraverso l'URL, vengono inoltre forniti dei suggerimenti all'utente durante l'editing del diagramma. Al fine di realizzare una performante collaborazione ciascun utente può invitare più persone al collaborare nella stesura di un diagramma ; la comunicazione tra gli utenti del team è resa possibile attraverso una chat che permette di comunicare in tempo reale durante la creazione di un diagramma condiviso; l'icona del relativo utente che sta operando su un oggetto viene posizionata vicino ad esso si possono creare delle cartelle condivise che permettono di rendere editabili dei diagrammi ad un certo numero di utenti. I diagrammi tuttavia possono anche essere resi privati oppure pubblici associandoli ad un URL . i diagrammi sono esportabili in SVG o PNG e possono essere condivisi sui social network. È possibile realizzare l'interoperabilità di Cacoo con altri servizi ed applicazioni usando le API messe a disposizioni.

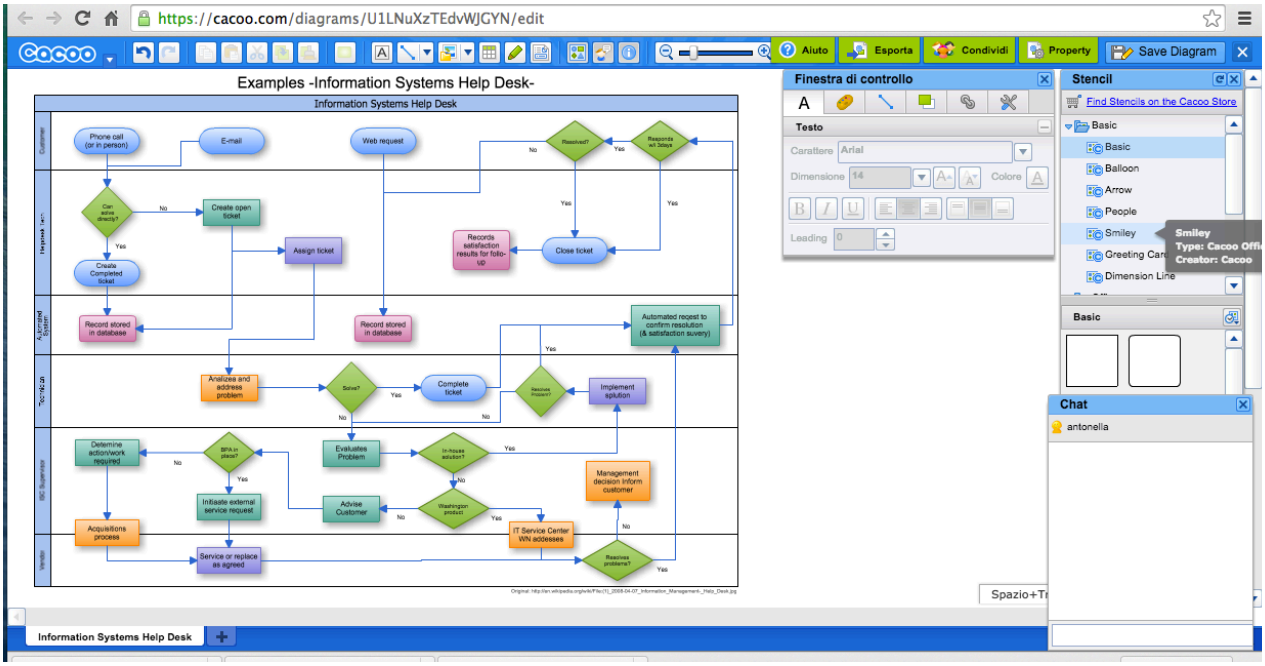


Figura 16 - Cacoo Editor[5]

H. Oryx/Signavio

Signavio è un editor di business Process basato sul web, lanciato nel maggio 2009. Il prodotto consente la creazione di diagrammi di processo che utilizzano la notazione BPMN ed è

disponibile sia come Software as a Service (SaaS) che come software commerciale per istallazioni On-Premise.

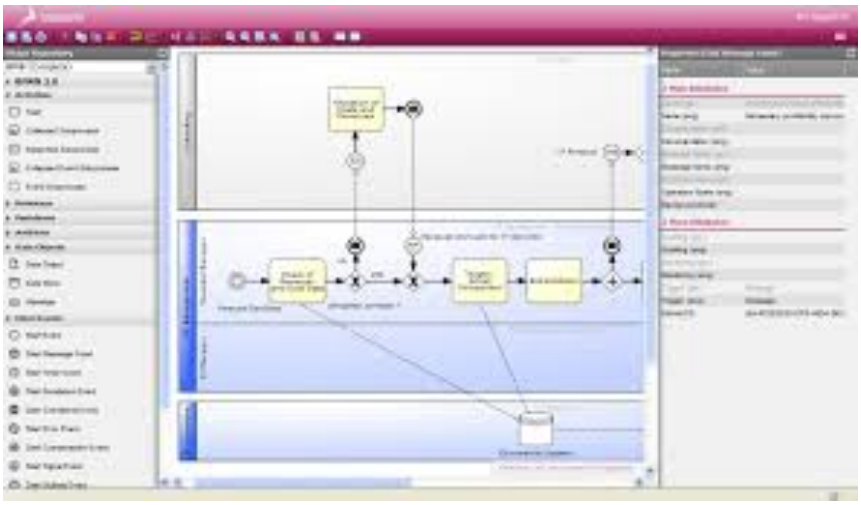


Figura 17 - Editor Signavio[7]

L'editor creato premette di creare diagrammi e di condividerli , inviarli ad altri ed aggiungere commenti. Le principali caratteristiche riguardano:
la modellazione del processo,
la possibilità di effettuare la simulazione del workflow per individuare eventuali colli di bottiglia o valutare delle possibili alternative.
modello di repository per costruire architetture di processo multilivello, gestire le versioni e riutilizzare oggetti;
funzionalità relative alla collaborazione e alla condivisione di diagrammi
pubblicazione di documentazione del processo usando meccanismi di reporting.
Il servizio cloud completo è disponibile attraverso l'acquisto di una licenza, oppure è messa a disposizione degli utenti una versione ti prova gratuita di 30 giorni. Si può accedere alle funzionalità in seguito all'effettuazione del login.
Esiste tuttavia una versione open source dell'editor Signavio che fornisce delle funzionalità di base ma che ne permette l'estensibilità, infatti il progetto Signavio muove i suoi passi a partire dall'editor Oryx; ovvero un editor web based per la modellazione di processi di business attraverso i linguaggi BPMN. I principali sviluppatori del progetto Oryx hanno fondato Signavio nel maggio 2009, per tale motivo non viene più garantita la manutenzione del codice dell'editor Oryx poiché il progetto converge ormai verso l'editor Signavio.
L'editor Signavio si presenta molto performante con un interfaccia grafica intuitiva e completa di tutte le funzionalità utili alla definizione di un esaustivo workflow. Fornisce tutti i principali elementi in conformità alla specifica BPMN. È permesso salvare il file creato e recuperarlo per modifiche future, impostare tutti i principali attributi relativi ad ogni elemento, infine attraverso un apposito pulsante è possibile individuare gli errori sintattici contenuti nel diagramma e procedere alla loro correzione.

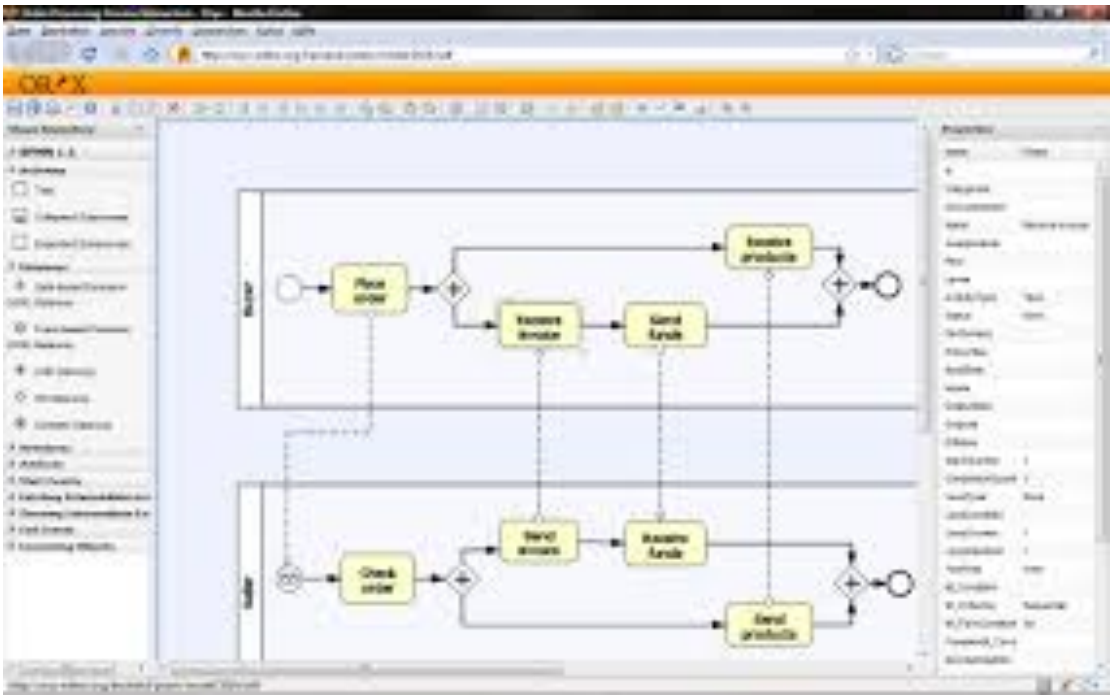


Figura 18 -Editor Oryx [7]

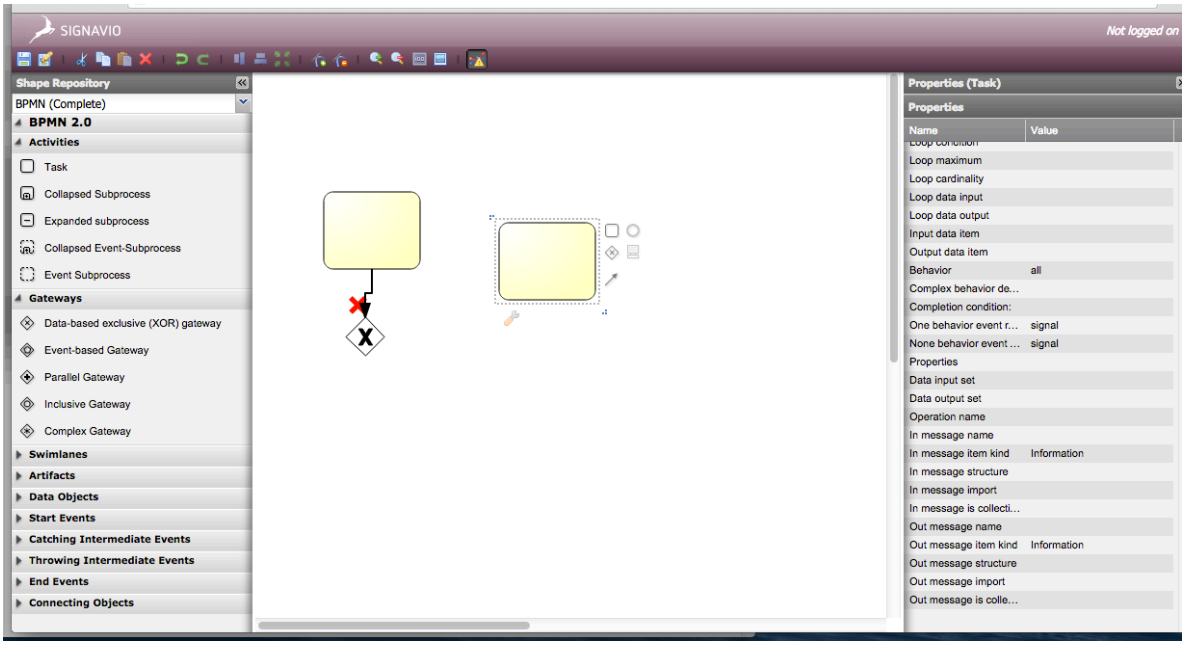


Figura 19 Editor Signavio : controllo sintattico [7]

I. Bpmn.io

È un servizio web messo a disposizione da Camunda (piattaforma open source per la gestione di workFlow e l'automazione di processi di business)per l'editing di diagrammi BPMN. L'editor può essere usato stand-alone oppure integrato nelle applicazioni personali, essendo a disposizione degli sviluppatori il codice sorgente accessibile su GitHub. Vengono attualmente forniti solo pochi elementi essenziali (Event, Gateway, Task, Pool, Annotation) per definire un semplice diagramma BPMN, non viene invece offerta la possibilità di poter impostare le varie

tipologie associate agli elementi di tipo Task (Manual, Receive, Send, Service, ecc) né per gli elementi Gateway (Pareallel, Inclusive,Exclusive). Inoltre non vi è la possibilità di inserire alcun elemento di tipo Data Object né Message. L'utente può visualizzare dei diagrammi di esempio o generarne uno ex-novo, inoltre è possibile effettuare l'undo delle azioni di editing eseguite, salvare diagramma in un file .bpmn che contiene la definizione xml del diagramma in accordo alla specifica BPMN 2.0, oppure salvarlo come immagine SVG. Inoltre è possibile recuperare un file precedentemente salvato per apportare delle ulteriori modifiche. Intuitivo è inoltre l'interazione con i vari elementi inseriti nel diagramma, nonché utile le icone che vengono mostrate all'utente alla selezione di un oggetto grafico che permetto di evidenziare le azioni consentite conformemente alla alle regole sintattiche della specifica BPMN. L'editor si presenta quindi nel complesso preformante ma molto rudimentale relativamente alle funzionalità e agli elementi messi a disposizione dell'utente.

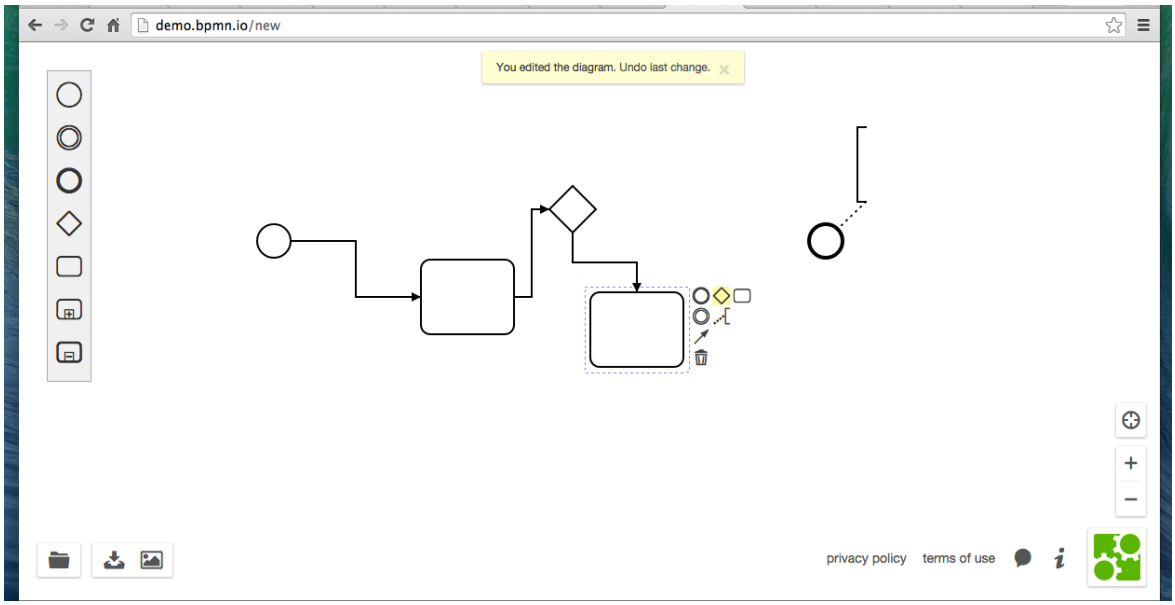


Figura 20 - Editor BPMN.io [6]

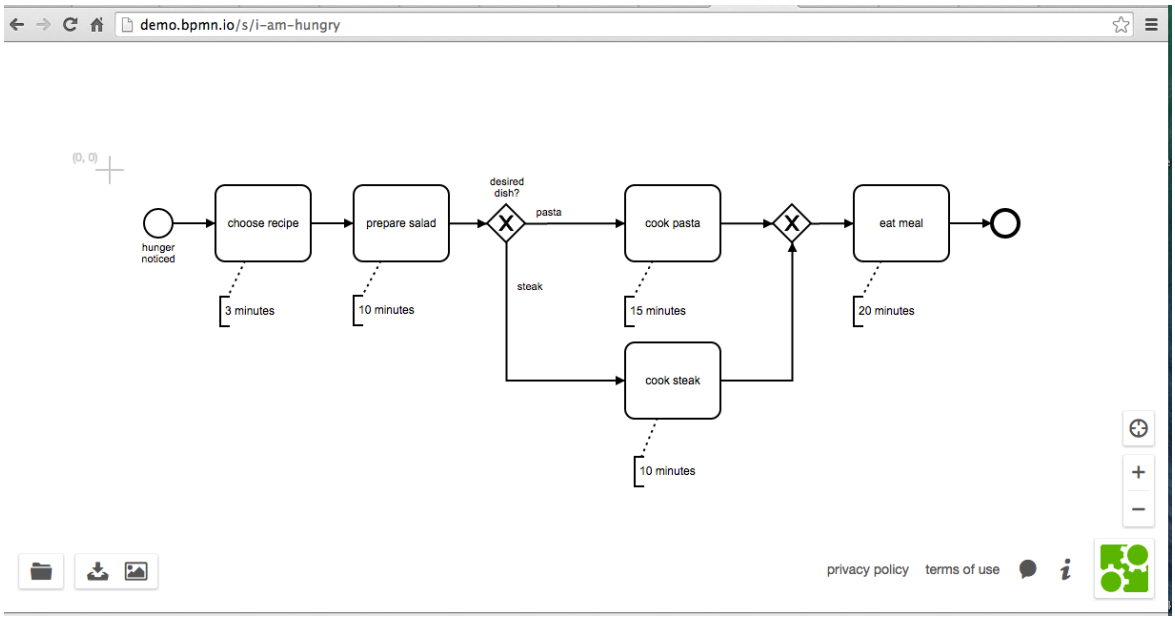


Figura 21 - Esempio diagramma Bpmn.io [6]

PROGETTAZIONE

In seguito all'analisi dei vari tools web based per la definizione di diagrammi di workflow , è emerso che sono pochi gli strumenti open source capaci di implementare delle specifiche funzionalità quali:

- Generazione dinamica degli elementi della palette
- Validazione sintattica e semantica di un workflow
- Generazione ed Editing della specifica goalsSpec di un processo e injection nel middleware MUSA

le funzionalità necessarie ai fini del progetto, si è resa pertanto indispensabile lo studio delle librerie a supporto dello sviluppo di un editor web based attraverso il quale l'utente possa definire un proprio business process usando i principali elementi definiti nello standard BPMN.

A. Diagramma dei casi d'Uso

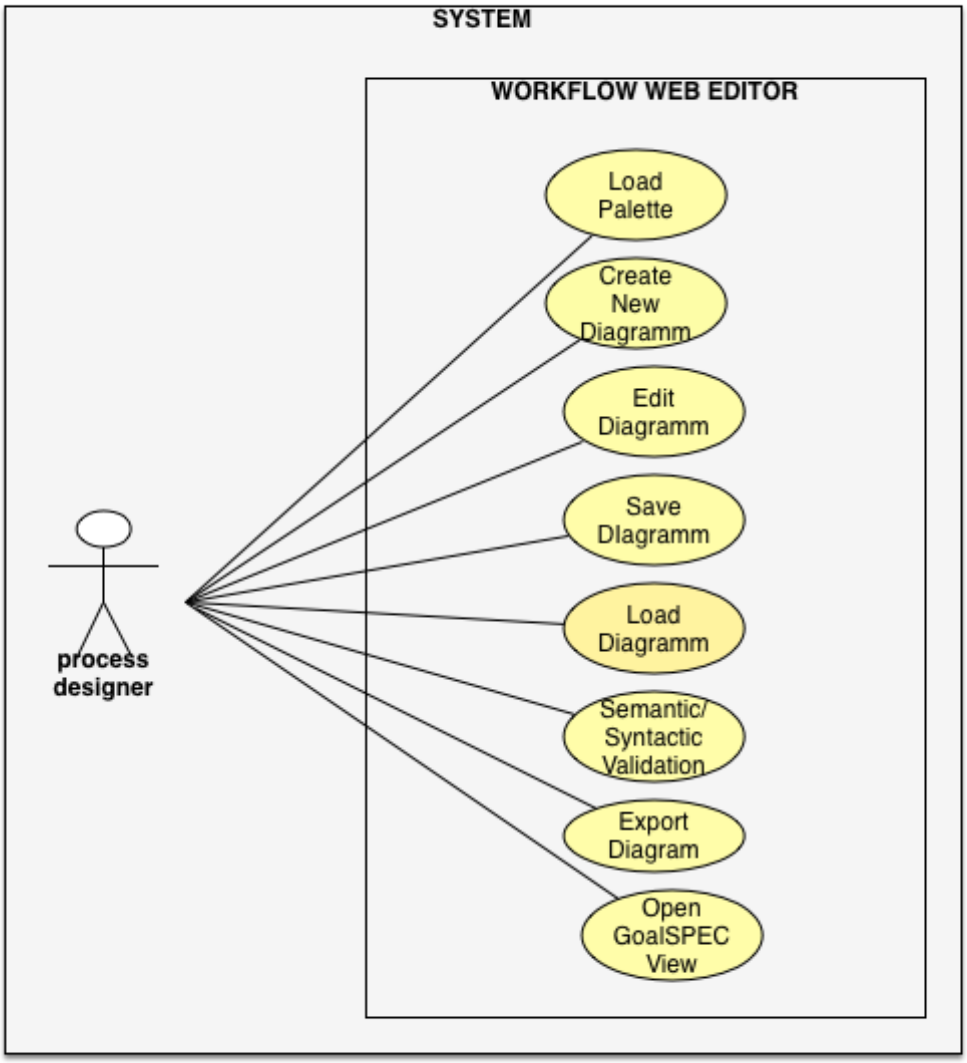
A. 

Figura 22 - Diagramma dei casi d'uso

La Figura 22 mostra il diagramma dei casi d'uso relativo al tool realizzato. Nel seguito di questo paragrafo essi saranno singolarmente descritti ed analizzati.

- ***Load Palette***
  1. ***Name***: Load Palette
  2. ***Participant actor***: Process Designer
  3. ***Entry condition***: il Process Designer seleziona il file contenente la definizione della palette
  4. ***Exit condition***: gli elementi sono stati inseriti nella barra degli strumenti
  5. ***Flow of events***:
     a. Il process designer seleziona il file contenete la definizione degli elementi che devono essere presenti nella palette
     b. Il sistema inserisce correttamente gli elementi grafici
     c. Il process designer può inserire gli elementi desiderati nel diagramma attraverso il drag and drop
  6. ***Special requirement***: none

- ***Create new Diagramm***
    1. ***Name***: Create New Diagramm
    2. ***Participant actor***: Process Designer
    3. ***Entry condition***: il Process Designer attraverso l'apposito pulsante sulla barra del menu inizia una nuova sezione di lavoro
    4. ***Exit condition***: il process designer salva il diagramma
    5. ***Flow of events***:
        a. Il process designer avvia una nuova sezione di lavoro
        b. Il process designer realizza il diagramma necessario
        c. Il process designer salva il progetto creato
    6. ***Special requirement***: none

- ***Edit Diagramm***
    1. ***Name***: Edit Diagramm
    2. ***Participant actor***: Process Designer
    3. ***Entry condition***: il Process Designer crea un nuovo diagramm
    4. ***Exit condition***: il process designer salva il diagramma
    5. ***Flow of events***:
        a. Il process designer crea un nuovo diagramma
        b. Il process designer inserisce tutti gli elementi necessari a realizzare il workflow
        c. Il process designer salva il progetto creato
    6. ***Special requirement***: none

- ***Save Diagramm***
    1. ***Name***: Save Diagramm
    2. ***Participant actor***: Process Designer
    3. ***Entry condition***: il Process Designer ha completato il diagramma
    4. ***Exit condition***: il process designer salva il diagramma
    5. ***Flow of events***:
        a. Il process designer clicca sul pulsante salva
        b. Il sistema verifica la correttezza semantica e sintattica del diagramma creato
        c. Il sistema comunica gli eventuali errori presenti nel diagramma
        d. Il diagramma creato viene salvato sul file system dell'utente
    6. ***Special requirement***: none

- ***Load Diagramm***
    1. ***Name***: Load Diagramm
    2. ***Participant actor***: Process Designer
    3. ***Entry condition***: il Process Designer clicca il pulsante open del menu
    4. ***Exit condition***: il sistema mostra all'utente il diagramma caricato
    5. ***Flow of events***:
        a. Il process designer clicca sul open file
        b. Il sistema importa il file scelto dall'utente
        c. Il diagramma scelto viene mostrato all'utente nell'area di lavoro
    6. ***Special requirement***: none

- ***Semantic/Syntactic Validation***
    1. ***Name***: Semantic/Syntactic validation

2. ***Participant actor***: Process Designer
3. ***Entry condition***: il Process Designer clicca il pulsante di validazione presente nel menu
4. ***Exit condition***: il sistema mostra all'utente gli errori presenti nel diagramma
5. ***Flow of events***:
    a. Il process designer clicca sul pulsante di validazione del diagramma
    b. Il sistema effettua i controlli sintattici e semantici del diagramma
    c. Il sistema mostra all'utente l'elenco degli errori presenti nel diagramma
    d. Il sistema evidenzia nel diagramma gli elementi interessati dal ciascun errore
    e. L'utente apporta le modifiche necessarie
6. ***Special requirement***: none

- ***Export Diagramm***
    1. ***Name***: Export Diagramm
    2. ***Participant actor***: Process Designer
    3. ***Entry condition***: il Process Designer clicca il pulsante per esportare il file nel formato xmi standard
    4. ***Exit condition***: il sistema crea nel file system dell'utente di il file contenente la definizione xmi del diagramma
    5. ***Flow of events***:
        a. Il process designer clicca sul pulsante di export del diagramma
        b. Il sistema effettua il parsing del diagramma nel formato xmi BPMN definito dallo standard OGM
        c. Il file contenente la definizione xmi del diagramma viene creato nel file system dell' utente
    6. ***Special requirement***: none

- ***Open GoalSPEC view***
    1. ***Name***: Open GoalSPEC view
    2. ***Participant actor:*** Process Designer
    3. ***Entry condition:*** Il process designer clicca sul pulsante che permette la visualizzazione dell'editor GoalSPEC
    4. ***Exit condition:*** il sistema mostra all'utente una nuova finestra in cui viene mostrato un editor GoalsSPEC con la possibilità di ottenere la specifica in tale linguaggio del workflow definito
    5. ***Flow of events:***
        a. Il process designer clicca sul pulsante di visualizzazione della definizione goalSPEC del workflow
        b. Il sistema genera una nuova finestra rappresentante l' editor goalsSPEC
        c. L'utente visualizza la definizione goalSPEC del workflow
    6. ***Special requirement: none***

B.

B. Diagrammi di Sequenza

Nel seguito di questo paragrafo verranno illustrati i diagrammi di sequenza corrispondenti ai principali casi d'uso considerati. Attraverso la loro analisi è facilmente intuibile quali sono le

modalità di interazione tra i vari moduli javascript che sono coinvolti in ciascuno scenario rappresentato.

C.

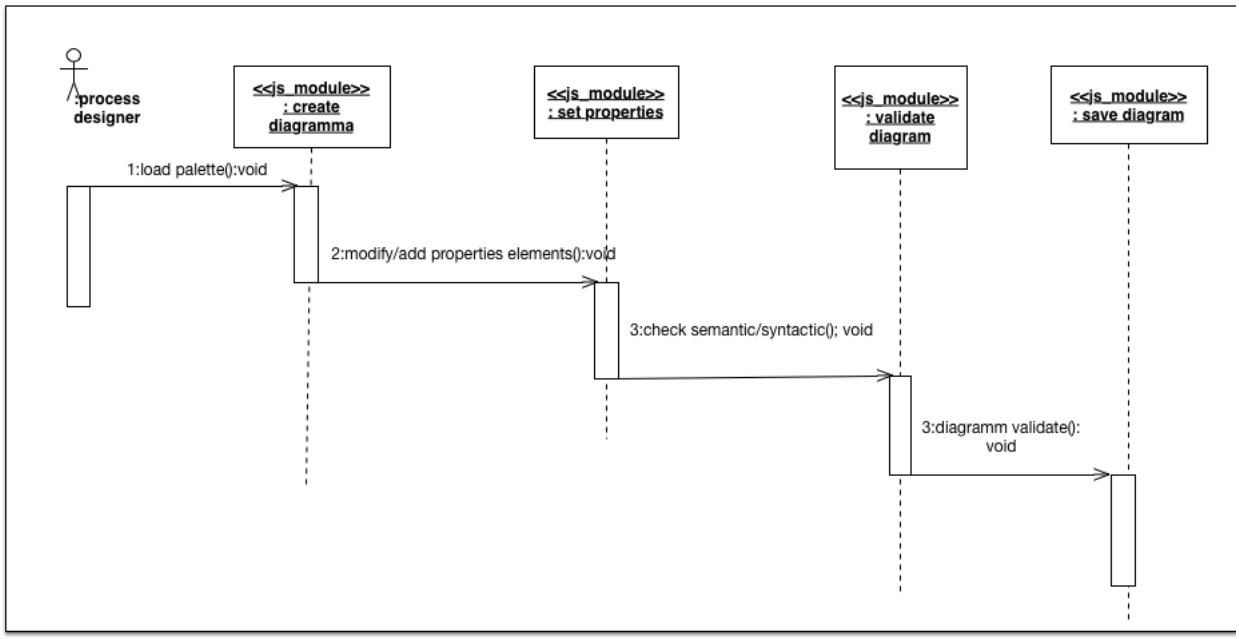


Figura 23 - Diagramma di sequenza principale

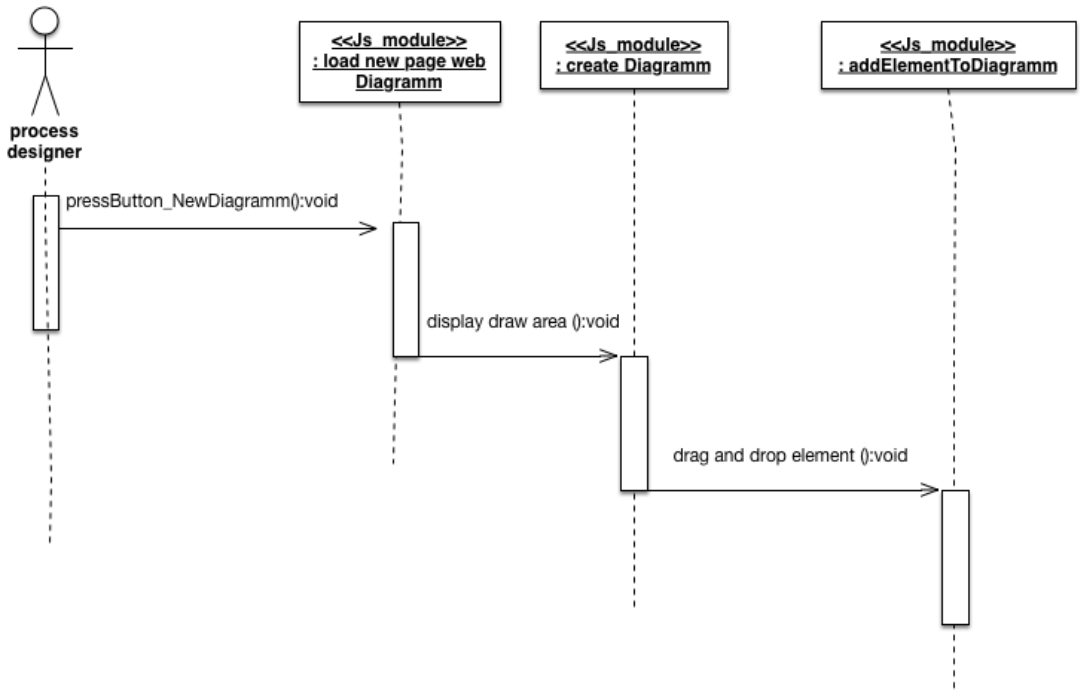


Figura 24 - Diagramma di sequenza relativo alla creazione di un nuovo diagramma

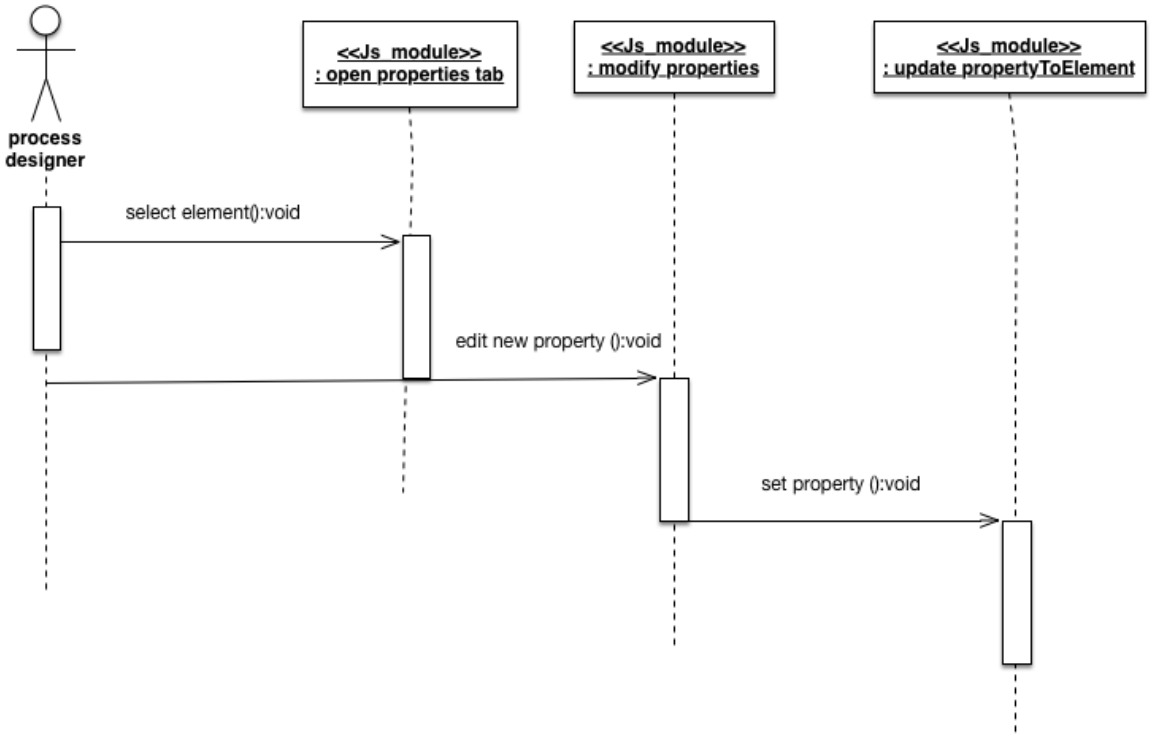


Figura 25 - Diagramma di sequenza relativo all'aggiornamento di una proprietà di un elemento

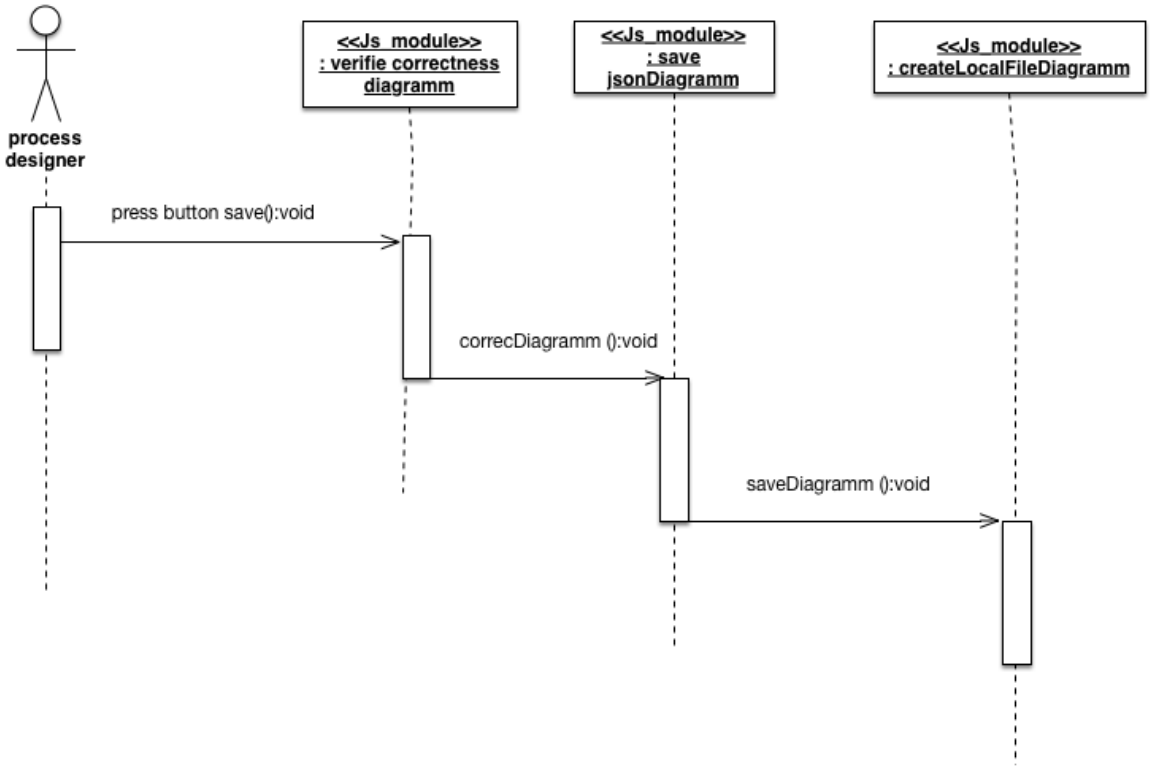


Figura 26 - Diagramma di sequenza relativo al salvataggio del Diagramma

C. Diagramma delle Attivita’

D.

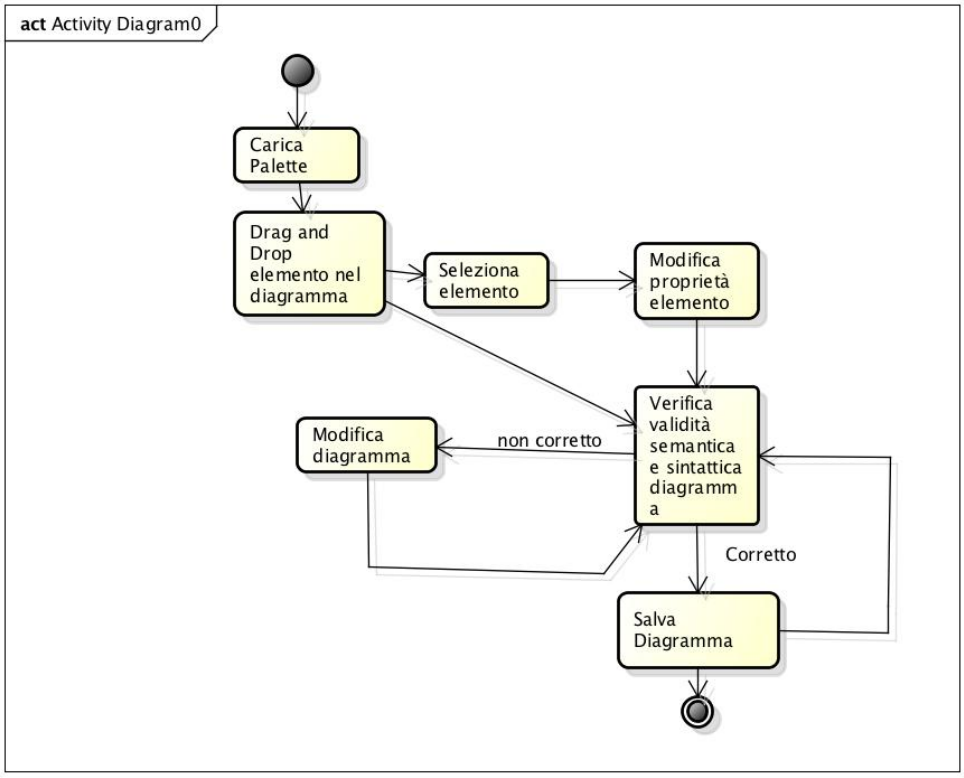


Figura 27 - Diagramma delle Attività

LIBRERIE UTILIZZATE

Al fine di poter realizzare le funzionalità dell’editor web based progettato si è reso necessario l’utilizzo di particolari librerie open source le cui caratteristiche e modalità di utilizzo sono illustrate nel seguito del seguente paragrafo.

A. Jointjs

L’interattività nella definizione di un workflow è stata realizzata sfruttando le potenzialità della libreria javascript JointJS. Grazie all’architettura MVC sulla quale si basa la libreria, è stato possibile realizzare nella definizione dell’editor una separazione tra gli elementi relativi al Models e le corrispettive Views. Come emerge nella Figura 28 i principali elementi presenti in un diagramma con i quali l’utente interagisce sono il *paper* , al quale deve essere legato il modello del diagramma contenuto in un oggetto di tipo *graph*, che contiene al suo interno le viste relative ad *elementi e link.*

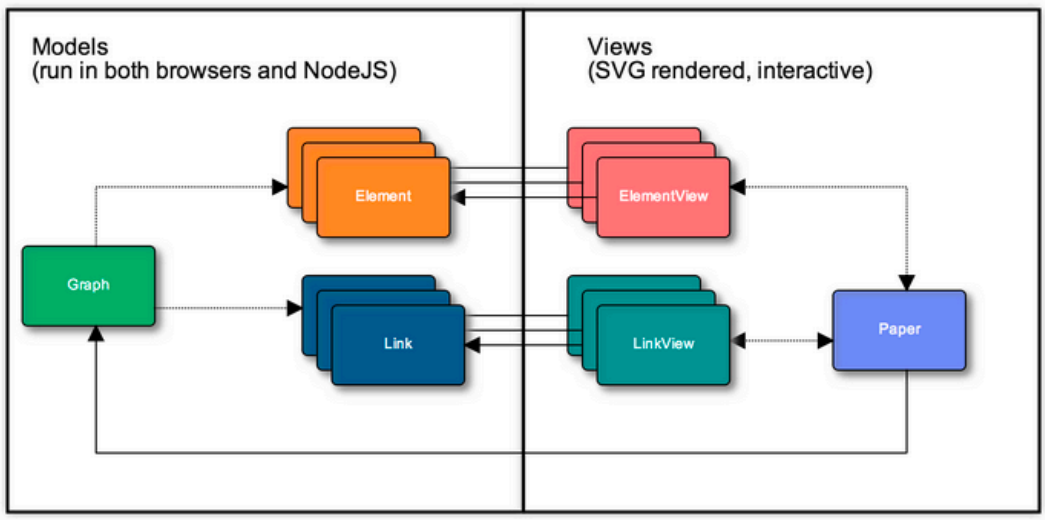


Figura 28 - Architettura alla base della Libreria JointJS

Personalizzando i costrutti di base forniti dalla libreria sono stati definiti tutti gli elementi necessari per generare un diagramma BPMN; in particolare per ciascun elemento BPMN è stato implementata una nuova joint.shapes, il cui aspetto grafico è conforme al relativo elemento definito nella specifica BPMN,. Ciascuna forma è stata creata combinando gli elementi di base messi a disposizione da jointJS. Si consideri a titolo di esempio la definizione di un elemento di tipo Activity definito nella Figura 29, come si può notare oltre alla dichiarazione relativa alla composizione dell'elemento grafico è stato necessario gestire opportunamente l'evento associato ad scelta della tipologia di Task da inserire nel diagramma, contemplando tutti i possibili tipi definiti nella specifica BPMN (Service, User, Manual, Script, Send, Receive, Reference). Analoghe operazioni sono state svolte relativamente alla definizione di elementi di tipo Gateway, Event, Message, DataObject e Pool.

```
joint.shapes.basic.ActivityView = joint.shapes.basic.TextBlockView;
                    joint.shapes.basic.Activity = joint.shapes.basic.TextBlock.extend({
                                markup: '<g class="rotatable"><g class="scalable"><rect class="body outer"/><rect class="body
inner"/></g><switch><foreignObject requiredFeatures="http://www.w3.org/TR/SVG11/feature#Extensibility" class=" body fobj"><body
xmlns="http://www.w3.org/1999/xhtml"><div/></body></foreignObject><text class="content"/></switch><image/></g>',
defaults: joint.util.deepSupplement({
                    type: 'basic.Activity',
                            attrs: {
                                rect: {
                                    rx: 8,
                                    ry: 8,
                                },
                                  '.body': {
                                    fill: '#ffffff',
                                    stroke: '#000000',
                                },
                                '.outer': {
                                          fill: 'white',
                                          stroke: 'white',
                                          width: 400,
                                          height:400,
                                          magnet:true
                                },
                                '.inner': {
                                      transform: ' translate(30,30)',
```

```
                width: 340,
                height: 340,
            },
             image: {
                    ref: '.inner',
                'ref-x': 5,
               'ref-y': 7,
                width: 13,
                height: 13,
                'xlink:href': ''
            },
              '.label': {
                ref: '.body',
                'ref-x': .5,
                'ref-dy': 5,
                text: '',
                'text-anchor': 'middle'
              },
              '.notes':{text:''},
              '.resourceProperty':{text:''}
          },
          content: {
            text:'',
              'text-anchor': 'middle',
            },
          '.fobj': {
            ref: '.inner',
             },
            taskType:'default'
}, joint.shapes.basic.TextBlock.prototype.defaults),
           onTaskTypeChange: function(cell, type) {
            switch (type) {
             case 'default':
                      cell.attr('image/xlink:href', '');
                      break;
             case 'service':
                    cell.attr('image/xlink:href', 'img/icon/serviceIcon.jpg');
                      break;
             case 'user':
                    cell.attr('image/xlink:href', 'img/icon/userIcon.jpg');
                      break;
             case 'manual':
                    cell.attr('image/xlink:href', 'img/icon/handIcon.png');
                      break;
             case 'script':
                    cell.attr('image/xlink:href', 'img/icon/textIcon.jpg');
                      break;
             case 'send':
                    cell.attr('image/xlink:href', 'img/icon/sendIcon.jpg');
                    break;
             case 'receive':
                    cell.attr('image/xlink:href', 'img/icon/messageIcon.jpg');
                      break;
             case 'reference':
                    cell.attr('image/xlink:href', 'img/icon/referenceIcon.jpg');
                      break;
          }
      },
});
```

Figura 29 - Esempio definizione elemento Task

Al suo interno la libreria JointJs sfrutta le potenzialità di due ulteriori librerie "*Vectorized*" e "*Geometry*", che permettono rispettivamente di manipolare gli elementi SVG agevolmente e di effettuare delle operazioni geometriche sulle forme.

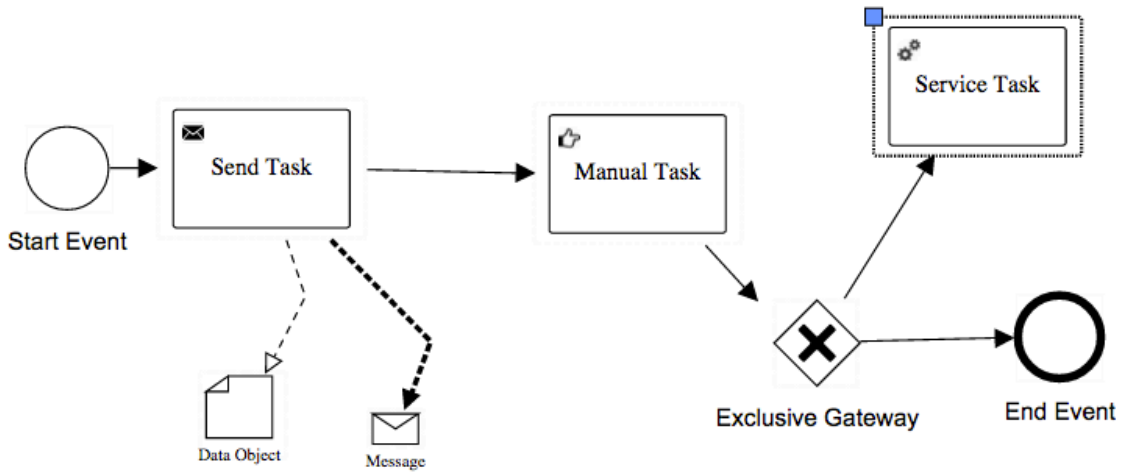


Figura 30 -Elementi BPMN creati con la libreira JointJS

Inoltre sfruttando la gestione degli eventi associati alle vari entità della libreria JointJs, sono stati opportunamente implementate alcune principali funzionalità quali:

- la possibilità di apertura di una barra delle proprietà a destra del diagramma dove l'utente ha la possibilità di monitorare e modificare le informazioni associate ad una specifica entità selezionata,
- la rilevazione dei link che non risultano associati ad alcun elemento target o source; vengono automaticamente colorati di rosso dal sistema in modo da permettere all'utente di effettuare le opportune correzioni.
- Il cambio automatico della tipologia di link (sequenceFlow, messageFlow, objectFlow) come conseguenza della tipologia di oggetto al quale un link risulta collegato.
- la possibilità di spostare gli elementi tra diversi pool, o all'esterno di un pool, mantenendo gli elementi collegati all'ultima entità nella quale sono stati collocati dall'utente.

```
paper.on('blank:pointerdown', function(cellView,evt, x, y) {
                        deselectAllElements();
                $('#newsbox').hide();
        });

graph.on('change:size', function(cell, newPosition, opt) {
        if(cell instanceof joint.shapes.basic.Activity ){

        cell.attr('.fobj/transform',' translate(10,10)');
        cell.attr('div/style/width',cell.get('size').width-30);
        cell.attr('div/style/height',cell.get('size').height-20);
        cell.attr('.fobj/width',cell.get('size').width-20);
          cell.attr('.fobj/height',cell.get('size').height-20);

        }

        if (cell.get('skipParentHandler')) return;
        if (cell.get('embeds') && cell.get('embeds').length) {
                cell.set('originalSize', cell.get('size'));
        }
});
```

Figura 31 - Gestione eventi

B. BackBoneUndo

Backone. Undo.js è una libreria javascript che dipende da Backbone.js e da Underscore.js, che permette di gestire le funzionalità di undo e redo tenendo traccia dei punti di undo attraverso il trigger degli eventi che si verificano. Tale libreria si è rivelata utile per al gestione delle funzionalità di undo/redo. Come è mostrato nella Figura 32 la gestione di tali funzionalità viene realizzata attraverso una prima fase di impostazione del modello al quale si intende applicare un undo ciò si realizza definendo un oggetto di tipo UndoManager al quale è associato l'oggetto attraverso l'attributo register il modello che si intende monitorare e infine si attiva l'ascolto dei trigger che verranno attivati su di esso.
Nelle ultime righe di codice della figura avviene la chiamata alle relative funzioni di undo e redo che gestiscono opportunamente le azioni necessarie.

```
var myUndoManager = new Backbone.UndoManager({     register: [app.mainModel,
app.mainCollection], // pass an object or an array of objects
track: true // changes will be tracked right away });


$(".undo-button").click(function () {       undoManager.undo(); });
$(".redo-button").click(function () {       undoManager.redo(); });
```

Figura 32 - Elementi principali libreria Backbone Undo

Per adattare le funzionalità messa a disposizione dalla librerai alle esigenze dell'editor di workflow sono state sfruttate ulteriori potenzialità che BackboneUndo fornisce. In particolare la libreria permette di annullare le azioni associate agli eventi di tipo : change, add, remove e reset; ma fornisce anche un Api per permettere agli sviluppatori di reimpostare il comportamento associato a tali eventi o di creare dei propri personalizzati.
Relativamente alle finalità dell'editor sviluppato ciò si è reso indispensabile e utile nella gestione dell'undo relativo al movimento degli oggetti in un diagramma.
In particolare è emersa l'esigenza di una personalizzazione del comportamento dell'undo associato al trigget change:position, infatti poiché nel modello di JointJs lo spostamento di un elemento non è punto-punto , ma avviene tenendo traccia di tutti i punti che costituiscono il percorso tra posizione iniziale e finale, è stato opportunamente definito uno specifico undoType in grado di effettuare l'annullamento del cambio di posizione di un elemento mantenendo nello stack dell'undo esclusivamente una singola posizione, perdendo quindi tracci dell'intero cammino seguito(Figura 33).
Analoghe operazioni sono state necessarie per la gestione dell'inserimento e la rimozione di un nuovo elemento nel diagramma nonché per gestire l'annullamento delle operazioni connesse al cambio di tutte le proprietà associate ad un elemento.

```
Backbone.UndoManager.removeUndoType("change");

Backbone.UndoManager.addUndoType("change:isChanging", {
"on": function (model, isChanging, options) {
        if(!addLink){
                if (isChanging) {
                        if (before==null) {
```

```
                    before = model.toJSON();

                }
            } else {
              stored_before=before;
              stored_after=model.toJSON();
              before = null;

             return {
                  "object": model,
                  "before": stored_before,
                  "after": stored_after
                              }
                }
                        }
            },
            "undo": function (model, before, after, options) {
                model.set(before);
                var beforeString=JSON.stringify(before);
                var pos  =beforeString.search("vertices");
                 if(pos<0)
                   model.set("vertices","");
                  model.set("isChanging",false);
                   },

           "redo": function (model, before, after, options) {
                model.set(after);
                model.set("isChanging",false);
                   }
            });
```

Figura 33 - Gestione UndoType

C. Ecore

Ecore.js è una libreria che unisce la meta-modellazione Ecore a Javascript,, ovvero permette di implementare il modello Ecore in javascript. L'utilizzo dello script è subordinato all'import delle librerie underscore.js e sax.js dalle quali dipende.
Gli elementi fondamentali di Ecore sono i packages(EPackage) che contengono entità di tipo EClass, eDataType, EEnum con le rispettive caratteristiche strutturali definiti attraverso EAttribute, EReference. Si considera ad esempio la definizione di un modello relativo ad una Libreria. Il relativo Package sarà definito come mostrato in Figura 34, mentre la Figura 35 mostra come può essere creata l'EClass Library, infine il costrutto EReference permette di creare le relazioni tra le classi (Figura 36)

```
var LibraryPackage = Ecore.EPackage.create({
                                name: 'library',
                                nsPrefix: 'library',
                                nsURI: 'http://www.example.org/library'
                    });
```

Figura 34 - Esempio Package Libreria

```
var Library = Ecore.EClass.create({
                name: 'Library',
               eStructuralFeatures: [
                       Ecore.EAttribute.create({ name: 'name', eType: Ecore.EString });
                       ]
               });
```

Figura 35 - EClass Library

```
var LibraryWriters = Ecore.EReference.create({
                                name: 'writers',
                                upperBound: -1,
                                containment: true,
                                eType: Writer
```

```
});
```

Figura 36 - EReference Esempio

La creazione di istanze specifiche e l'accesso ai relativi attributi viene realizzata attraverso i relativi metodi create, set e get (Figura 37).

```
var myLibrary = Library.create({ name: 'My Library' });
var emfBook = Book.create();
myLibrary.get('items').add(emfBook);
emfBook.set('title', 'EMF Modeling Framework');
emfBook.get('title'); // -> EMF Modeling Framework
```

Figura 37 - Creazione istanze e accesso ai dati

Infine attraverso Ecore.js è possibile gestire e personalizzare gli eventi associati agli EObject ed EList, gli eventi sono inoltre anche disponibili sulle Risorse e su ResourceSet. Un ulteriore funzionalità interessante messa a disposizione dalla libreria è la possibilità di serializzare il modello in Json o in XMI.
All'interno dell'editor il modello Ecore è stato utilizzato per tenere traccia della tipologia di elementi inseriti in un diagramma in un determinato istante. In particolare viene creato un EPackage attraverso il quale si definiscono le entità che può contenere e delle EClass , con le rispettive EReference se presenti, corrispondenti agli elementi di un diagramma BPMN. Tale modello viene utilizzato al momento di aggiunta di un nuovo elemento nel diagramma , in quanto viene creata una nuova istanza dell'oggetto ed aggiunto alla rispettiva lista. La Figura 38 mostra un esempio relativo alla gestione degli oggetti di tipo Activity.

```
var DiagramPackage = Ecore.EPackage.create({
            name: 'diagramPackage',
            nsURI: 'http://www.example.org/sample',
            nsPrefix: 'diagramPackage',
            eClassifiers: [
                        Activity,
                        Gateway,
                        Event,
                        DataObject,
                        Message,
                        Link
            ]
        });


var Activity = Ecore.EClass.create({
            name: 'Activity',
            eStructuralFeatures: [
                Ecore.EAttribute.create({ name: 'name', eType: Ecore.EString }),
                Ecore.EAttribute.create({ name: 'type', eType: Ecore.EString }),
                Ecore.EAttribute.create({ name: 'id', eType: Ecore.EString })
            ]
               });

var Activity_links = Ecore.EReference.create({
                   name: 'links',
                   upperBound: -1,
                   eType: Link
               });
                       Activity.get('eStructuralFeatures').add(Activity_links);

var newActivity = Activity.create({ name: nameActivity ,type:'Task',id: activity.id});
activityList[activityList.length]=newActivity;
```

Figura 38 - Applicazione Ecore nel progetto

D. Jquery UI

La libreria JQueryUI è stata utilizzata per gestire l'interfaccia utente , gli effetti dell'editor web. In particolare si è reso fondamentale per la gestione del Drag and Drop degli elementi dalla palette all'area di disegno(vedi Figura 39)

```
$( "#draggActivity" ).draggable(
    {
                helper : "clone",
                stop:function(event,ui)
                {
                setPosition(event,ui);
                addActivity(xPos,yPos,"First activity");
                }
  });

$( "#content" ).droppable({
      drop: function( event, ui ) {
        $( this )
      }
});
```

Figura 39 - Gestione Drag and Drop

Inoltre sono stati utilizzati i Widgets Accordion e Dialog messi a disposizione dalla Libreria per gestire rispettivamente il menù della palette posizionato a sinistra dell'editor, le finestre di popup che mostrano informazioni all'utente o per inserimento di ulteriori dati relati al diagramma che si intende definire

E. XMl Writer

È una libreria javascript che permette di creare file in formato xml gestendo opportunamente la gerarchia tra gli elementi e i relativi attributi. Ai fini del progetto si è reso utile il suo utilizzo per la generazione dei file contenente la definizione xmi del diagramma di workflow creato, in aderenza alla specifica OGM per BPMN.

F. FileSaver

Libreria Javascript che permette di realizzare il salvataggio di file lato client, utile per le applicazioni web che hanno la necessità di generare file, o per la memorizzazione ad esempio di dati sensibili che non devono essere inviati ad un server esterno. Attraverso la libreria è stata implementata la funzionalità relativa l salvataggio in locale del file contenente la definizione di un diagramma di workflow creato al fine di poterlo caricare in seguito nell'editore per apportate ad esempio modifiche.

ARCHITETTURA DEL SISTEMA REALIZZATO

Il principale pattern di riferimento sul quale si è basata la realizzazione del tool considerato è MVC (MODEL–VIEW-CONTROL). Infatti grazie alle scelte implementative realizzate si è resa semplice stabile un isolamento tra i dati di business (Modelli) rispetto alle interfacce utente

(Viste) nonché la distinzione e gestione separata dei componenti che gestiscono la logica di business (Controllori) nonché tuti gli input provenienti dall'utente. La maggior parte delle librerie utilizzate utilizzano principalmente la libreria Backbone che permette di strutturare le applicazioni web secondo le caratteristiche del pattern MVC. Nel seguito di questo paragrafo verrà descritto con maggiore dettaglio quali sono le parti principali in cui possibile scomporre il sistema realizzato al fine di individuare i principali elementi costitutivi del pattern MVC.

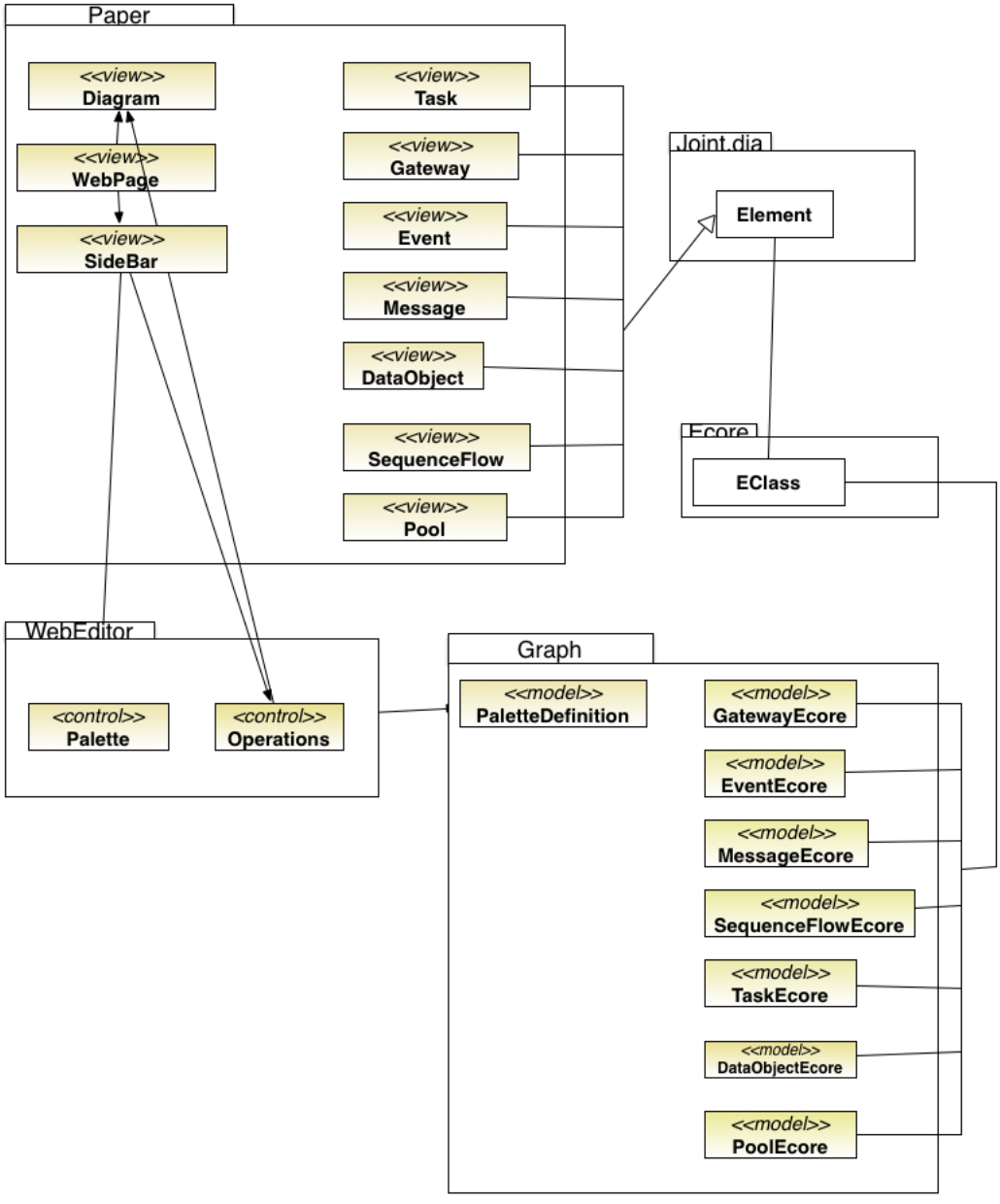


Figura 40 - Diagramma delle classi generale dell'applicazione

A. Models

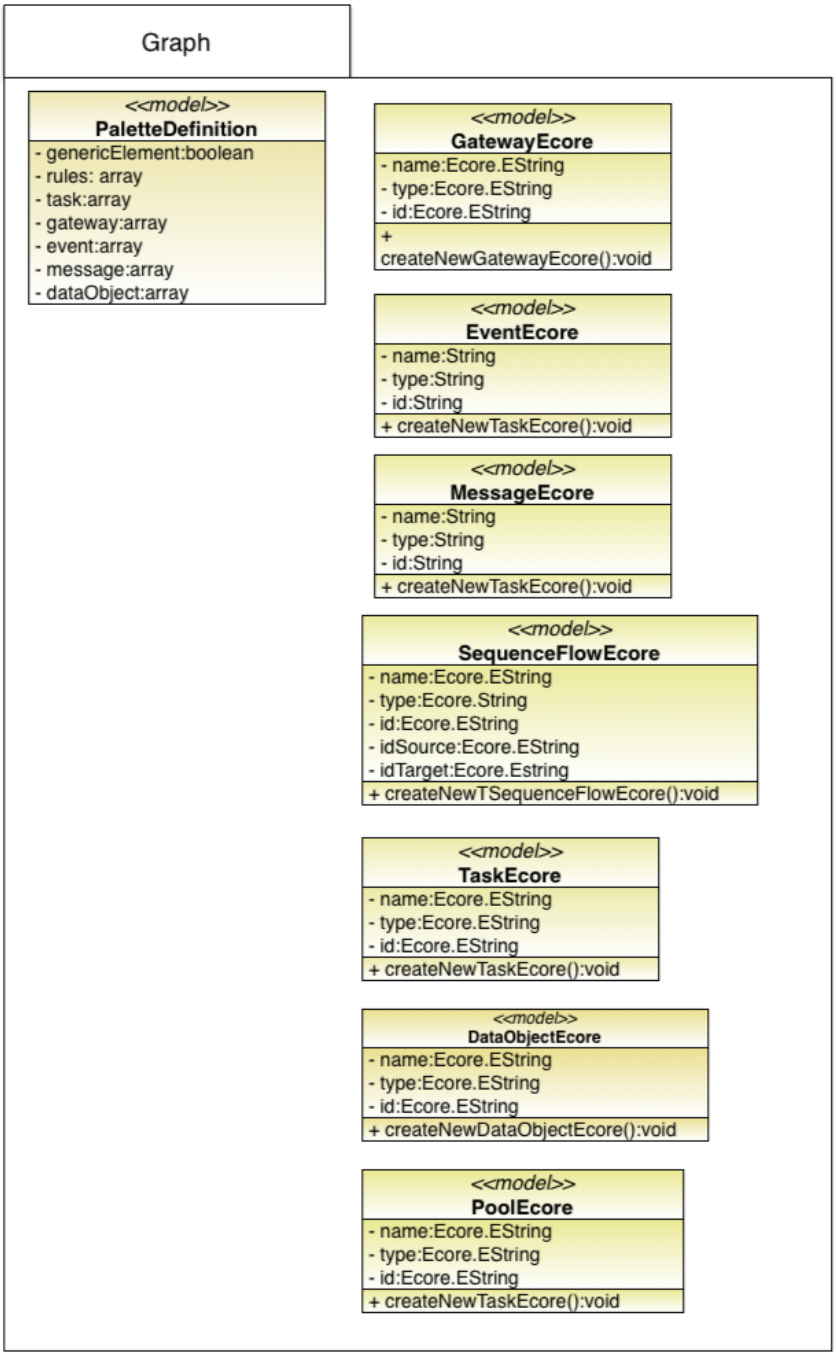


Figura 41 - Diagramma delle classi Model

Il modello permette di gestire i dati di un applicazione, nel nostro caso la sezione model è stata realizzata attraverso l'utilizzo della libreria *Ecore.js.* La libreria già definita nei precedenti paragrafi ha permesso di creare un modello . A titolo esemplificativo si consideri qui di seguito un esempio di definizione del modello di un elemento Gateway.

```
var Gateway = Ecore.EClass.create({
        name: 'Gateway',
        eStructuralFeatures: [
          Ecore.EAttribute.create({ name: 'name', eType: Ecore.EString }),
          Ecore.EAttribute.create({ name: 'type', eType: Ecore.EString }),
          Ecore.EAttribute.create({ name: 'id', eType: Ecore.EString })
        ]
      });
```

Figura 42 - Esempio modello ecore elemento gateway

Documentazione delle classi

PaletteDefinition

E' la classe principale del package Graph. Si tratta della che istanzia la definizione della palette che permette di inserire gli elementi grafici presenti nello stencil set, che l'utente può inserire nel diagramma.

Attributi della classe

- ***private boolean genericElement***

Questo attributo permette di stabilire se inserire degli elementi standard del BPMN all'interno della della bassa degli strumenti. Se settato a true gli elementi verranno automaticamente caricati nella palette in caso contrario verranno inibiti.

- ***private array rules***

Questo elemento contiene l'array che definisce le regole semantiche che devono essere rispettate all'interno del diagramma affinché esso si ritenuto corretto

- ***private array task***

Questo elemento contiene l'array che definisce tutti gli elementi di tipo task e le loro relative caratteristiche che devono essere ineriti nel menù laterale.

- ***private array gateway***

Questo elemento contiene l'array che definisce tutti gli elementi di tipo gateway e le loro relative caratteristiche che devono essere ineriti nel menù laterale.

- ***private array event***

Questo elemento contiene l'array che definisce tutti gli elementi di tipo event e le loro relative caratteristiche he devono essere ineriti nel menù laterale.

- ***private array message***

Questo elemento contiene l'array che definisce tutti gli elementi di tipo message e le loro relative caratteristiche che devono essere ineriti nel menù laterale.

- ***private array dataObject***

Questo elemento contiene l'array che definisce tutti gli elementi di tipo dataObject e le loro relative caratteristiche che devono essere ineriti nel menù laterale

TaskEcore

E' una classe del package Graph. Si tratta della classe che contiene la definizione del modello corrispondente all'elemento grafico task, secondo il meta-modello Ecore su cui si basa

Attributi della classe

- ***private EString name:***

Questo attributo permette di definire il nome associato al task

- ***private EString type***

Questo attributo permette di definire il tipo associato ad un task

- ***private EString id***

Questo attributo mantiene l'informazione relativa all'id associato ad un task

Metodi della classe

- ***public void createNewTaskEcore()***

Questo metodo permette di istanziare di inizializzare un nuovo oggetto di tipo task ecore

B. Views

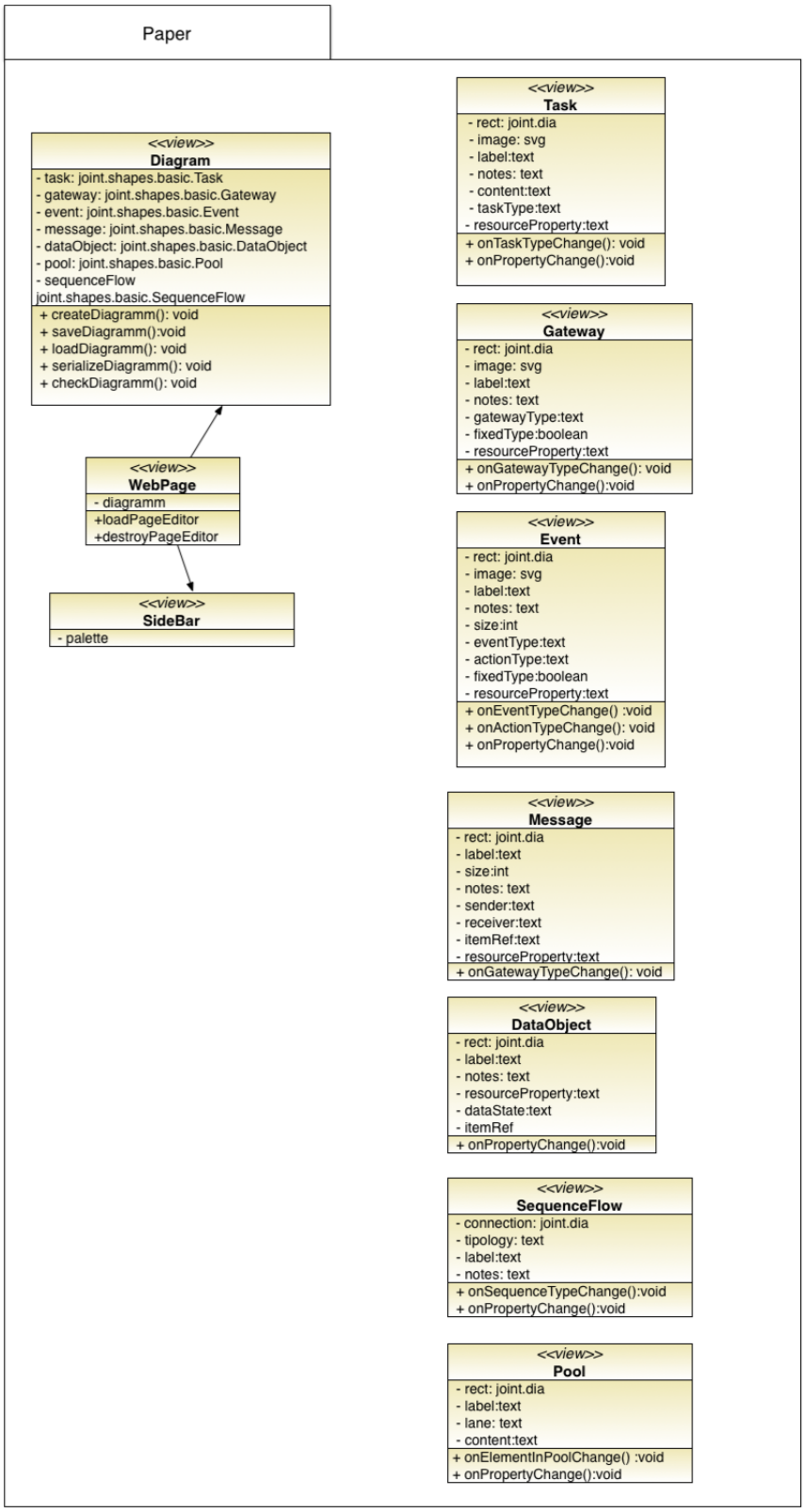


Figura 43 - Diagramma delle classi view

Le viste sono la rappresentazione visiva del modello che presentano una visualizzazione filtrata del loro stato corrente, in javascript le viste sono relative alla costruzione e al mantenimento degli elementi del DOM. gli utenti sono in grado di interagire soltanto con la vista attraverso la lettura e la modifica dei valori degli attributi legati al modello.: poiché la

vista è il livello di presentazione , si è reso necessario renderne la modifica e l'aggiornamento user-friendly. Nel tool le viste interagiscono con l'utente attraverso la barra delle proprietà posta a destra dell'editor, questa si attiva alla selezione del rispettivo elemento presente nel diagramma. Inoltre altre viste sono state utilizzate nella definizione degli elementi grafici del diagramma per mezzo della libreria JointJS, infatti come si può vedere in Figura 44 ciascun elemento è stato opportunamente codificato per permetterne una corretta definizione grafica nell'area di lavoro.

```
//Message
joint.shapes.basic.Message = joint.dia.Element.extend({
      markup: '<g class="rotatable"><g class="scalable"><rect /><polygon    class="body
      inner"/></g><text class="label"/></g>',
      defaults: joint.util.deepSupplement({

            type: 'basic.Message',
            size: {
                  width: 70,
                  height: 50
               },
            attrs: {
               '.body': {
                  points: '0,0 60,0 60,40 0,40 0,0 60,0 30,0 0,0'
                  stroke: '#000000',
                  fill: '#ffffff'
                   },
               'rect': { fill: 'white',
                stroke: 'white',
               'follow-scale': true,
                width: 65,
                height: 45 ,
                  magnet:true
                 },

               '.inner': {
                  transform: 'scale(0.9,0.9) translate(5,5)',
                  },
               '.label': {
                   ref: '.body',
                   'ref-x': .5,
                   'ref-dy': 5,
                   text: '',
                   'text-anchor': 'middle'
                 },
                '.notes':{text:''},
                '.sender':{text:'Default Sender'},
                '.receiver':{text:'Defalut Receiver'},
                '.resourceProperty':{text:''},
                '.itemRef':{text:''}
                   }

             }, joint.dia.Element.prototype.defaults)
});
```

Figura 44 - Vista JointJS relativa ad elemento Message

Documentazione delle classi

Diagram

E' la classe principale del package Paper. Si tratta della classe che contiene la definizione degli elementi grafici principali che compongono lo stencil set nonché permete di gestire le principali interazioni e funzionalità dell' editor durante la realizzazione di un workflow.

Attributi della classe

- ***task: joint.shapes.basic.Task***

Variabile contenente la definizione dell'elemento task per la sua rappresenzione grafica all'interno del workflow

- ***gateway: joint.shapes.basic.Gateway***

Variabile contenente la definizione dell'elemento gateway per la sua rappresenzione grafica all'interno del workflow

- ***event: joint.shapes.basic.Event***

Variabile contenente la definizione dell'elemento event per la sua rappresenzione grafica all'interno del workflow

- ***message: joint.shapes.basic.Message***

Variabile contenente la definizione dell'elemento message per la sua rappresenzione grafica all'interno del workflow

- ***dataObject: joint.shapes.basic.DataObject***

Variabile contenente la definizione dell'elemento dataObject per la sua rappresenzione grafica all'interno del workflow

- ***pool: joint.shapes.basic.Pool***

Variabile contenente la definizione dell'elemento pool per la sua rappresenzione grafica all'interno del workflow

- ***sequenceFlow joint.shapes.basic.SequenceFlow***

Variabile contenente la definizione dell'elemento sequenceFlow per la sua rappresenzione grafica all'interno del workflow

Metodi della classe

- ***public void createDiagramm()***

Questo metodo gestisce la creazione di un nuovo diagramma, inizializza tutti gli elementi grafici presenti nel file di definizione della palette e li inserisce correttamente all'interno della barra alterale degli strumenti.

- ***public file saveDiagramm()***

Questo metodo gestisce il salvataggio di un diagramma creato dall'utente. All'interno del metodo viene prima effettuato un controllo sulla validità semantica e sintattica del diagramma creato attraverso l'invocazione della funzione *checkDiagramm()*. Al file json corrispondente alla definizione grafica del diagramma vengono aggiunte le informazioni relative agli elementi Items e Resources che sono stati definiti dall'utente.

- ***public void loadDiagramm()***

Questo metodo permette di caricare un diagramma precedentemente creato all'interno dell'area di lavoro al fine di apportare delle modifiche. Al caricamento del file vengono rilevati tutti gli elementi grafici che costituiscono il workflow, nonché la loro reciproca relazione. Al caricamento di un file contenente la definizione del diagramma inoltre vengono memorizzate le informazioni relative agli items e alle resources definiti. Infine viene effettuato il parse file json contenente la definizione del diagramma affinche venga opportumente converito in un oggetot di tipo graph di JoinJS.

- ***public void serializeDiagramm()***

Questa funzione effettua il parser del diagramma definito attraverso la libreria grafica JointJS nella corrispondente definizione XMI secondo quanto definito dallo standard OGM per BPMN. Viene quindi creato un opportuno file xml all'interno del quale vengono inseriti i tag relativi a tutti gli elementi grafici che compongono il diagramma. Infine il file xml generato viene salvato in un file di testo creato nel file system dell'utente.

- ***public void checkDiagramm()***

Questa funzione validare un diagramma sia da un punto di vista sintattico che semantinco. Il controllo sintattico viene effettuato sulla base delle principali proprietà sintattiche che deve rispettare un diagramma BPMN secondo lo standard OGM(quali ad

esempio verifica che tutti i link inseriti abbiano un source e un target associato), tutti gli eventuali errori che vengo rilevati vengono inseriti in una messaggio e gli elementi grafici coinvolti vengono evidenziati in rosso all'interno del diagramma. In seguito viene effettuato il controllo semantico in base alle regole definite nel file di definizione della palette; in particolare vengono ad esempio verificati che non siano stati realizzati tutti i collegamenti inibiti tra due task all'interno di un diagramma. Gli elementi che sono interessati da errori semantici vengono evidenziati in blu; tutti gli errori semantici rilevati vengono raccolti e sommati agli errori sintattici per esser presentati all'utente

C. Controllers

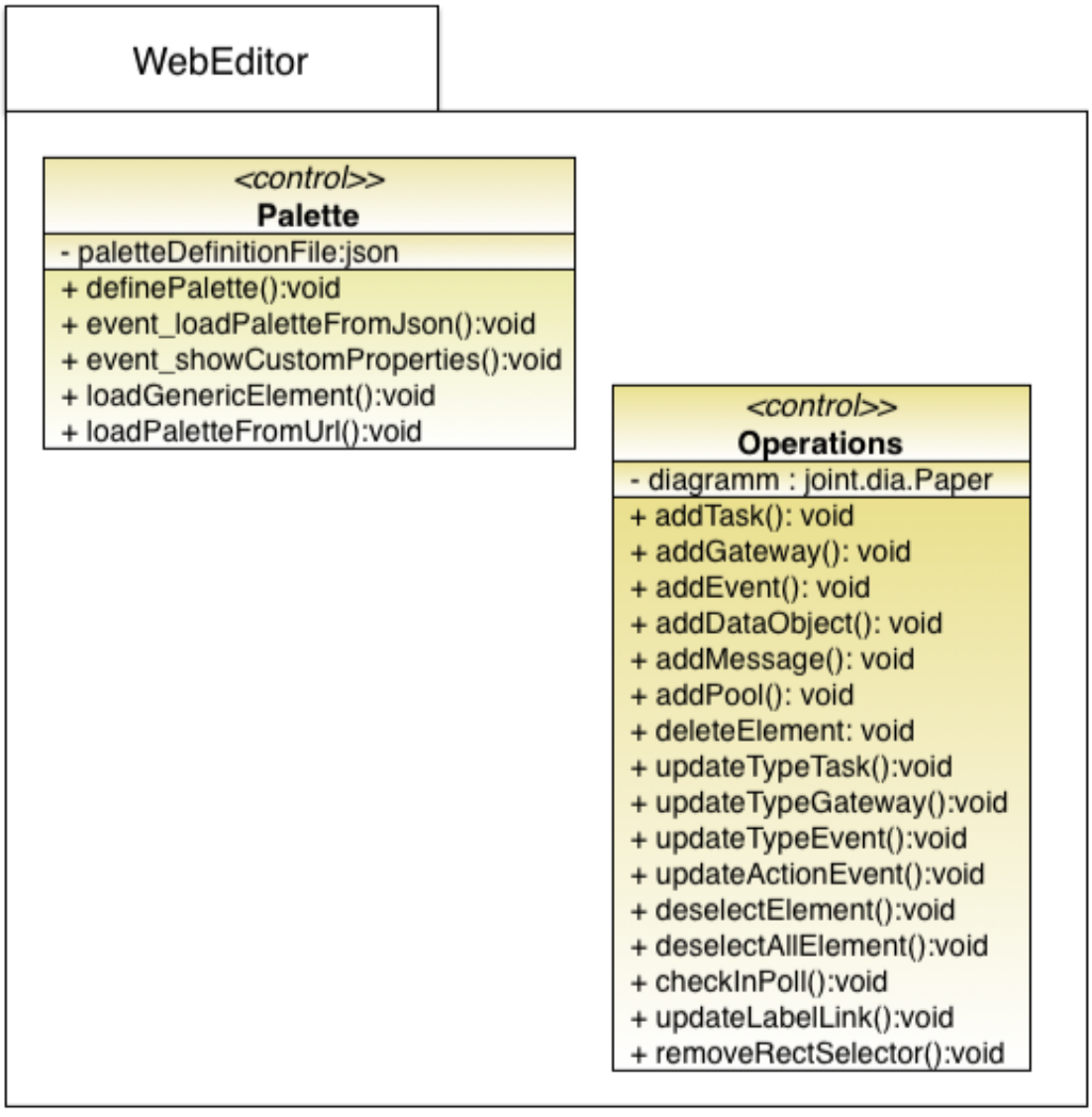


Figura 45 - Diagramma delle classi control

I Controllers sono intermediari tra i modelli e le viste, ovvero sono generalmente responsabili dell'aggiornamento del modello quando l'utente manipola la vista.
Nell' applicazione sviluppata i controllers sono responsabili della gestione del cambiamento dei valori che popolano il pannello delle proprietà, infatti i campi di detto form saranno opportunamente riempiti con le proprietà associate ad corrispondente elemento selezionato. La funzione di controller viene inoltre esplicata attraverso la gestione opportuna degli eventi che vengono registrati sul diagramma. Si consideri ad esempio la **Figura 46** che mostra

l'opportuna gestione associata all'aggiunta di una nuova *sequenceFlow* nel diagramma, un opportuno controller verifica che esso abbia sia un source che un target associati, se ciò non si verifica allora la *sequenceFlow* sarà disegnata di colore rosso per evidenziare all'utente la presenza di un errore sintattico.

```
graph.on('add',function(cell)
                                {

if(cell instanceof joint.dia.Link ){

                                linkList[linkList.length]=cell;
                                addLink=true;
                                endtarget=cell.get('target');
                                endsource=cell.get('source');
                                 cell.label(0, { attrs: { text: { 'font-size':15} } });


if(endtarget.id !== undefined && endsource.id!=undefined && endsource!==endtarget || endtarget
instanceof joint.shapes.basic.Pool)
        {

                        cell.attr({
                                '.marker-target': {fill: 'black',d: 'M 10 0 L 0 5 L 10 10 z'},
                                '.connection' : { stroke: 'black' },

                                });


                                }
                else{
                                console.log("LINK RED")

                                cell.attr({
                                        '.marker-target': {fill: 'red',d: 'M 10 0 L 0 5 L 10 10 z'},
                                        '.connection' : { stroke: 'red' },

                                });
                }
```

Figura 46 - Gestione eventi esempio creazione sequenceFlow

Documentazione delle classi

Operations

E' una classe del package Graph. Si tratta della classe che contiene la definizione del modello corrispondente all'elemento grafico task, secondo il meta-modello Ecore su cui si basa

Attributi della classe

- ***private JointJS:Graph diagramm:***

Questo contiene la definizione del diagramma JointJS editato dell'utente.

Metodi della classe

- ***public void addTask()***

Questo metodo permette di istanziare un nuovo oggetto di tipo task all'interno del diagramma, che viene attivato dall'utente attraverso il drag and drop.

- ***public void deleteElement***

Questo metodo gestisce l'eliminazione di un oggetto selezionato all'interno del diagramma

- ***public void updateTypeTask()***

Questo metodo gestisce l'aggiornamento della tipologia di un task a seconda di quella scelta dall'utente

- ***public void deselectElement()***

Questo metodo rimuovere il riquadro di selezione relativo ad un elemento nel diagramma, nel caso in cui non sia più necessario renderlo selezionato

- ***public void deselectAllElement()***

Questo metodo rimuovere i riquadri di selezione relativo da tutti gli elementi presenti nel diagramma

- ***public void checkInPoll()***

Questo metodo verifica se un elemento grafico è stato inserito all'interno dell'elemento BPMN Pool, in modo da poter gestire opportunamente la gerarchia di parents che li lega.

- ***public void updateLabelLink()***

Questo metodo permette di aggiornare l'etichetta associata d un link nel diagramma

- ***public void removeRectSelector()***

Questo metodo permette di rimuovere il rettangolo di selezione che permette di effettuare il ridimensionamento di un elemento grafico, presente nell'angolo superiore sinistra

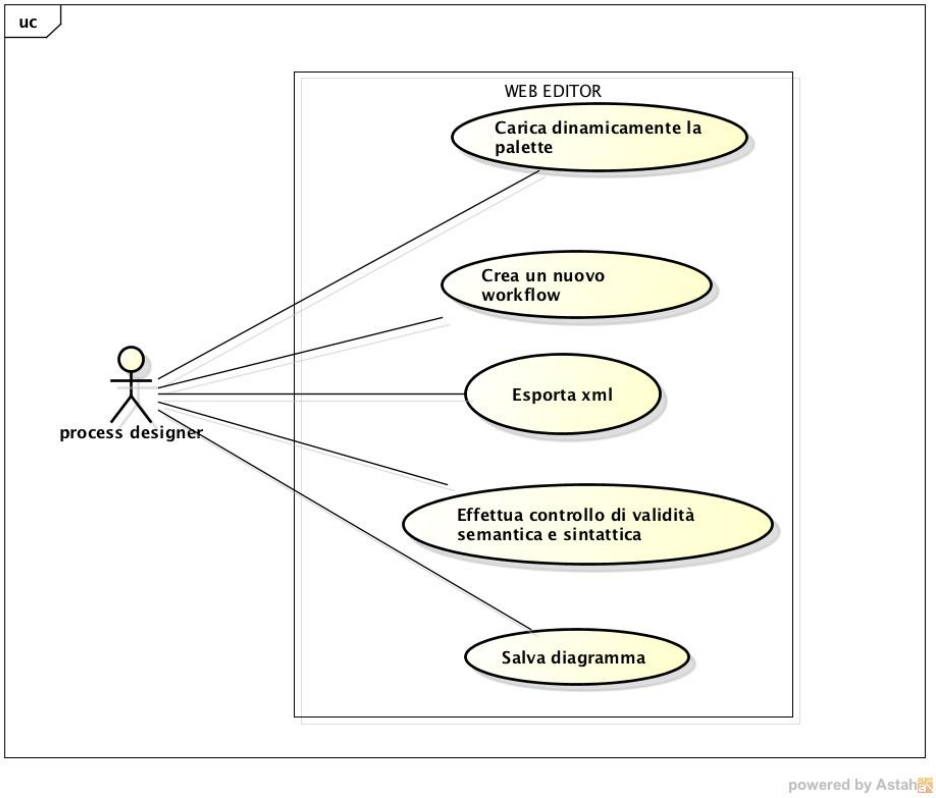


Figura 47 - Diagramma dei Casi D'uso

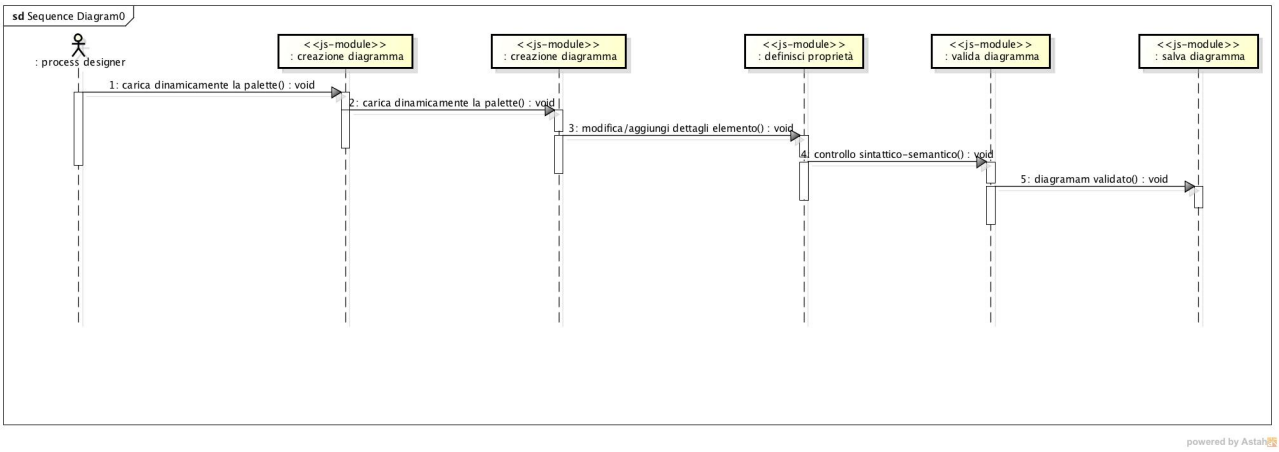


Figura 48 - Diagramma di Sequenza

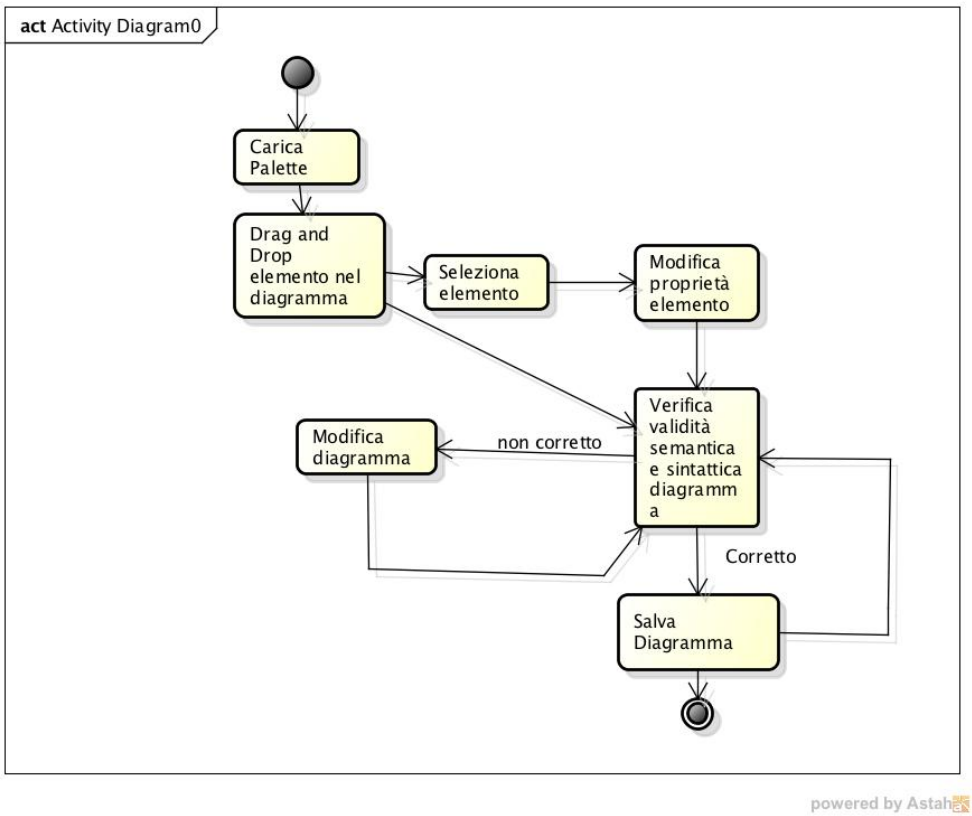


Figura 49 - Diagramma delle Attività

LE FUNZIONALITA' DI EDITING DEL TOOL

Il tool per la definizione di un processo secondo lo standard BPMN che è stato progettato fornisce all'utente la possibilità di definire un nuovo diagramma di workflow attraverso l'interfaccia web mostrata in **Figura 50**.
Come si può facilmente intuire attraverso il drag&drop degli elementi che compongono la palette a sinistra sull'area di lavoro centrale si avrà la possibilità di definire e modificare le relative proprietà di ogni elemento selezionato visualizzabili nella property tab collocata a destra dell'interfaccia
Attraverso i pulsanti presenti nella barra degli strumenti l'utente ha la possibilità di creare un nuovo diagramma, salvare il file contenente la definizione del diagramma generato oppure recuperare un workflow precedentemente realizzato per apportarvi delle modifiche; gli ulteriori pulsanti presenti nella barra degli strumenti premettono rispettivamente di realizzare l'export del diagramma in formato XML aderente alle specifiche BPMN.

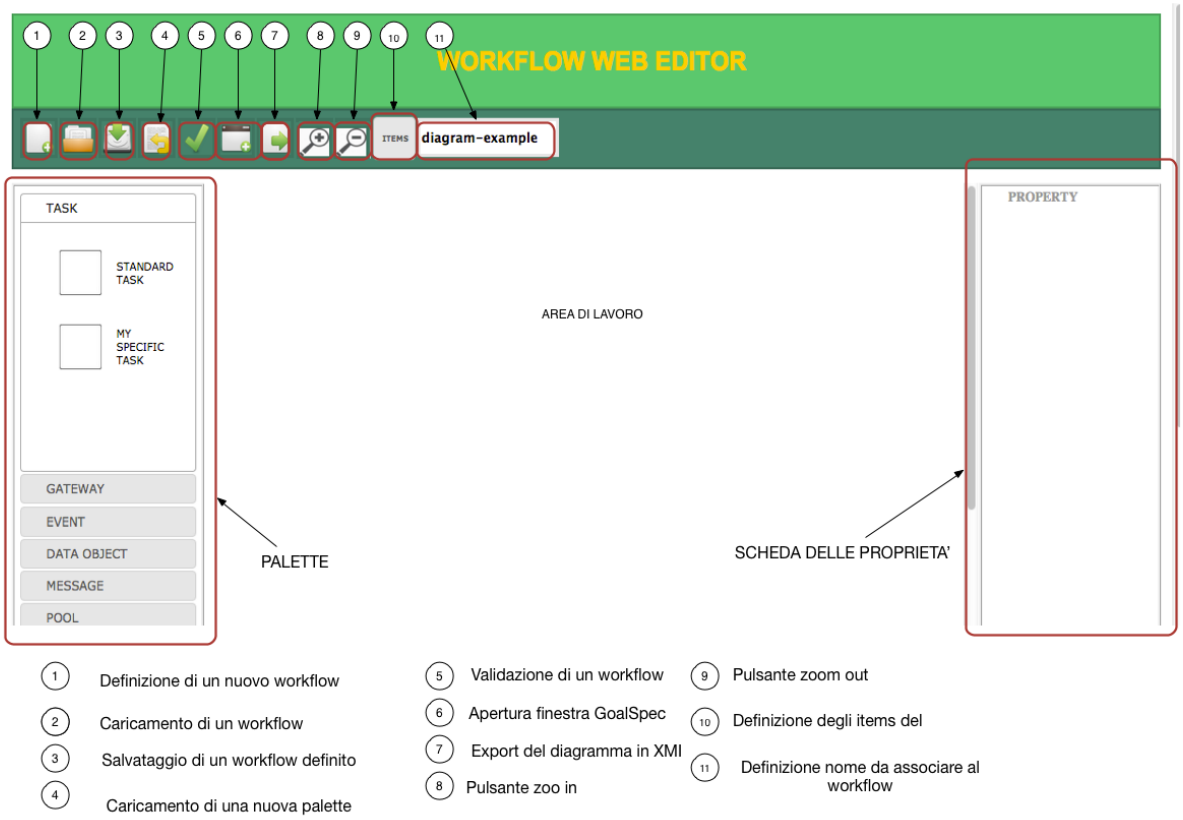


Figura 50 - Interfaccia Web Editor

La barra laterale posta a destra dell'interfaccia si attiva alla selezione di un elemento nel diagramma e permette all'utente di monitorare e modificare tutte le principali proprietà associate alla relativa entità. La selezione di un elemento permette inoltre di personalizzarne le dimensioni (larghezza-altezza) muovendo l'icona corrispondente al rettangolo posto nell'angolo superiore sinistro del relativo elemento. La palette contiene inoltre tutti i principali elementi per la definizione di un diagramma BPMN(Task, Gateway, Event, Data Object Message, Pool); inoltre in aderenza alle specifiche OMG l'utente ha la possibilità di impostare e gestire le varie tipologie di event, gateway e task che un diagramma BPMN può prevedere, ciò si realizza selezionando il tipo desiderato attraverso una select box presente nella barra delle proprietà, la scelta effettuata dall'utente genera nel diagramma il cambiamento dell'elemento grafico associato alla scelta, in particolare i task saranno completati con le rispettive icone che ne descrivono la tipologia cosi come impone lo standard BPMN (**Figura 52**).

Figura 51 - Proprietà asscociate agli elementi

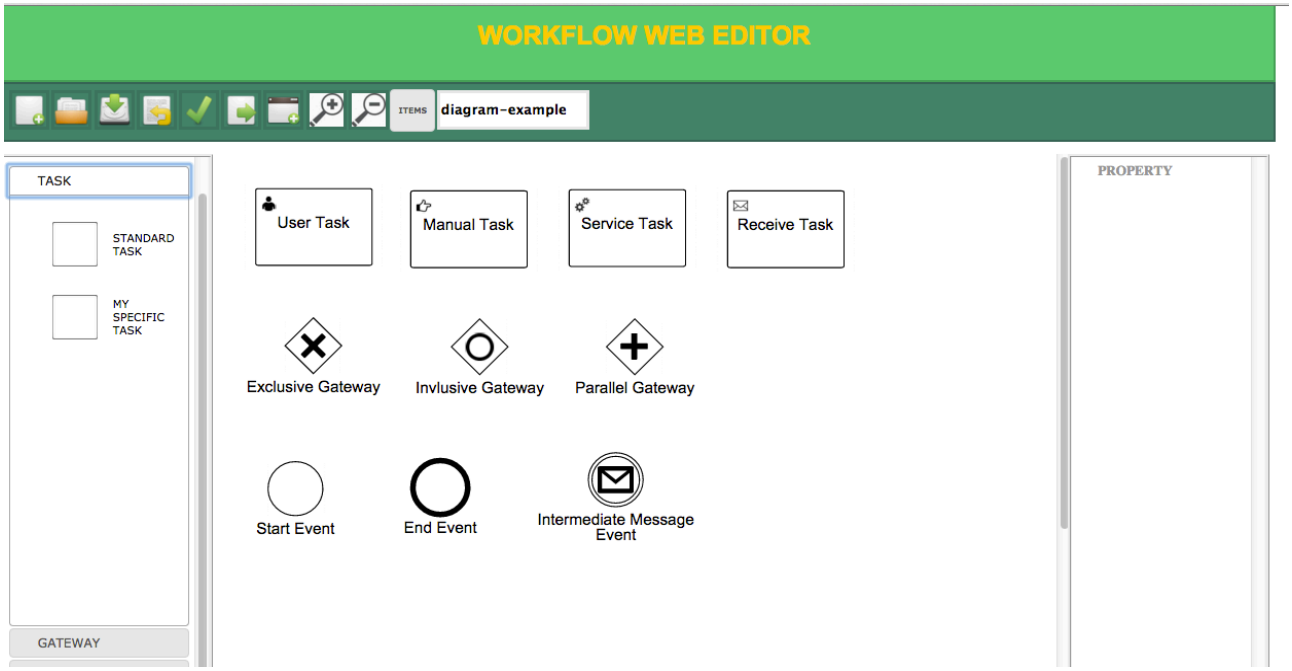


Figura 52 - Esempi di tipologia di elementi BPMN

L'interoperabilità del tool realizzato con gli altri editor bpmn è stata garantita grazie una specifica funzione che permette di poter esportare il diagramma in un formato XMI compatibile con le specifiche BPMN di OMG, ciò rendere quindi il tool aderente alle specifiche standard. Attraverso il relativo pulsante della barra degli strumenti viene generato un file xml, la **Figura 53** mostra il file xml relativo al diagramma mostrato in **Figura 54**.

```
<?xml version="1.0" encoding="MacRoman" standalone="no"  ?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:tns="http://sourceforge.net/bpmn/definitions/_1419237461760" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:yaoqiang="http://bpmn.sourceforge.net" exporter="" exporterVersion="2.2.5 (GPLv3, Non-
Commercial)" expressionLanguage="http://www.w3.org/1999/XPath" id="_1419237461760" name=""
targetNamespace="http://sourceforge.net/bpmn/definitions/_1419237461760" typeLanguage="http://www.omg.org/spec/BPMN/20100524/MODEL
http://bpmn.sourceforge.net/schemas/BPMN20.xsd">
        <collaboration id="COLLABORATION_0" isClosed="false" />
        <process id="PROCESS_0" isClosed="false" isExecutable="true" processType="None">
            <sequenceFlow id="_14" sourceRef="_1" targetRef="_0" name="" />
            <sequenceFlow id="_15" sourceRef="_0" targetRef="_13" name="" />
            <sequenceFlow id="_16" sourceRef="_13" targetRef="_2" name="yes">
                <conditionExpression>yes</conditionExpression>
            </sequenceFlow>
            <sequenceFlow id="_17" sourceRef="_13" targetRef="_3" name="no">
                <conditionExpression>no</conditionExpression>
            </sequenceFlow>
            <sequenceFlow id="_18" sourceRef="_4" targetRef="_6" name="" />
            <sequenceFlow id="_19" sourceRef="_2" targetRef="_4" name="sac">
                <conditionExpression>sac</conditionExpression>
            </sequenceFlow>
            <sequenceFlow id="_20" sourceRef="_3" targetRef="_5" name="" />
            <sequenceFlow id="_21" sourceRef="_3" targetRef="_7" name="" />
            <sequenceFlow id="_22" sourceRef="_7" targetRef="_8" name="Late delivery">
                <conditionExpression>Late delivery</conditionExpression>
            </sequenceFlow>
            <sequenceFlow id="_23" sourceRef="_5" targetRef="_10" name="undeliverable">
                <conditionExpression>undeliverable</conditionExpression>
            </sequenceFlow>
            <sequenceFlow id="_24" sourceRef="_8" targetRef="_9" name="" />
            <sequenceFlow id="_25" sourceRef="_10" targetRef="_11" name="" />
            <sequenceFlow id="_26" sourceRef="_11" targetRef="_12" name="" />
            <task completionQuantity="1" id="_0" isForCompensation="false" name="Check availability" startQuantity="1">
                <incoming>_14</incoming>
                <outgoing>_15</outgoing>
            </task>
            <startEvent id="_1" name="Order Receiived" parallelMultiple="false" isInterrupting="true">
                <outgoing>_14</outgoing>
                <messageEventDefinition id="_1_EM_1" />
            </startEvent>
            <task completionQuantity="1" id="_2" isForCompensation="false" name="Ship article" startQuantity="1">
                <incoming>_16</incoming>
                <outgoing>_19</outgoing>
            </task>
            <task completionQuantity="1" id="_3" isForCompensation="false" name="Procurement" startQuantity="1">
                <incoming>_17</incoming>
                <outgoing>_20</outgoing>
                <outgoing>_21</outgoing>
            </task>
            <task completionQuantity="1" id="_4" isForCompensation="false" name="Financial settlement" startQuantity="1">
                <incoming>_19</incoming>
                <outgoing>_18</outgoing>
            </task>
            <startEvent id="_5" name="" parallelMultiple="false" isInterrupting="false">
                <incoming>_20</incoming>
                <outgoing>_23</outgoing>
                <errorEventDefinition   id="_5_EER_1" />
            </startEvent>
            <endEvent id="_6" name="Payment received">
                <incoming>_18</incoming>
                <terminateEventDefinition    id="_6_ET_1" />
            </endEvent>
            <startEvent id="_7" name="" parallelMultiple="false" isInterrupting="false">
                <incoming>_21</incoming>
                <outgoing>_22</outgoing>
                <escalationEventDefinition  id="_7_ET_1" />
            </startEvent>
            <task completionQuantity="1" id="_8" isForCompensation="false" name="Inform customer" startQuantity="1">
                <incoming>_22</incoming>
                <outgoing>_24</outgoing>
            </task>
            <endEvent id="_9" name="Customer informed">
                <incoming>_24</incoming>
            </endEvent>
            <task completionQuantity="1" id="_10" isForCompensation="false" name="Inform customer" startQuantity="1">
                <incoming>_23</incoming>
                <outgoing>_25</outgoing>
            </task>
            <task completionQuantity="1" id="_11" isForCompensation="false" name="Remove article from calatogue" startQuantity="1">
                <incoming>_25</incoming>
                <outgoing>_26</outgoing>
            </task>
            <endEvent id="_12" name="Article removed">
                <incoming>_26</incoming>
            </endEvent>
            <exclusiveGateway id="_13" name="Article available" gatewayDirection="Diverging">
                <incoming>_15</incoming>
```

```
                    <outgoing>_16</outgoing>
                    <outgoing>_17</outgoing>
                </exclusiveGateway>
            </process>
            <collaboration id="COLLABORATION_1" isClosed="false" />
</definitions>
```

Figura 53 - File Xml relativo ad un diagramm BPMN

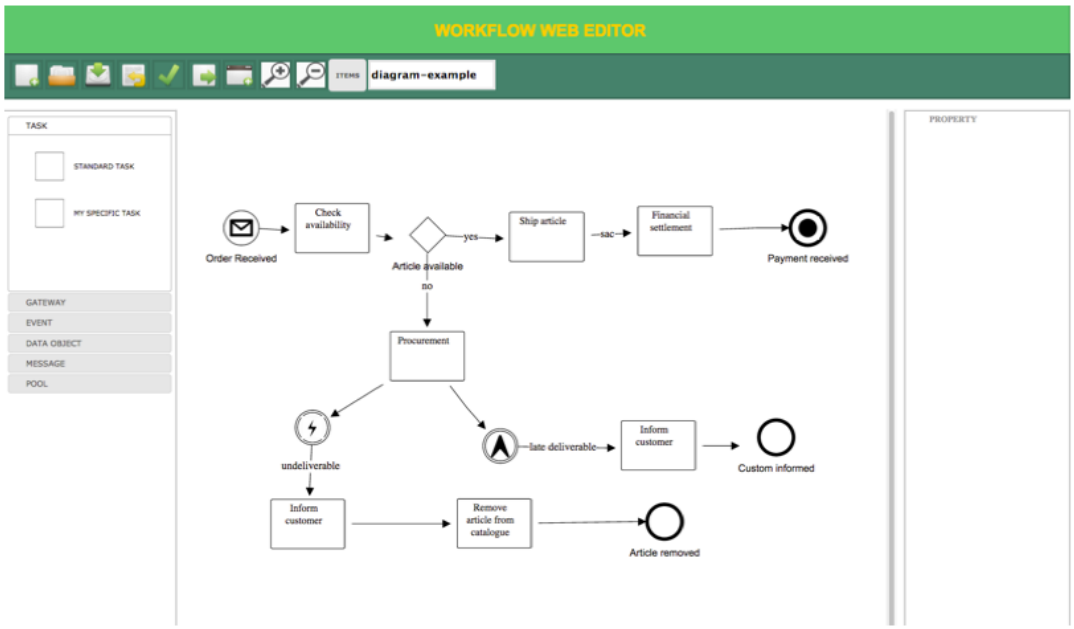


Figura 54 - Esempio WorkFlow Processo Ordine

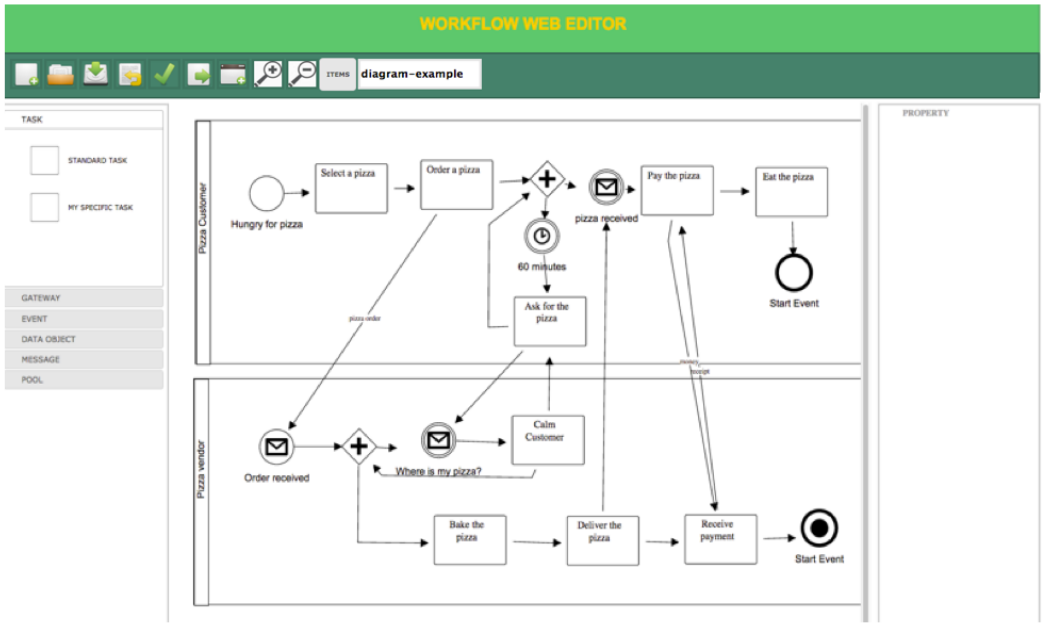


Figura 55- Esempio workflow Processo Ordinazione Pizza

LE FUNZIONALITA' AVANZATE

Al fine di dotare il sistema realizzato dei requisiti che permettano di realizzare l'interoperabilità con il middleware MUSA si è reso necessario implementare delle ulteriori funzioni che estendono e completano quelle di base precedentemente definite. A tale scopo sono stati introdotti ulteriori pulsanti nella barra geli strumenti ciascuno dei quali corrisponde a delle ben precise funzioni implementative. Le principali operazioni che è possibile realizzare attraverso l'interazione con la barra degli strumenti sono i seguenti:

- possibilità di generare in maniera dinamica tutti gli elementi che compongono la palette, attraverso il caricamento di un file di definizione in formato json nel quale è possibile impostare le regole di composizione semantica degli elementi grafici
- realizzazione del controllo di validità sintattica e semantica del workflow realizzato con visualizzazione grafica di tutti gli errori e i warning individuati, nonché descrizione testuale delle motivazioni di tali errori
- possibilità di esportare il processo di business definito all'interno dell'area di lavoro del web editor nel formato xmi definito dallo standard OGM
- capacità di ottenere la definizione in linguaggio GoalSpec corrispondente al processo di business realizzato
- injection della specifica goalSpec nel middleware MUSA

Un maggiore definizione e illustrazione della realizzazione di tali funzionalità verrà mostrata nel seguito di questo paragrafo

A. Generazione dinamica degli elementi costitutivi della palette

Attraverso il pulsante 4 mostrato in **Figura 50** è possibile importare nel framework la definizione di una nuova palette. Ciò permetterà di poter definire uno specifico workflow i cui elementi costitutivi sono esclusivamente quelli definiti nel file di configurazione che al caricamento popoleranno la palette posta a sinistra dell'editor. È necessario che il file di configurazione della palette abbia una struttura ben definita al fine di poter garantire la corretta esecuzione della funzionalità, vedremo nel seguito con maggiore dettaglio quali sono gli elementi necessari per la definizione di una palette e quale debba essere il formato json del file di configurazione.

Il file si compone di alcuni elementi principali che necessitano di essere definiti opportunamente questi sono:

- *genericElement :* booleano che impostato a *true* permette di avere nella palette anche gli elementi principali definiti nello standard BPMN[1] di OMG altrimenti il valore *false* inibirà la loro presenza tra gli elementi della palette e quindi il loro utilizzo nel workflow definito.
- *rules:* array di oggetti di tipo regole ciascuna delle quali definita dai seguenti attributi
    - *idCapability:* identificativo corrispondenti all'elemento per i quali si vuole inibire il collegamento.
    - *idcapabilityProibitionAfter:* identificativo univoco corrispondente all'elemento per il quale non ammesso che esso segua l'elemento individuato attraverso il campo idCapability nella definizione di un workflow
    - *description:* campo di testo attraverso il quale può essere descritta la motivazione che ha indotto a definire tale regola di composizione.

Si noti che qualora sia stata definita una regola che inibisce il collegamento tra due task , allora il framework considererà non validi tutti i percorsi che collegano i tue task non direttamente tra loro.

- *task:* attraverso questo identificativo è possibile definire un array di elementi di tipo task ciascuno dei quali risulta definito attraverso i seguenti elementi:

    - *typeId*: identificativo univoco del task
    - *name:* nome testuale associato al task che verrà ad esso associato sia quando verrà caricato nella palette laterale che quando sarà posto nell'area di lavoro attraverso il drag&drop.
    - *iconClass:* identificativo della classe associata all'elemento che permette di definire l'icona grafica che si vuole associare al task definito
    - *url:* url associata all'elemento che permette di aprire un pop-up ad esso relativo dove è possibile visualizzare maggiori informazioni ad esso associate.
    - *exiteCodes:* array contenente la definizione testuale delle condizioni possibili di uscita associabili al task. Questo tipo di informazione è utilizzata nel momento in cui una nuova istanza del task viene creata nel diagramma. Infatti ad un elemento per cui risultano definiti degli exitCodes verrà associato un gateway dai quali si ramificano tanti elementi *sequence* corrispondenti agli *exitesCodes* definiti
- *gateway :* array di oggetti di tipo gateway ciascuno dei quali risultano completamente definiti dai seguenti attributi
    - *typeId:* identificativo univoco del gateway
    - *name:* nome associato al gateway che sarà ad esso legato quando caricato nella palette e quando l'elemento verrà disegnato nell'area di lavoro attraverso il drag&drop.
    - *iconClass:* identificativo della classe associata all'elemento che permette di definire l'icona grafica ad esso associabile.
    - *exitCodes:* array contenente la definizione testuale delle condizioni possibili di uscita associabili al gateway. Tale informazione aggiuntiva associata all'elemento permette di creare, nel momento in cui tale gateway viene istanziato nel diagramma, tanti elementi *sequence* in uscita quanti sono gli *exiteCodes* definiti.
- *event:* etichetta che permette di definire l'insieme degli elementi di tipo event ciascuno caratterizzati da:
    - *typeId:* identificativo univoco dell'elemento event
    - *name:* nome associato all'elemento event che sarà ad esso legato quando caricato nella palette e quando l'elemento verrà disegnato nell'area di lavoro attraverso il drag&drop.
- *dataObject:* permette di definire gli elementi di tipo dataObject caratterizzati rispettivamente da:
    - *typeId*: identificativo univoco corrispondente all'elemento dataObject
    - *name*: nome associato all'elemento dataObject che verrà mostrato nella palette associata all'elemento e sarà abbinato all'elemento non appena verrà effettuato il drag&drop di esso nell'area di lavoro
    - *iconClass*: identificativo della classe associata all'elemento che permette di definire l'icona grafica che è possibile associare ad esso.
- *message:*
    - *typeId:* identificativo univoco dell'elemento message

- *name:* nome associato all'elemento message che sarà ad esso legato quando caricato nella palette e quando l'elemento verrà disegnato nell'area di lavoro attraverso il drag&drop. *iconClass:* identificativo della classe associata all'elemento che permette di definire l'icona grafica ad esso associabile.
- *iconClass:* identificativo della classe associata all'elemento che permette di definire l'icona grafica ad esso associabile.
- *messageType:* valore che permette di definire se un messaggio debba essere esclusivamente di input (se impostato a valore 2) o di output (se impostato a valore 1). Ciò sarà poi tenuto in considerazione nel momento in cui viene effettuato il controllo semantico sul diagramma, in quanto sarà rilevato all'utente se sono presenti dei messaggi in input che sono elementi in uscita da qualche task o viceversa se sono presenti dei messaggi di output che sono in ingresso a qualche task.

```
{
    "genericElement":false,
    "rules":[
        {
            "idcapability":"01",
            "idcapabilityProibitionAfter":"02",
            "description":"Incompatible Task Connected"
        }

    ],
    "task":[
        {
            "typeId":"1",
            "name":"FIRST TASK",
            "iconClass":"iconTask2"
        },
        {
            "typeId":"2",
            "name":"SECOND TASK",
            "iconClass":"iconTask2"
        }
    ],
    "gateway":[
        {
            "typeId":"G2",
            "name":" MY GATEWAY",
            "iconClass":"iconTask2"
        }
    ],
    "event":[
        {
            "typeId":"E1",
            "name":" MY EVENT",
            "iconClass":"iconTask2"
        }
    ],
    "dataObject":[
        {
            "typeId":"D1",
            "name":" MY DATA OBJECT ",
            "iconClass":"iconTask2"
        }
    ],
    "message":[
        {
            "typeId":"M1",
            "name":" MY MESSAGE OUTPUT",
            "iconClass":"iconTask2",
            "url":"prova2.html",
            "messageType":1
        },
        {
            "typeId":"M2",
            "name":"MY MESSAGE INPUT",
            "iconClass":"iconTask2",
            "url":"prova.html",
            "messageType":2
        }
    ]
}
```


Figura 56 - File JSON della definizione della palette

B. Validazione semantica- sintattica del processo definito

Il tool realizzato fornisce all'utente la possibilità di effettuare un controllo sulla validità semantica e sintattica di uno specifico workflow definito. Tale funzionalità è accessibile tramite il pulsante n°5 mostrato in **Figura 50**. La verifica della correttezza sintattica si basa sull'analisi del soddisfacimento dei principali vincoli di correttezza definiti dallo standard BPMN, in particolare in ogni diagramma definito devono essere garantiti i seguenti principali requisiti:

- non è ammesso per un elemento di tipo *start event* una *sequenceFlow* in ingresso
- non è ammesso per elemento *end event* una *sequenceFlow* in uscita
- deve essere garantito per ciascun elemento del diagramma che esso presenti almeno una *sequenceFlow* ad esso collegato(in ingresso o in uscita), ovvero non son ammessi elementi scollegati dal diagramma stesso.
- È inibito il collegamento tra elementi di tipo *dataObject e message* con elementi di tipo *gateway*
- tutti gli elementi *sequenceFlow*, *messageFlow* e *objectFlow* devono essere associati ad un target e ad un source , ovvero non possono esserci link liberi nel diagramma.

I controlli di validità semantica invece si basano sostanzialmente sul soddisfacimento dei seguenti requisiti:

- verifica che siano state rispettate tutte le regole principali semantiche definite nel file di configurazione della palette, ovvero che sono inglobate nell'elemento *rules* del file .json che descrive la specifica palette che si sta prendendo in considerazione.
- verifica sulla correttezza degli exitCodes presenti definiti c per ciascun elemento nel file di configurazione della palette. Nello specifico viene garantito che siano presenti tutti gli exitCodes previsti e senza alcun duplicato.
- per ciascun elemento task di tipo *Send* e *Receive* eventualmente presente nel diagramma viene verificato che sia presente almeno un elemento di tipo *message* rispettivamente in uscita e in ingresso ad esso.

Le seguenti figure (**Figura 57-Figura 58-Figura 59**) mostrano le interfacce grafiche presentate all'utente a seguito dell'esecuzione del controllo di validità sintattica e semantica di un workflow. In particolare nella **Figura 59** si può notare come le tipologie di errori evidenziati nel grafico vengano suddivise in

- warnings che raggruppano tutte le imperfezioni semantiche rilevate, le entità che ne sono causa nel diagramma vengono evidenziate in colore blu in modo da permettere all'utente una immediata individuazione.
- errors: corrispondenti al verificarsi di errori sintattici i cui rispettivi elementi che li hanno generati sono mostrati in rosso nel workflow.

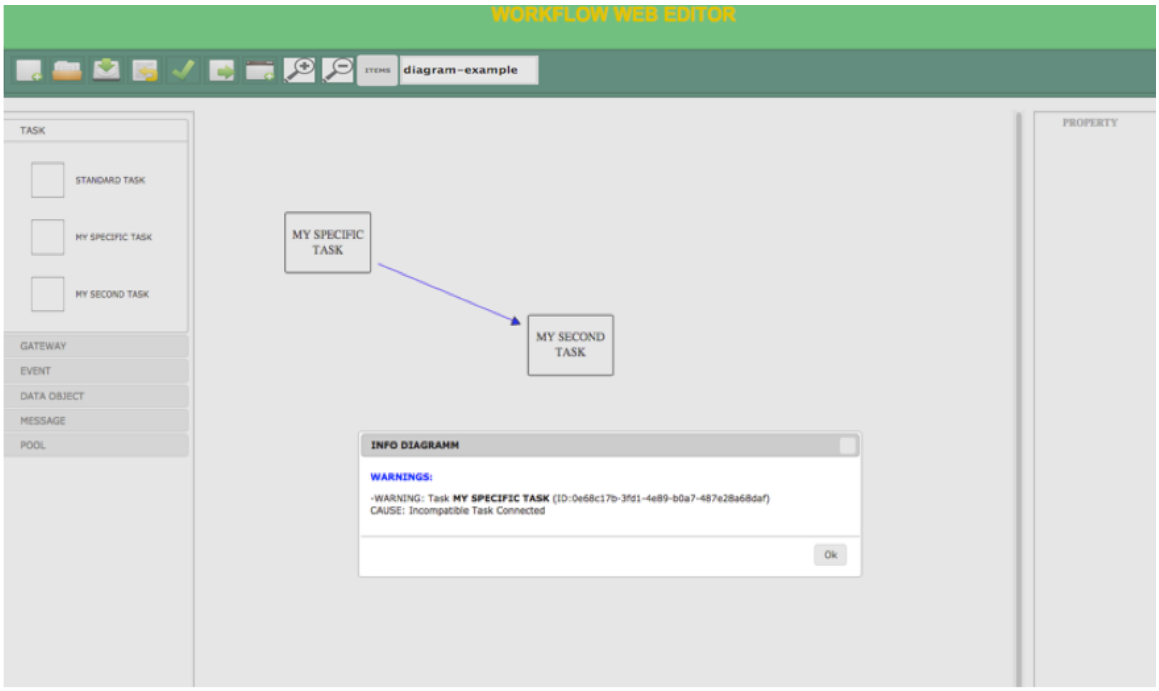


Figura 57 - Esempio di errore semantico riscontrato

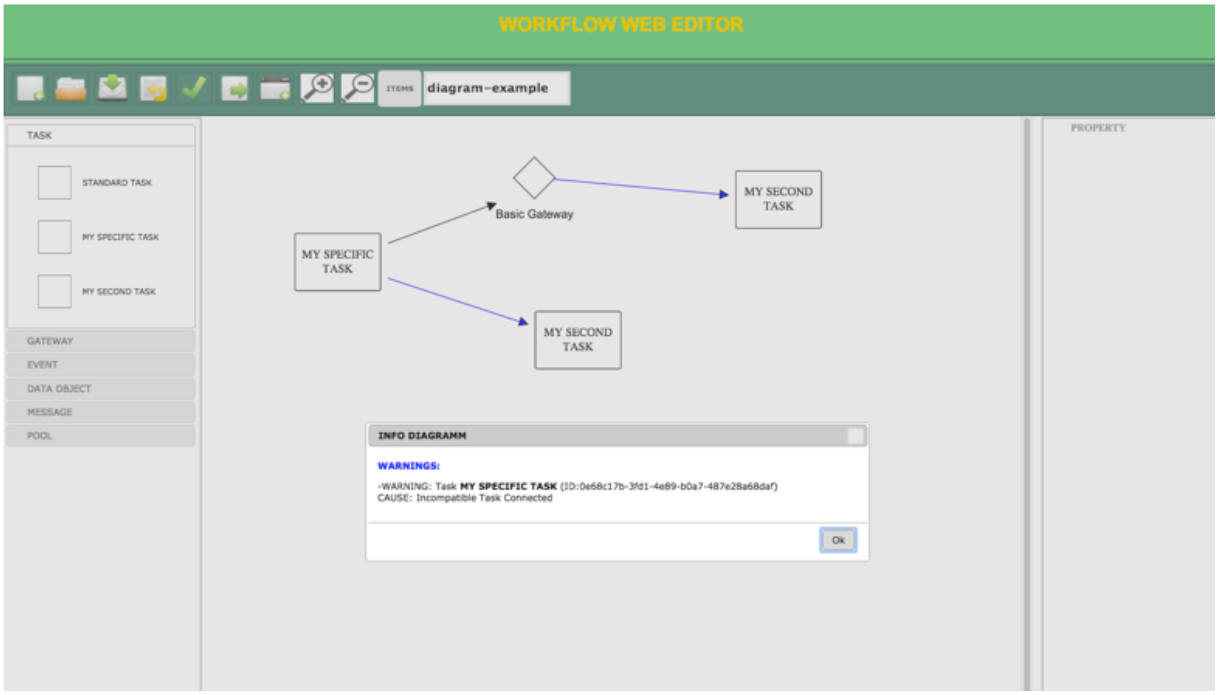


Figura 58 - Rilevamento errore di collegamento tra due task incompatibili anche non direttamente collegati

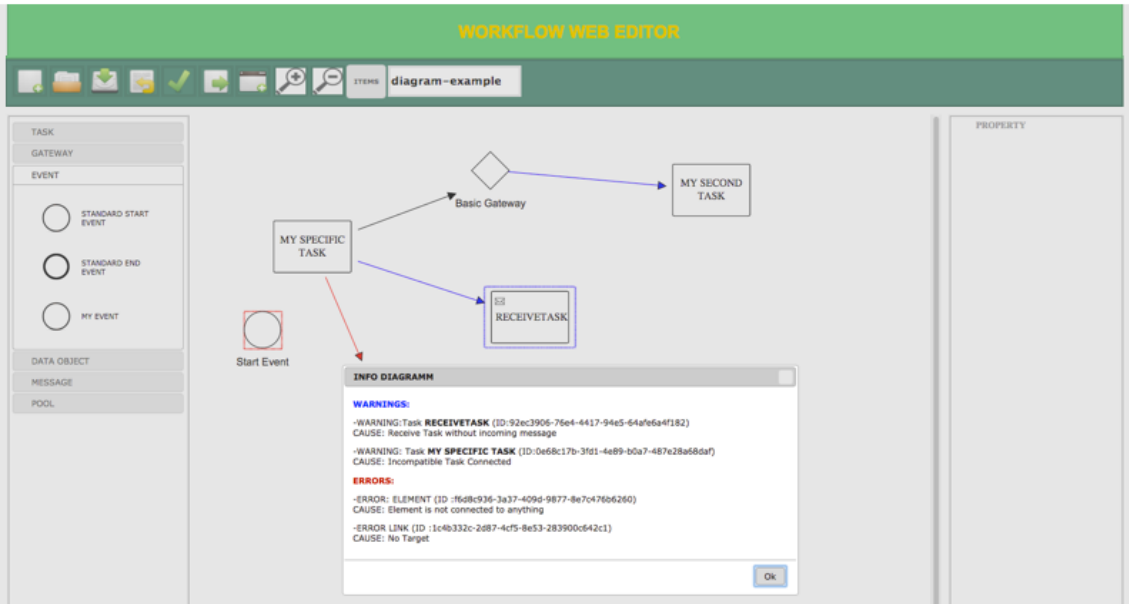


Figura 59 - rilevazione di errori semantici e sintattici presenti in un workflow

C. Esportazione del diagramma in un formato XMI standard

L'interoperabilità tra i diagrammi definiti attraverso il web editor realizzato e altri tool di definizione di diagrammi di workflow secondo lo standard BPMN è stata garantita attraverso lo sviluppo di un apposito modulo in grado di tradurre il processo creato in un formato *.bpmn* ovvero secondo lo standard XMI definito da OGM. In particolare attraverso il pulsante n°6 mostrato in **Figura 50** l'utente potrà effettuare il download del file contente la definizione xml del workflow definito nell'area di lavoro aderente allo schema XMI dello standard BPMN. Il modulo analizza tutti gli elementi presenti nel diagramma e individua quali sono i corrispondenti tag che permettono di definirli correttamente in un file *.bpmn*. Va infine sottolineata la funzionalità associata all'elemento n° 10 evidenziato in **Figura 50** attraverso il quale è possibile definire gli elementi Items che si intendono utilizzare nella definizione di un diagramma. Tale pulsante permette di attivare un interfaccia mostrata in **Figura 60** che fornisce all'utente la possibilità di definire tutti gli attributi necessari per caratterizzare completamente un elemento di tipo Items che può essere utilizzato nel workflow che si intende definire. Tali items una volta definiti potranno essere correttamente associati agli elementi di tipo message e dataObject attraverso la relativa selezione attivabile con un menu a tendina presente nella barra delle proprietà a seguito della selezione dell'elemento di interesse. Infine per ogni elemento presente nel diagramma è possibile definirne le risorse associate attraverso il relativo campo di testo presente nella barra delle proprietà. Tutte queste informazioni relative ad items e resource verranno correttamente processate nel momento in cui viene effettuato l'export del diagramma secondo lo standard BPMN.

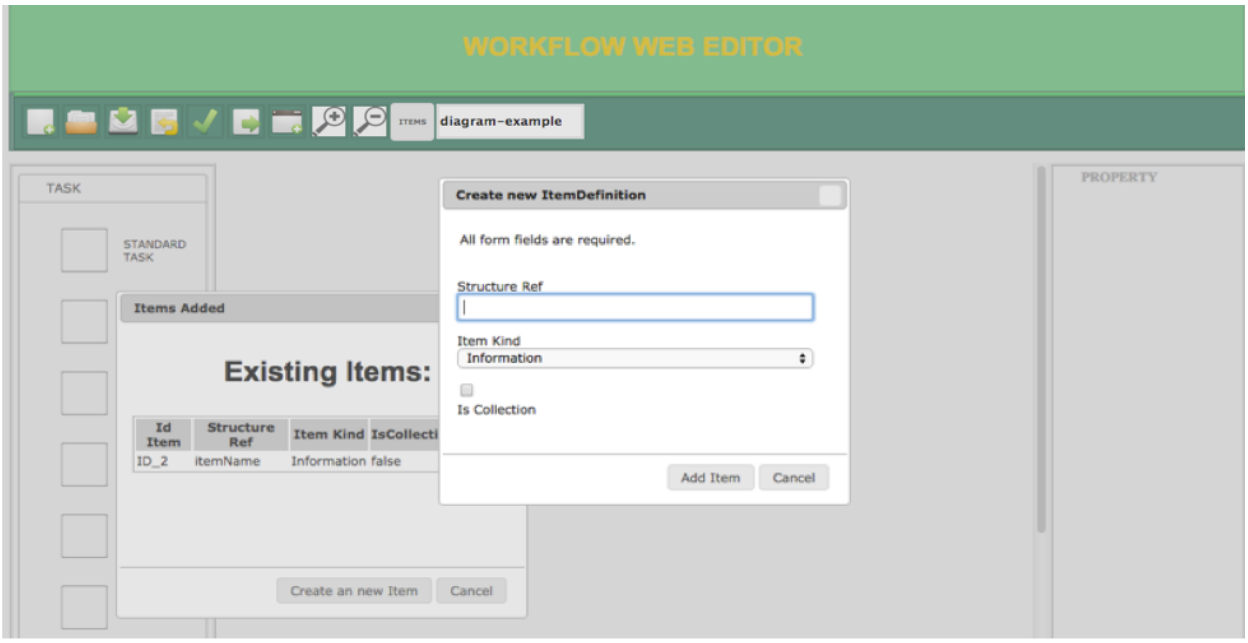


Figura 60 - Maschera di definizione degli elementi Items

D. Generazione della corrispondente definizione GoalSpec di un processo

Al fine di comprendere come l'editor di workflow sia integrato con il workflow engine che esegue un processo definito si tenga presente l'architettura mostrata nella **Figura 61** che mostra come a partire dalla definizione di un processo di business realizzato dagli analisti del dominio (utilizzando la notazione standard BPMN) si arrivi alla completa gestione del processo definito all'interno del motore di workflow considerato. Per realizzare tale passaggio si può notare come sia necessario l'utilizzo di un ulteriore modulo di conversione di un processo modellato secondo lo standard BPMN in uno specifico linguaggio dichiarativo GoalSPEC al fine di dotare il processo del necessario grado di adattività e renderlo eseguibile nel workflow engine ad agenti (MUSA). Il sistema proposto considera che esista un sistema ad agenti in esecuzione in cui ciascun membro che lo compone è a conoscenza delle sue reali capacità, quando un nuovo processo di business è pronto esso viene automaticamente tradotto in un insieme di goals secondo la specifica GoalSPEC che possono essere iniettati nel sistema. Gli agenti del sistema sono in grado di captare quando un nuovo goal viene introdotto e poter quindi verificare quale delle sue capability possono essere utili per prendere in carica uno o più goals. Esiste quindi un agente che realizza una funzione sociale attraverso la quale un goal viene assegnato ad un determinato agente. Nel momento in cui tutti gli agent goal sono assegnati può essere attivato il relativo social goal e il workflow può quindi essere eseguito.

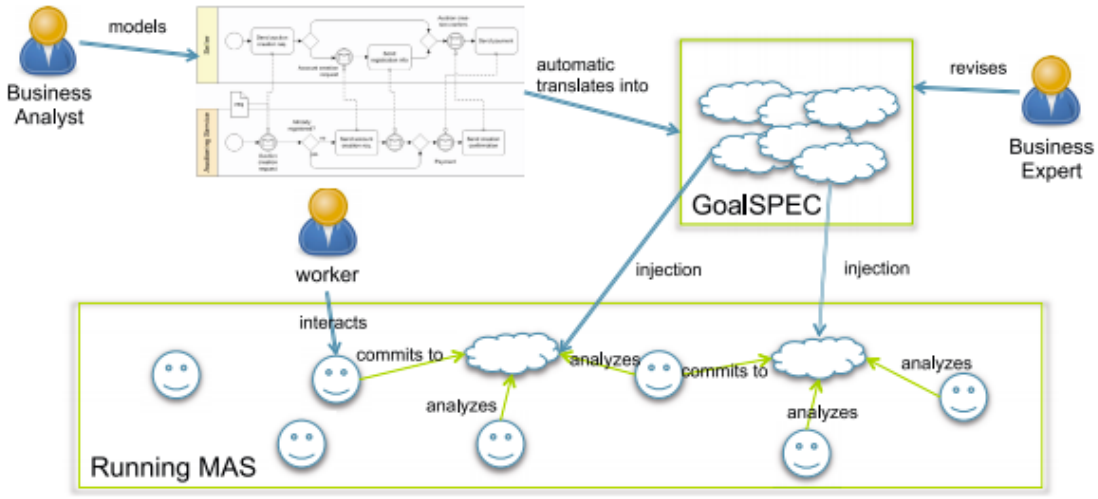


Figura 61 - Architettura del sistema di workflow management

La generazione della specifica GoalSPEC di un processo definito è stata realizzata nell'interfaccia web dell'editor tramite pulsante evidenziato nella barra degli strumenti nella **Figura 62** , la cui attivazione rende possibile ottenere la corrispondente definizione secondo il linguaggio goalSPEC relativa al diagramma che l'utente ha realizzato nell'area di lavoro.
Per chiarire meglio quale sia l'obiettivo principale di tale funzionalità verranno illustrate le principali caratteristiche del linguaggio goalSPEC e l'algoritmo che supporta la traduzione di un processo definito secondo lo standard BPMN in tale linguaggio.

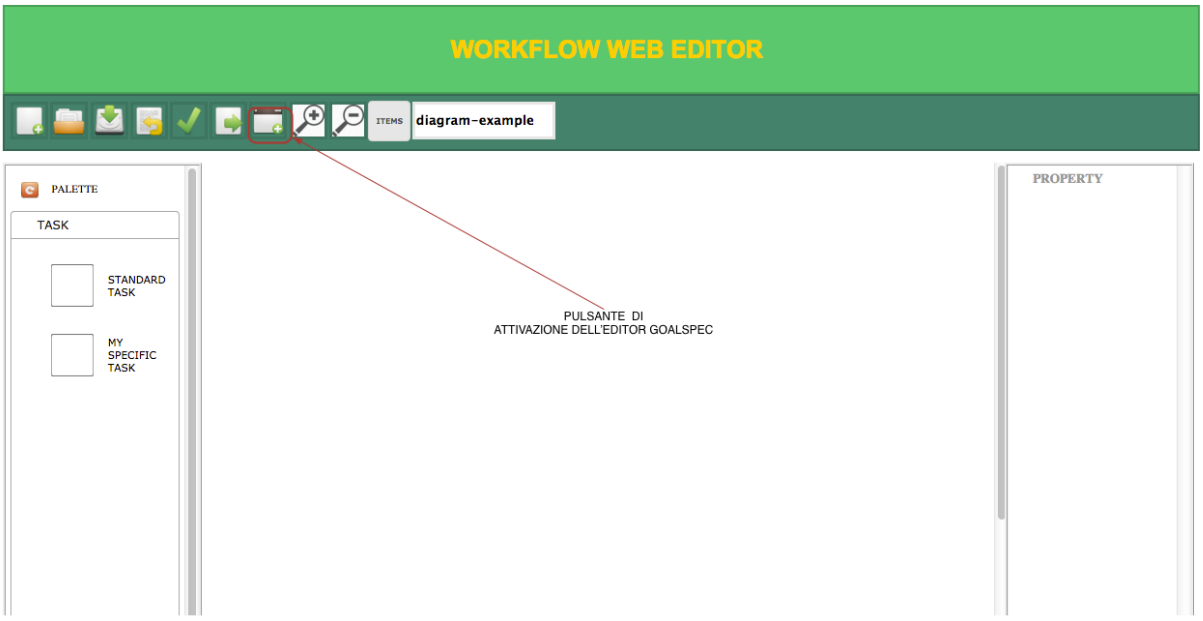


Figura 62 - Pulsante per attivazione editor goalsSPEC relativo ad un diagramma

Il linguaggio GoalSPEC

Il linguaggio dichiarativo GoalSPEC definito in [10] è stato ideato per la specifica formale dei requisisti. Esso in particolare permette di definire i goals che possono essere iniettati a run time in un sistema. Il linguaggio si basa principalmente sulla decomposizione dei goals di business, (che rappresentano cosa deve essere ottenuto) dalla loro implementazione (ovvero come raggiungere tali obiettivi). Tale decomposizione permette ad un sistema di adattare il suo comportamento a seconda delle condizioni operative correnti, quindi GoalSpec può essere considerato il substrato che rende possibile l'abilitazione delle capacità adattative ed evolutive di un sistema. Ciò è particolarmente utile se si considera che molti dei domini di business sono caratterizzati da un elevato grado di variabilità del contesto applicativo, infatti le regole di business possono cambiare ciò rende necessaria l'evoluzione delle strategie di business precedentemente adottate. Poiché lo standard BPMN non supporta un contesto dinamico, infatti è necessario ridefinire un intero workflow al fine di gestire opportunamente un eventuale nuovo requisito, si è reso necessario l'utilizzo di uno strumento di supporto quale GoalSPEC il quale rilassando i legami statici tra che cosa (*what*) e come (*how*) permette di rendere adattativo un sistema e supportarne l'evoluzione. I principali concetti su cui si basa la definizione di tale linguaggio sono

- business goals: ovvero gli interessi strategici dell'azienda che rendono necessari l'esecuzione di processi aziendali, essi vengono individuati nella fase di analisi e permettono di modellare i requisiti del sistema
- system goals: descritti come stati del mondo che il sistema cerca di raggiungere. Essi sono generalmente dei sottoinsiemi dei business goals che vengono delegati per attuare qualche tipo di automazione

In generale un goals è composto da una condizione di trigger iniziale, l'insieme degli attori coinvolti e uno stato finale del mondo che si desidera ottenere.

La specifica del linguaggio permette di gestire due categorie di system goals:

- agent goals: ovvero dei goal atomici, relativi ad uno specifico risultato nell'istanza del workflow, essi derivano dai Task presenti nella definizione di un processo e non possono essere decomposti in dei sotto-goals. Il soddisfacimento di un agent goal produce un avanzamento verso il conseguimento dell'obiettivo del workflow
- social goals: che sono dei goals che possono essere decomposti in ulteriori sotto-goal. Tali goals derivano da processi e sotto-processi presenti nel workflow. Un social goal non è necessariamente soddisfatto quando tutti i suo sotto goals sono soddisfatti. È necessario che la sua condizione finale sia verificata affinché essa possa essere considerata realizzato.

A esempio si consideri il processo mostrato nella **Figura 63**, la definizione goalSPEC corrispondente alla parte di diagramma evidenziata è il seguente. È possibile individuare in tale definizioni:

- la realtiva trigger condition *[(WHEN done(controllo_ordine) AND NOT WHEN done(emissione_fattura))*]
- le risorse umane o di sistema coinvolte nella realizzazione del goals [*THE SYSTEM SHALL ADDRESS*]
- il final state che il goal si prefigge di ottenere [*done(notifica_con_email)*]

```
GOAL order_manager.g4 :
(WHEN done(controllo_ordine) AND NOT WHEN done(emissione_fattura))
THE SYSTEM SHALL ADDRESS
done(notifica_con_email)
```

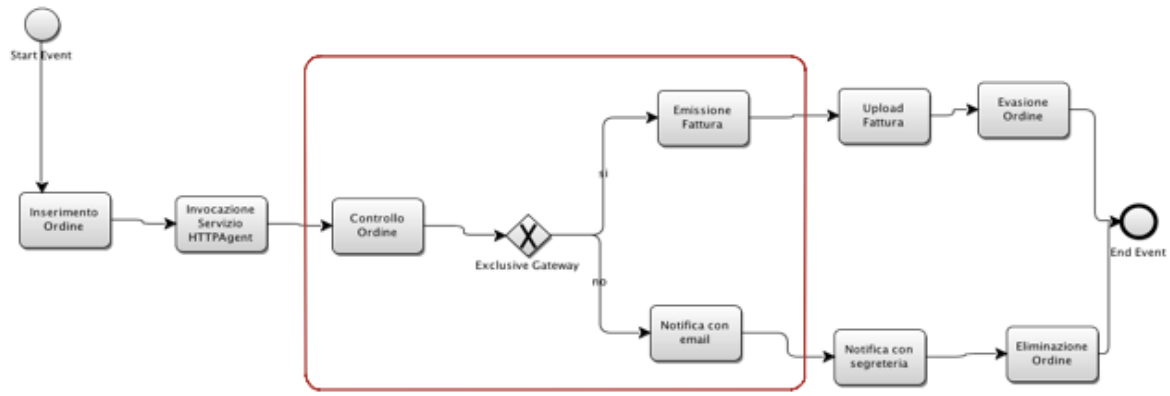


Figura 63 - Esempio di processo gestione ordine

A. L'algoritmo di generazione della specifica GoalSPEC

La generazione della specifica GoalSPEC di un diagramma BPMN viene realizzata attraverso l'utilizzo di uno specifico modulo software: BPMN2GoalsSpec. Il parsing di un workflow si avvia dalla sua definizione in xml e tiene conto della semantica adottata nella specifica BPMN. L'obiettivo principale che si pone l'algoritmo di traduzione è quello di individuare lo specifico stato del mondo corrispondente ad un generico punto del workflow. Infatti per generare i goals corrispondenti al processo considerato è necessario definire la Trigger Condition e il Final State che compongono il goal per ciascuna Activity e per un qualsiasi porzione del workflow considerata tra due punti comunque presi. È necessario sottolineare come i diversi tipi di elementi BPMN influenzino in maniera diversa lo stato nelle porzioni di workflow considerate. Infatti presupponendo una visione ontologica del mondo che risulta decomposto in stati, si può osservare che i *Gateway* nella struttura di un processo non influenzano direttamente lo stato del mondo in quanto la loro funzione è solo quella di realizzare il passaggio del flusso di controllo tra gli archi in ingresso e quelli in uscita. Ben diverso è invece il ruolo che assumono gli elementi *Event* in quanto essi sono, cosi come le Activity, responsabili di un cambiamento dello stato del mondo; infatti lo stato osservato prima dell'esecuzione di essi è generalmente diverso da quello che ne risulta in seguito alla loro attivazione. Tuttavia affinchè tali elementi si attivino è necessario che si verifichi qualche condizione, quindi si è pensato di caratterizzare tali elementi con un comportamento di tipo catching&trowing che considererà le condizioni sempre vere nel caso in cui venga applicato ad elementi di tipo gateway. Quindi per ciascun elemento presente in un diagramma si individueranno le relative WaitingCondition(WC), ovvero la condizione attraverso la quale viene effettuato il catch necessaria affinchè esso venga attivato, e la GeneratedCondition(GC) ciò quella che esso produce dopo aver effettuato il throw. Per esaminare bene quale è il comportamento di un processo tuttavia non è necessario solo il soddisfacimento delle sole condizioni di WC e di GC ma bisogna considerare anche il contesto in cui si trovano gli elementi adiacenti a quello considerato.

Pseudo – Codice dell'algoritmo

Input: il workflow BPMN
Output: un insieme di Goal in GoalSpec
Descrizione:
per ogni Activity e Throw Event *x* nel workflow

sia *waiting(x)* il suo waiting event,
generated(x) il suo generated state
crea un nuovo goal
crea condizioni congiuntive *triggCondition*, *finState*
aggiungi *waiting(x)* a *trigCondition*
aggiungi *generated(x)* a *finState*
crea condizioni disgiuntive *successorCondition*, *predecessorCondition*
per ogni incoming Sequence Flow *i* di *x*
sia *cond(i)* il predicato di *i* (se ne possiede),
backward(i,source(i),EVENT) l'evento backward del nodo sorgente
aggiungi a *predecessorCondition cond(i)* AND *backward(i,source(i) ,EVENT)*
per ogni outgoing Sequence Flow *j* di *x*
sia *cond(j)* il predicato di *j* (se ne possiede),
forward(target(j),STATE) lo stato forward del nodo destinazione
aggiungi a *successorCondition cond(j)* AND *backward(source(j) ,STATE)*
aggiungi *predecessorCondition* a *trigCondition*
aggiungi *successorCondition* a *finState*
inserisci *trigCondition* e *finState* nel goal
per ogni Process p del workflow
crea un nuovo social-goal
crea condizioni disgiuntive *triggCondition*, *finState*
per ogni nodo BPMN privo di incoming Sequence Flow s
sia *forward(s,EVENT)* l'evento forward del nodo
aggiungi *forward(s,EVENT)* a *trigCondition*
per ogni nodo BPMN privo di outgoing Sequence Flow e
sia *backward(e,STATE)* lo stato backward del nodo
aggiungi *backward(e,STATE)* a *finState*
inserisci *trigCondition* e *finState* nel social-goal

B. Realizzazione della funzionalità web based BPMN to GoalSpec

Al fine di poter integrare l'algoritmo di traduzione di un diagramma BPMN con la sua definizione secondo le specifiche del linguaggio GoalSPEC si è reso necessario realizzare un servizio web che preso in input l'url in cui è possibile reperire il file .bpmn, contente la definizione del processo di business, dopo aver effettuato le opportune elaborazioni ritorna come output la definizione GoalSPEC corrispondente al processo ricevuto. In particolare attraverso l'attivazione del pulsante corrispondente a tale funzionalità nella barra degli strumenti viene invocata una funzione javascript che effettua dapprima il salvataggio sul server del diagramma in formato .bpmn e permette poi di aprire una nuova finestra del browser che rappresenta un editor web based della corrispondente definizione GoalSPEC. In particolare la funzione invocata (*openGoalsSpecView*), mostrata nel seguito, dopo aver ottenuto la definizione bpmn standard del diagramma invocando la funzione javascript *serializeDiagramm*, ingloba tale informazione in pacchetto dati per effettuare una chiamata ajax al servizio *SaveBpmnDiagramm* che ricevuto l'intero diagramma ne effettua il salvataggio su uno specifico percorso del server. In particolare non appena il servizio invocato termina correttamente la sua esecuzione un nuovo file al quale è associato il nome bpmnDiagramm*.xml* è reso reperibile e pubblicamente accessibile all'indirizzo *http://ipHost:8080/WebEditor/file/ bpmnDiagramm.xml*

```
function openGoalsSpecView(){
        var  bpmnStandardDiagramm=serializeDiagramm();

        $.ajax({
           url : "http://ipHost:8080/WebEditor/SaveBpmnDiagramm",
           type: "POST",
           data:{"bpmnDiagramm": bpmnStandardDiagramm.flush()},
           dataType: 'text',
           success : function (data) {

               }

       });
       window.open("http://ipHost:8080/WebEditor/goalsSpecFromBPMN.jsp");
}
```

La nuova pagina web mostrata all'utente rappresentata nella **Figura 64** mostra la corrispondente definizione GoalSPEC del processo definito nell'ambiente di lavoro del Web Editor. La descrizione GoalSPEC del processo è ottenuta attraverso l'invocazione di un servizio web appositamente realizzato invocabile al seguente indirizzo
http://ipHost:8080/BPMN2REQWEB/Bpmn2GoalSpecService
Che necessita del parametro *urlFile* con il quale viene indicata la localizzazione del file che contiene la specifica xmi secondo lo standard BPMN di uno specifico diagramma. La definizione goalsSPEc viene mostrata in un campo di testo editabile al fine di permettere all'utente di effettuare le opportune modifiche eventualmente necessaria affinché i goals generati possano essere inviati ad un motore di workflow per essere elaborati. Nell'editor è stato inoltre introdotto un ulteriore pulsante che permette di salvare in formato testuale i goals generati ed eventualmente modificati affinché l'utente abbia la possibilità di tenerne traccia per poter riutilizzare nel caso in cui sia necessario.

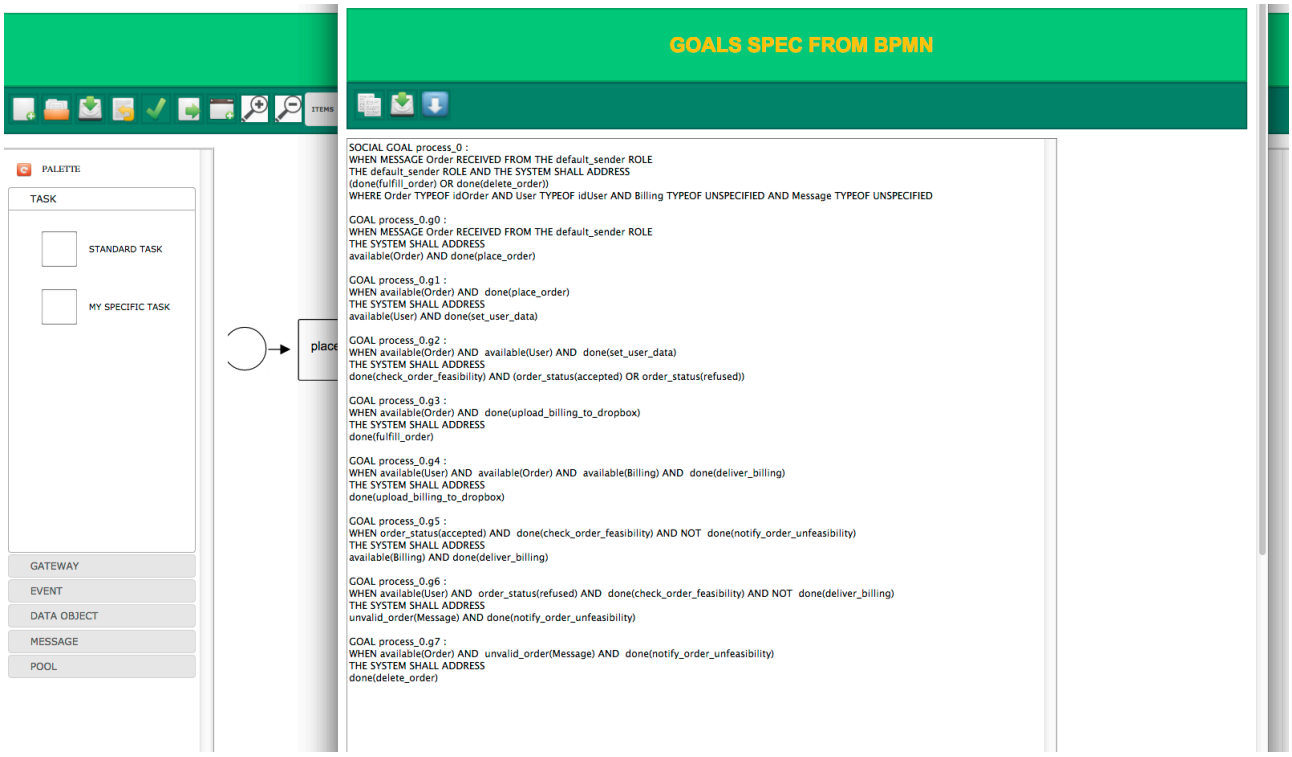


Figura 64 - Editor GoalSPEC

RIFERIMENTI

1. Diagramo Editor- http://diagramo.com/
2. LucidChart Editor- https://www.lucidchart.com
3. DrawIo Editor- https://www.draw.io/
4. Gliffy Editor- https://www.gliffy.com/
5. Cacoo Editor- https://cacoo.com/lang/en/
6. BPMN IO Editor- http://bpmn.io/
7. Signavio editor- http://www.signavio.com/
8. Oryx editor - https://code.google.com/p/oryx-editor/
9. JALAVA: http://sourceforge.net/projects/jalava/
10. Sabatucci, L., Ribino, P., Lodato, C., Lopes, S., & Cossentino, M. (2013). GoalSPEC: A Goal Specification Language Supporting Adaptivity and Evolution. In Engineering Multi-Agent Systems (pp. 235-254). Springer Berlin Heidelberg.
11. BPMN : Business process model and Notation. http://www.bpmn.org/
12. JQUERY UI - https://jqueryui.com/
13. JOINTJS - http://www.jointjs.com/