

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione



POLO TERRITORIALE DI COMO
Master of Science in Computer Engineering

Personal Process Management

Supervisor: Prof. Marco Brambilla

Master Graduation Thesis by:

Pierfilippo Bianchi Student Id. number 770115
Davide Maria Filippo Ripamonti Student Id. number 770028

Academic Year 2012 - 2013

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione



POLO TERRITORIALE DI COMO
Corso di Laurea Specialistica in Ingegneria Informatica

Gestione di Processi Personali

Relatore: Prof. Marco Brambilla

Tesi di laurea di:

Pierfilippo Bianchi	Matr. 770115
Davide Maria Filippo Ripamonti	Matr. 770028

Anno Accademico 2012 - 2013

Sommario

In questa tesi affrontiamo il problema dell'integrazione dei metodi propri del BPM nella gestione dei processi personali, sfruttando le funzionalità messe a disposizione dalle reti sociali. Al giorno d'oggi, la maggior parte degli utenti connessi ad Internet è registrata ad almeno un social network, tramite il quale interagisce e comunica con amici e colleghi. Una tendenza recente è quella della condivisione della gestione e della realizzazione di liste di azioni. Un semplice esempio è dato dalle molteplici applicazioni web per la gestione delle cosiddette "to-do list". Esse sono molto semplici e intuitive da usare, e, in alcuni casi, offrono anche funzionalità social. Tuttavia hanno anche alcuni limiti rappresentati principalmente dal fatto che non consentono di gestire le attività in maniera strutturata: sono semplici sequenze di azioni. Al contrario, la notazione BPM, usata in ambito business, permette di esprimere dipendenze e condizioni sull'esecuzione delle azioni ma è inadatta a questo contesto. Per prima cosa analizziamo le applicazioni simili esistenti sul mercato, allo scopo di comprendere sia i loro punti di forza che i loro aspetti critici. Successivamente studiamo la Business Process Management Notation con l'intenzione di identificare un set di costrutti adatto alla gestione di processi personali. Per ciascuno di essi proponiamo la miglior soluzione grafica in termini di espressività e aderenza ai principi di Moody, dopodiché formuliamo quattro sintassi diverse, ognuna formata da una combinazione degli elementi, per farle testare e valutare dagli utenti. Presentiamo poi la progettazione e l'implementazione di un prototipo di applicazione fornita di editor grafico per la modellazione di scenari di vita quotidiana. Infine discutiamo i dati raccolti durante gli esperimenti di test e traiamo delle considerazioni sulle quattro sintassi.

Abstract

In this thesis we face the problem of the integration of BPM methods into the management of personal processes, exploiting social network features. Nowadays, most of the users connected with the Internet have at least an account on a social network, which they use to interact and to communicate with friends and colleagues. A recent trend in the online social sharing is represented by the socialization of task management, which consists in performing lists of actions in a collaborative way. A simple example is given by all the existing web applications of to-do list management. All these web applications are very simple and user friendly, and in some cases they integrate social features. However they have also some limits, because they do not allow to manage the activity sequence in a structured way, like BPM does: they are simple sequences of actions. We start analyzing similar existent applications in order to understand their points of strength and their critical aspects. Then we study the Business Process Management Notation with the purpose to identify a set of constructs suitable to describe personal tasks. For each one we have chosen the best graphical solution which is visually expressive and compliant with the Moody's principles. Then we create four syntaxes, each one made up of a combination of the original set of elements and let them be tested by end users. We then present the design and the development of an application prototype with a graphical editor, which allows modeling daily life scenarios. At the end we discuss data collected during the experiments in order to evaluate the four syntaxes.

Contents

Sommario	v
Abstract	vii
1 Introduction	1
2 Related Works	5
2.1 Web applications	5
2.1.1 To-do list applications	5
2.1.2 Workflow management	14
2.1.3 Application conclusions	18
2.2 Academic works	18
3 Background	21
3.1 Business Process Management	21
3.2 Business Process Modeling	21
3.3 Business Process Modeling Notation	23
3.4 Social BPM	25
3.5 The Physics of Notation	26
3.5.1 Semiotic Clarity	27
3.5.2 Perceptual Discriminability	27
3.5.3 Semantic Transparency	28
3.5.4 Complexity Management	29
3.5.5 Cognitive Integration	29
3.5.6 Visual Expressiveness	30
3.5.7 Dual Coding	31
3.5.8 Graphic Economy	31
3.5.9 Cognitive Fit	32
3.5.10 Interactions among principles	33

3.6	OAuth	34
3.7	Draw2D	36
4	Design	39
4.1	Description	39
4.2	Requirements	40
4.2.1	Use case	41
4.3	Modeling syntax	43
4.3.1	BPMN Analysis	43
4.3.2	PPM Notation	44
4.3.3	Restrictions	49
4.3.4	Moody's principles	51
4.3.5	Building the four syntaxes	55
4.3.6	Application Behavior	56
4.4	Engine	58
5	Implementation	61
5.1	The Web Site	61
5.1.1	Profile	63
5.1.2	Contacts	63
5.1.3	Processes	64
5.1.4	Social APIs	65
5.2	Graphical Editor	68
5.2.1	Classes	68
5.2.2	Ports and connections	69
5.2.3	Gateways positioning	71
5.2.4	Elements management	72
5.2.5	Control Algorithms	74
5.3	The Logging System	78
6	Experiments	81
6.1	Preparation of the experiments	81
6.2	Experiment procedure	85
6.3	Data analysis	87
6.3.1	Outliers	87
6.3.2	Durations	87
6.3.3	Number of elements' creations and deletions	88

6.3.4	Validation requests	89
6.3.5	Validation errors	90
6.3.6	Wrong connections	92
6.3.7	Editing elements	94
6.3.8	Questionnaires	95
6.4	Conclusions	100
6.4.1	Syntax one	100
6.4.2	Syntax two	101
6.4.3	Syntax three	102
6.4.4	Syntax four	102
6.4.5	Elements evaluation	103
7	Conclusions	105
7.1	Experience and discussion	105
7.2	Future work	106
	Bibliografia	107

List of Figures

3.1	Visual variables	27
3.2	Semantic transparency of relationships between elements . . .	29
3.3	Visual Expressiveness	31
3.4	Interactions among principles	33
3.5	OAuth authorization flow	37
4.1	System architecture	40
4.2	Use case	42
4.3	Task element	44
4.4	Task icons	45
4.5	Sequence connection	45
4.6	Parallel opening and closure	46
4.7	Conditional opening and closure	46
4.8	Global parameters	47
4.9	One-local parameter	47
4.10	Loop element	48
4.11	Intermediate events: Wait For and Wait Till	48
4.12	Start and End events	49
4.13	Blocks levels	49
4.14	The first diagram is syntactically correct while the second presents a blocks structure violation: the loop backward con- nection points to a task with a different level	50
4.15	Wrong cascading gateways openings and correct cascading gateways closures	54
4.16	Process states	58
4.17	Task states	59
5.1	ER diagram	62

5.2	Profile page	63
5.3	Contacts page	64
5.4	Social network's friends import panel	65
5.5	Friend's details and contacts merging	66
5.6	Process page	67
5.7	Graphical editor interface	69
5.8	Editor class diagram	70
5.9	Connection creation	71
5.10	Parallel gateway	71
5.11	Context menu for the creation of task and events	72
5.12	Context menu for the creation of gateways	72
5.13	Task editing panel	73
5.14	Conditional branch editing panel	74
6.1	Graeco-Latin square of use case couples	84
6.2	Valid experiments	85
6.3	Valid experiments	85
6.4	Durations per syntax in seconds	88
6.5	Experiments' duration per single user	89
6.6	Average number of creations and deletions per process	90
6.7	Average number of elements per process	92
6.8	Average number of validation requests per single test	92
6.9	Percentage of validations per syntax	94
6.10	Validation errors by syntax per single test	94
6.11	Validation errors per single test	95
6.12	Errors'percentages per syntax	96
6.13	Total connection errors per syntax	97
6.14	Connection errors per single test	98
6.15	Average editing time per process by syntax and element	99
6.16	Number of info panels opened per single test	99
6.17	Syntaxes difficulty	100

List of Tables

2.1	To-do list and workflow management analyzed applications . . .	6
2.2	To-do list applications' features	15
2.3	Workflow applications' features	17
4.1	Elements analysis	52
4.2	Events analysis	53
4.3	The four syntaxes	56
5.1	Logged actions	80
6.1	Use cases	83
6.2	Experiments sets	86
6.3	Validation errors codes	91
6.4	Wrong connections codes	93

Chapter 1

Introduction

With the advent of Web 2.0 and the explosion of the social networks phenomenon, we have assisted to an evolution that was also spreading across the business world. In fact, social networks' peculiarities are widely exploited in the realization of business applications to achieve a better collaboration and performance improvement.

Nowadays, most of the users connected with the Internet have at least an account on a social network, which they use to interact and to communicate with friends and colleagues. Social networks themselves make available the functionality to allow their integration in existing systems and to exploit the resources they offer.

Web 2.0 technologies can thus be an interesting and powerful instrument for organizations. Their interactivity can bring more employees into daily contact at a lower cost. If properly used, they can also encourage participation in projects and idea sharing. Last but not least, they may also strengthen bonds with customers and improve communications with outside partners. In a company lot of resources are generally invested in internal management. Therefore, an evolution in this way would represent a great benefit in terms of costs.

A recent trend in the online social sharing is represented by the socialization of task management, which consists in performing lists of actions in a collaborative way. A simple example is given by all the existing web applications of to-do list management like *RememberTheMilk*, *Astrid*, *To-doledo* and many others. All these web applications are very simple and user friendly, and in some cases they integrate social features. However they have also some limits, because they do not allow to manage the activity sequence

in a structured way, or to impose dependencies and constraints among them: they are simple sequences of actions.

In business environment there have long been solutions for the simplification and optimization of the processes management, which allow increasing the performances at different company levels. The most used solution is the *BPM* (Business Process Management) which has a well-structured notation with a great expressive power: the *BPMN*. On the other hand, this one and the other types of notations are not suitable to describe simple processes like those of daily life. In fact they have elements too specific for the working environment and they use symbols that represent concept too difficult to be understood by common people who are not familiar with modeling in general. Moreover, an attempt to bring social integration has been developed, the *Social BPM*, but it remains confined in academic and industrial setting and has not been exploited for end users.

The aim of our thesis is to find the connecting link between these two worlds, the business and the daily one, building an application for the management of personal processes based on a structured notation like the *BPMN*, but at the same time easy and intuitive for common people. From these premises, it develops the idea to create a simple but effective notation for the description of personal processes, the *PPM* (Personal Process Management). The concept is to take inspiration from the *BPMN* syntax, identifying those elements and notions that might be useful and discarding those too specific for the business or less significant for the description of daily life processes.

An important aspect is that of socialization: in fact generally the social networks integration greatly increase the interaction and allow exploiting the data and potentialities offered by social networks themselves. In our case it would allow to involve other persons and to assign tasks to them, so that the process goal will be reached in a participatory way. The integration permits a quick access to the user contacts, so that he can import and save them and afterwards assign them to tasks. It is also fundamental to send notifications regarding task and process states.

The key point is to achieve an intuitive notation such that all kind of users, also those without any confidence with diagrams, would be able to use it in the modeling of their processes. This target can be realized through an

accurate choice of the notation trying to find the best compromise between its descriptive power and its intuitivity and usability. To achieve our goal we have developed a prototype of a graphical editor with the purpose of let it be tested by people and to collect significant data and to make judgments based on evidence.

This thesis is structured as follows:

- Chapter 2 - Related Works. In this chapter we will list and analyze the similar web application already existent for the management of personal processes. We will compare them and highlight both the positive and negative aspects of each one. Also some academic works concerning PPM will be presented.
- Chapter 3 - Background. In this chapter we will present the background theory on which our work is founded and the notions we have studied in order to develop our application. In particular we will first analyze the BPM and the features of its notation. Then we will illustrate the guidelines to keep in mind to build consistent and valid graphic notations, and finally some interesting technology we have used to implement our application.
- Chapter 4 - Design. In this chapter we will examine in depth the design of each aspect of our application. We will explain the methods we have followed to come up with the final visual notation, together with all the problems we have faced and the solutions we have adopted to resolve them.
- Chapter 5 - Implementation. In this chapter we will explain how we have developed the web application and all the technical aspects. Some screenshots will be presented in order to better explain the general functioning of the system.
- Chapter 6 - Experiments. In this chapter we will explain the exact procedure we have followed to let the users test our application and the proposed syntaxes. We then will show and analyze the data collected through the experiments.

- Chapter 7 - Conclusions. In this chapter we will check if the goals we set have been achieved. We will give some final consideration and we will identify the aspects to improve and some other interesting ideas to develop in future.

Chapter 2

Related Works

In this chapter we will analyze the tools available on the Web and the academic researches made in the context of the personal task management. Essentially, existing tools can be divided into two sets: web application for the management of to-do lists and web applications for the management of workflows. Although quite similar, none of these tools seems to have the requirements and the behavior we are proposing in our thesis. Also in the academic world, there are only few researches made in this direction.

2.1 Web applications

There are a lot of web and mobile applications to manage to-do lists and workflows. The majority of them are focused on planning daily life actions. They are simple and immediate to use, some others instead are more oriented to support workflow management and are focused to be used in a company environment. Table 2.1 shows the web applications we have gathered and found interesting.

2.1.1 To-do list applications

All the considered applications for the management of to-do lists have some common features and some original ones. Here are listed all the analyzed features with their explanations:

- **social network integration:** it simplifies the application login without the need to register a new account, allows a quick access to the user contacts and could be used to publish contents and send notifications directly on social networks

Application	URL
Remember The Milk	http://www.rememberthemilk.com/
Online Task List	http://www.onlinetasklist.com/
HiTask	http://hitask.com/
Todoist	http://todoist.com/
Toodledo	http://www.toodledo.com/
Voo2do	http://voo2do.com/
Astrid	http://astrid.com/
Cozi	http://www.cozi.com/
Bla-bla List	http://blablalist.com/
ccToDo	http://www.cctodo.com/
The Online CEO	http://roughunderbelly.com/user/login
Gtdagenda	http://www.gtdagenda.com/
Do	https://do.com/
Producteev	https://www.producteev.com/
Flow	http://www.getflow.com/
KiSSFLOW	http://kissflow.com/
Comindware	http://www.comindware.com/

Table 2.1: To-do list and workflow management analyzed applications

- **task via email:** it allows to add new tasks to a list, simply sending them to the application using email protocol
- **quick insert:** tasks are inserted in a quick and simple way without unnecessary steps
- **categories:** they are used to index tasks and organize them in a logical way simplifying their research
- **geolocation:** it allows to indicate on a geographic map the exact physical location where the task will be performed
- **contacts management:** it allows to organize and manage own contacts list simplifying the task assignments
- **API support:** the application offers powerful methods to facilitate system integration and data access
- **import:** it supports the import of lists into the application
- **export:** it supports the export of lists in other formats
- **feed:** it supports the information sharing through feed RSS
- **search:** it is possible to search through lists and tasks using categories and tags
- **notifications:** they are messages that notify users about deadlines, tasks completions and other general information
- **synchronization:** it is a mechanism that synchronizes lists created within the application with external and mobile devices, so that it is possible to have all the daily activities under control
- **keyboard shortcuts:** they increase new tasks creation and are more suitable to insert information
- **project management:** it helps the management of multiple lists and offers business collaborative tools such as a chat to exchange ideas and discuss the tasks
- **permissions:** the administrator can set permissions to control the creation, deletion, editing of tasks, and the reading and possibility to comment by other users

- **task assignment:** it allows to assign a task to one or more people
- **public sharing:** it allows the publication of own lists in order to share them and make them accessible to others people
- **reports / statistics:** they are graphs that summarize statistics based on data collected by the application, like task durations or the most productive users
- **drag & drop:** it increases the interaction and the ease of use of the application, for example dragging a contact over a task to assign it to him
- **file upload:** files can be uploaded and associated to a task or to a group of tasks.
- **time tracking:** it allows to constantly control the deadline of each task
- **multiple entry:** it allows to enter multiple tasks in a lists
- **backup / restore:** it allows to save lists and tasks progresses and to restore early states in case of errors
- **periodic checklists:** they provide the possibility to create lists that can be repeated over time (daily, weekly, monthly)
- **booklet print:** it allows to easily print a to-do list as a foldable booklet that you can put in your pocket
- **voice recognition:** they provide the possibility to insert vocal tasks.
- **mobile support:** it allows list interaction on mobile device trough dedicated apps or with adaptive interfaces
- **priorities / dependencies:** it allows to define priorities among tasks or to assign a priority tag to them.

Table 2.2 relates the web applications with the listed features.

Let's now analyze each solution describing their principal characteristics and listing their pros and cons.

Remember The Milk

This is one of the most full-featured to-do list managers that gives you many options, including the ability to be reminded of your task via many methods, such as emails and SMS. You can assign dates, categories and time estimates to tasks. Remember The Milk supports geolocation, so it is possible to locate on map the exact place of the tasks executions. It allows you to organize your tasks into tabs and tags to do your work smooth and fast. There is the possibility to repeat intervals and also collaborative features are available. It has both a web and a mobile version always in sync.

Pros	Cons
<ul style="list-style-type: none"> - Geolocation - API support - Feed - Keyboard shortcuts - Smart tags - Task submission via email 	<ul style="list-style-type: none"> - Small number of export formats

Online Task List

It has an easy to use and fast user interface for the managing of task. It supports task assignments to team members, notes, priorities, due dates and notifications via email or SMS. It is also possible to set recurring tasks, import and export lists from spreadsheet and search tasks by description, detail, category, priority, date and status.

Pros	Cons
<ul style="list-style-type: none"> - Keyboard shortcuts - Permissions - File upload - Task priorities - User roles 	<ul style="list-style-type: none"> - No synchronization - No mobile support - Not user-friendly

HiTask

This online task manager helps end users and businesses to effectively organize their projects. It has a very clean, well organize and simple to use interface, which allows scheduling, assigning and creating tasks and projects

in a short time. It has a reminder system that will send you reminders in your mailbox. HiTask is suited for individuals and teams and has a mobile application.

Pros	Cons
<ul style="list-style-type: none"> - API support - Feed - Permissions - File upload - Multiple entry - Backup / restore - Task hierarchy 	<ul style="list-style-type: none"> - No social network integration

Todoist

This task manager application is designed for creating hierarchies of projects and tasks. It is simple, fast and user-friendly and allows you to organize your lists as well as create calendars, sub-projects and sub-tasks. Its features comprehend grouping asks, email reminders and color coding and keyboard shortcuts. It has a mobile web version and is fully integrated into Gmail and other online productivity tools.

Pros	Cons
<ul style="list-style-type: none"> - Keyboard shortcuts - Intuitive 	<ul style="list-style-type: none"> - Absence of team support

Toodledo

This is an advanced and very complete to-do list organizer which handles hierarchies, folders, tags, notes, priorities, goals, time tracking, timers, and contexts. It supports also repeating tasks, search, history, sharing and collaboration, reminders via email and SMS. It is also available for mobile phones.

Pros	Cons
<ul style="list-style-type: none"> - Geolocation - File upload - Multiple entry - Backup / restore - Task priorities - Goal notion - Task priorities 	<ul style="list-style-type: none"> - A lot of premium features

Voo2do

This manager has an easy user interface and allows to work on different projects simultaneously, collaborate with others, publish task lists, prioritize the tasks, set the deadlines, and track the estimated and actual time.

Pros	Cons
<ul style="list-style-type: none"> - Multiple entry 	<ul style="list-style-type: none"> - No synchronization - No mobile support - No notifications - Poor interface

Astrid

This application allows to create and share lists, also with unregistered users. Astrid is set up to email the assigned person about the task, and in that email, the recipient can either say yes or no to helping out. A notification will be added to your account once they respond. You can also assign due dates and priority levels, add descriptions and comments. It is available for mobile phones, it has browsers' extensions and is always synchronized across devices. An interesting feature is the possibility to enter to-do lists by voice.

Pros	Cons
<ul style="list-style-type: none"> - Keyboard shortcuts - Voice recognition - Social oriented - Good integration with Twitter and Facebook 	<ul style="list-style-type: none"> - Notification system - File upload is not possible

Cozi

This application is designed for the family and helps you manage schedules, organize your grocery shopping lists and to-do lists and capture favorite memories in a family journal. It is accessible from any computer or mobile phone.

Pros	Cons
- Intuitive	- No categories - Designed for the family context only

Bla-bla List

This to-do list organizer allows to share lists with others even if they don't have an account. You can publish your lists with RSS so that others get instant updates. You can also share your lists privately and work on them together.

Pros	Cons
- Feed - Permissions - Simplicity and minimalism - Private and public sharing of lists	- No categories - No synchronization - No mobile support - It is not possible to share a single task

ccToDo

This application is simple and intuitive and provides cloud synchronization. It supports task submission via email.

Pros	Cons
- Task submission via email	- Minimum functionality

The Online CEO

The most interesting feature of this to-do list manager is the points system to prioritize tasks. It follows the principle that every kind of to do list falls in a category which can be assigned certain number of points for each task completed under that category. Once you have list down a to-do and

complete it, you get certain points. At the end of the week you can calculate your productivity by referring the points graph. This system is really innovative and if you work regularly it can give you a good motivation.

Pros	Cons
- Task priorities	- No categories - No synchronization - No mobile support

Gtdagenda

This to-do list manager supports lists for goals, projects, tasks, next actions, checklists, calendar and schedules. Schedules are templates for recurring daily or weekly time allocations. It has also a mobile version.

Pros	Cons
- Periodic checklists - Task priorities - Social networks integration	- -

Do

This application permits to easily organize track, and create tasks in projects. Tasks lists can be shared or submitted via email. Its features comprehend: notes, reminders, groups, conversations, templates, feeds, calendar synchronization. It has mobile versions.

Pros	Cons
- Feed - Permissions - File upload - Project templates - Social integration with Google and Linkedin	- -

Producteev

It has a clean and clear interface and supports a prioritization system. You can assign tasks to other people sending invitation via email or perform

actions in a collaborative way.

Pros	Cons
- Social networks integration	- -

Flow

This application provides all the collaboration tools needed to manage projects online in one centralized place for everyone. Team members can be added and tasks can be delegated to them. It supports files upload, deadlines, discussions, tasks via mail and tagging. Rather than a static list of to-do items, each task can be commented on by team members. It is always synchronized with all the devices.

Pros	Cons
- social network integration	- -
- notifications	
- time tracking	

2.1.2 Workflow management

This kind of applications focuses on workflow management support oriented to business environment. Here follows the list of some significant features we have identified for this kind of tools, that come in addition to the features previously shown for to-do lists applications:

- **data submission:** it allows a data exchange between actors and application
- **workflow control:** it is possible to control the flow execution imposing conditions
- **workflow automation:** the execution of the activities is supported by the application that coordinates the flow
- **graphical workflow builder:** it allows to describe workflows in the application by means of graphical elements

We have found only two applications on the Web that belong to this category: KiSSFLOW and Comindware. Table 2.3 shows these applications and their features.

features	Remember The Milk	Online Task List	HiTask	Todoist	Toodledo	Voo2do	Astrid	Cozi	Bla-bla List	ccToDo	The Online CEO	Gtdagenda	Do	Producteev	Flow
social network integration	*				*		*					*	*	*	*
task via email	*		*			*				*		*	*	*	
quick insert	*		*		*		*	*	*		*		*	*	
categories	*	*	*	*	*	*	*			*		*	*	*	*
geolocation	*				*										
contacts management	*	*	*			*	*	*				*	*		
API support	*		*												
import		*			*	*						*		*	
export	*	*	*	*	*	*						*			*
feed	*		*						*				*		
search	*	*	*	*						*		*		*	
notifications	*	*	*		*		*							*	*
synchronization	*		*	*	*		*	*		*		*	*	*	*
keyboard shortcuts	*	*		*			*								
project management		*	*	*	*	*						*	*		
permissions		*	*						*				*		
task assignment		*	*			*	*	*	*	*				*	*
public sharing		*	*			*	*		*				*		
reports / statistics		*	*	*		*	*				*			*	
drag & drop			*	*		*				*			*		
file upload		*	*		*								*		
time tracking			*	*	*	*							*		*
multiple entry			*		*										
backup / restore			*		*	*									
periodic checklists												*			
booklet print			*		*										
voice recognition							*								
mobile	*		*	*	*		*	*		*		*	*	*	*
priorities / dependencies		*			*						*	*			

Table 2.2: To-do list applications' features

KISSFLOW

This application lets users build business workflows and it offers a simple and efficient way to automate workflows around their existing user base. It is pre-integrated with Google Docs and Gmail. Users can attach documents directly in their workflows and collaborate with others users and groups in their company. Some of the available features are permission levels, time tracking and statistics.

Pros	Cons
<ul style="list-style-type: none"> - Notifications - Social integration - Fully integrated with Google Apps - Permissions - Task priorities and dependencies - Graphical workflow builder 	<ul style="list-style-type: none"> - Social integration limited to Google - Mobile not supported

Comindware

This application is a collaborative work management solution that features workflow automation combined with task management, issue tracking, online collaboration, email integration, reporting dashboards, workspaces and API integration. It has a visual, interactive workflow design GUI with simple drag and drop controls.

Pros	Cons
<ul style="list-style-type: none"> - Process export - Task assignment - File upload - Time tracking - Mobile support - Task priorities and dependencies - graphical workflow builder 	<ul style="list-style-type: none"> - No permissions - No notifications

features	KiSSFLOW	Comindware
social network integration	*	
export		*
notifications	*	
permissions	*	
task assignment		*
public sharing	*	*
reports / statistics	*	*
drag & drop	*	*
file upload		*
time tracking		*
mobile		*
priorities / dependencies	*	*
data submission		
workflow control	*	
workflow automation	*	*
graphical workflow builder	*	*

Table 2.3: Workflow applications' features

2.1.3 Application conclusions

Summarizing, all the applications we have analyzed are rich of functions to help users in the management of their lives, but none of them has the characteristics we are searching for. First of all they do not have a full and robust integration with social networks. They do not support the management of series of tasks related by dependencies, except for KiSSFLOW and Comindware that however are business oriented and so are quite inapplicable in a personal context. This means that there is wide gap between BPM tools and the world of personal process management. There are a lot of to-do list applications focused on personal usage, however they are too far from what an integration of social BPM concepts in personal management would offer.

2.2 Academic works

In this section we will discuss about some of the publications that are inherent to this thread. Despite being an interesting and important issue, personal task planning has received limited attention from academic research so far. While BPM and social BPM are widely covered, we have found a few documents about Personal Process Management as BPM tools applied in daily life.

In a post on his blog, Dr. Michael Rosemann has been the first to suggest the idea of introducing the benefits, methods and tools of BPM into private life [1]. He have highlighted some applicable scenarios and have stated that surprisingly at the moment there have not been any corporations that have yet explored the commercialization of such personal processes. He however affirmed that it will be only a matter of time before some solution in this direction will come up.

The only structured research that can be found is reported in a Technical Report of University of New South Wales called “Personal Process Management: Design and Execution for End-Users” [2]. The report discusses a possible implementation of personal process management, presenting a novel approach for enabling end users to model and deploy processes they encounter in their daily work. The processes are modeled without compli-

cated constructs using a simple textual process representation. They stated that for example one person hardly ever pursues multiple tasks in parallel. The simplicity is achieved by allowing only few activity types in the process: filling forms, sending pre-formatted emails and filling HTML templates. The process models can then be translated to an executable format and be deployed, including automatically generated Web interface for user interaction. This paper is the only technical report about this topic, but it is based on simplifications done to facilitate its study. This is done because they start from the assumptions that personal needs are different from those of business. Hence they face the problem taking in consideration a single user and his personal process management, without consider multiple users, social network integration and tasks structure.

A first attempt to implement an application for the management of personal processes structured with the BPM principles has been done by Professor Marco Brambilla. In his paper “Application and Simplification of BPM Techniques for Personal Process Management” [3] some important features for the adoption of BPM into PPM are identified and highlighted. First of all a reduction in the expressive power of the BPM notation is needed, because it is too complex for end users. In second place it is necessary the introduction of social network integration to improve interaction, along with social sharing and gamification. Finally, ease of use, flexibility and productivity are critical aspect to reach in the application. After an informal investigation, he proposed a first syntax made up of only three elements: atomic tasks, sequential dependencies and parallel execution. In the conclusions he proposed to extend this first simple approach with something more structured, with the objective to isolate a reduced set of business process modeling construct acceptable and convenient for end users.

Chapter 3

Background

3.1 Business Process Management

Business Process Management (BPM) is an approach that focuses on optimizing business operations, both in the application and human aspect, to increase the performance, quality, cost and operating times.

Instead of looking at the functional organization of a company (production, accountability, marketing, etc.), BPM looks at the processes involved in each function (design, production, distribution, administration, control, etc.): it is based on the clear definition of processes.

A business process is a series of tasks or activities that produce a specific outcome. In many companies, business processes are informal and undefined. This often creates inefficiencies and bottlenecks when there is confusion as to employee responsibilities and company procedures.

The aim of BPM is to lead to a better overall view of all the company's work processes and their interactions in order to be able to optimize them as much as possible and automate them to the maximum through working applications.

3.2 Business Process Modeling

Business Process Modeling (BPM) consists in the representation of enterprise processes in order to analyze and improve them. It is generally used by analysts and managers and it can be assisted by informative tools to speed up repetitive actions.

A process model defines the behavior of a process and consists of a clear start, a number of tasks that need to be carried out, sequences and con-

ditions that determine the process flow, and a clear end. The scope of a complete process can involve one or more organization units.

Here follow some definitions that describe key concepts of this activity.

A *business model* is the representation of core aspects of a business made with a broad range of formal and informal descriptions.

A *business process* is a set of structured tasks that serve a particular product or service, or to reach a particular goal, for a particular customer.

There are three kinds of business processes:

- Management processes handle the operations of a system such as “Corporate Governance” and “Strategic Management”.
- Operational processes are the core business. Most common operational processes are purchasing, manufacturing, advertising and marketing.
- Supporting processes cooperate to support the core processes of the business. Accounting, recruitment, call center and technical support are examples of supporting processes.

A business process can be seen as a composition of several sub-processes achieving the goal of the super-process and can be modeled using several methods and techniques like Business Process Modeling Notation.

There are three types of business processes. Operational processes relate directly to the mission of the organization: they create the products or services that generate the organization’s income. Another type are the infrastructure processes that support the operational processes (for example, manage human resources). The last type are control processes that manage the operational and infrastructure processes, such as setting goals and monitoring results. All three kinds of models are closely linked. If it is needed to model more than one of them, it is better to model them separately to keep the models as simple and clear as possible.

There are many reasons why it could be needed to define business processes. Generally models are created to study processes trying to improve them in terms of efficiency and quality. A first step is the creation of the

as-is model through which analysts and managers understand and analyze the current way of working. Then come the modeling of the to-be model that is the redesigned and improved version of the starting process.

Process models are also useful in the definition of risks and controls or in the choice of roles and responsibilities for each task in a process, or they can simply be used as a communication tool to facilitate understanding across the organization.

3.3 Business Process Modeling Notation

Business Process Modeling Notation (BPMN) is an initiative that aims to define a common graphic notation to create models for business processes. The BPMN notation can be seen as a UML notation applied to business processes.

Each process model consists of:

- a start event
- an end event
- one or more tasks
- sequence flows that define the routing through the tasks

A task is an atomic activity that should have a well-defined input, a transformation and an output. The input of a task defines the state of the process just before the task starts. The output defines the state of the process after the task is complete. The transformation defines the changes the task do to the state of the process.

The size of a task is important: it should be such that it can be executed at one place in one attempt, otherwise it should be decomposed in other sub-tasks.

There are four types of routing that can be used while modeling a process:

- *Sequential routing* is the simplest one and it models sequential execution of tasks. There is generally a dependence between two sequential tasks.

- *Parallel routing* models two or more tasks or sequences of tasks that can be executed simultaneously. Parallel tasks are independent but they need to be completed to continue the process execution flow.
- *Conditional routing* is a way to control the paths of execution. When the flow reaches a conditional routing, a condition for each path is analyzed and the execution continues through the paths having satisfied conditions. BPMN allows event-based choice for routing which means that choice between tasks depends on an event.
- *Iteration* is similar to the conditional routing but it allows the repetition of one or more tasks until a condition is satisfied.

Huge processes with a large number of tasks become difficult to understand. For this reason their tasks can be grouped into sub-processes that can be modeled through separated diagrams. Each sub-process can in turn be divided into other sub-processes and so on. This mechanism of grouping and nesting processes is called hierarchical modeling. A general rule is to use a maximum of three levels.

Besides task and routing, BPMN supports also intermediate events. These elements are represented by a double circle with an icon that defines the event type:

- *Message*: it models the situation where a task can start only when a specific type is received
- *Timer*: it models the situation where a task can start only when a certain amount of time elapsed or by a specific date/time
- *Link*: it denotes that the completion of one process is the trigger for a task in another process
- *Rule*: it denotes that a task can start only when something outside the process meets a certain rule

In BPMN there are also intermediate events without an internal marking: these symbols are used to indicate a specific state of the process. States provide a useful point to collect management information about the key performance indicators such as waiting time and processing time.

When a process spans multiple organizational units or roles, BPMN makes the use of swim lanes. This technique shows relationships between process and organization structure and organizational unit or units responsible for each task.

3.4 Social BPM

Social BPM [4] is the integration of social software into BPM. This aims to increase performance by means of a controlled participation of external stakeholders to the design and execution of the process.

Social extension to BPM wants to improve the organization efficiency by exploiting social software potentialities. This can be done enhancing cooperation between people with consequential improvement in activity execution and dissemination of knowledge. Also mutual support among users is positively affected. Social functionality can also be used to increase process transparency and participation of people to the decisions by sharing their choices or feedback. Following this idea also informal communities can be involved in activity execution, assigning so the execution to a broader set of performers or to find most appropriate contributor. Summarizing, Social BPM believes that the constant sharing and tight contacts typical of the Social Web can make a significant contribution to the business world enhancing its processes.

In classic BPM, processes are defined centrally by the organization and deployed for execution by internal performers. This close-world approach can be opened introducing social features at different levels of control:

- *Participatory Design* opens the process design to multiple actors, including end users; the resulting process is then executed in the traditionally way;
- *Participatory Enactment* shifts socialization from design to execution, allowing limited user communities to gain visibility and limited participation to the process;
- *Social Enactment* opens the process execution to open communities of actors dynamically signed-up to the process.

All these aspects require the introduction of new elements into the business process notation and into development tools. These new features are well suited to describe classical business process models but are not sufficient to describe social aspects and informal interactions among people.

3.5 The Physics of Notation

Nowadays graphical languages are widely used in many fields to ease modeling and designing. The main reason of this success is that it is commonly accepted that a visual representation is more suitable also for novices. However, there are some aspects to take in consideration in order to obtain a cognitively effective language really claiming these goals.

The most accepted theory is the “Physics of Notations” [5] by Dr. Daniel Moody, who has identified a set of nine fundamental principles to evaluate and improve notations on a scientific basis.

These principles are:

- *Semiotic Clarity*
- *Perceptual Discriminability*
- *Semantic Transparency*
- *Complexity Management*
- *Cognitive Integration*
- *Visual Expressiveness*
- *Dual Coding*
- *Graphic Economy*
- *Cognitive Fit*

In his theory, Moody also describes the anatomy of visual notations identifying the characteristics involved in achieving these qualities.

He states that a visual notation consists of a set of graphical symbols (visual vocabulary), a set of compositional rules (visual grammar) and definitions of the meaning of each symbol (visual semantics). The visual vocabulary and visual grammar together form the visual syntax.

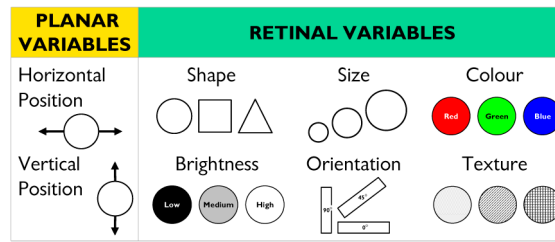


Figure 3.1: Visual variables

Graphical symbols are used to perceptually represent semantic constructs. Symbols are made up of visual variables combined together. Bertin’s “Semiology of Graphics” identifies eight visual variables divided into planar and retinal variables. The planar variables are the horizontal and vertical position, while the retinal variables are the shape, size, color, brightness, orientation and texture of the symbol (Figure 3.1).

3.5.1 Semiotic Clarity

There must be a one-to-one correspondence between symbols and their referent concepts. This is necessary to avoid the following anomalies:

- Symbol redundancy: occurs when multiple graphical symbols can be used to represent the same semantic construct.
- Symbol overload: occurs when two different constructs can be represented by the same graphical symbol.
- Symbol excess: occurs when graphical symbols do not correspond to any semantic construct.
- Symbol deficit: occurs when there are semantic constructs that are not represented by any graphical symbol.

3.5.2 Perceptual Discriminability

Symbols should be clearly distinguishable from each other. To measure the distance between symbols we can observe the number of visual variables on which they differ and the size of the differences. The size is measured by means of perceptible steps. Of all visual variables, shape plays a key role as it represents the primary basis on which we identify objects.

The greater the visual distance between symbols, the faster and more accurately they will be recognized. A good practice is to create visual elements having unique value for at least one variable (perceptual popout). On the other hand, symbols which are differentiated by unique combinations of values cause slower and error-prone understanding. Another important technique is redundant coding, which consists in using multiple visual variables to distinguish between symbols.

Using a textual differentiation is instead a bad choice, because text is not a dimension that helps increasing the visual distance. Moreover text processing makes the cognitive processes less efficient.

3.5.3 Semantic Transparency

Symbols should use a visual representation whose appearance suggests their meaning. Using intuitive symbols reduce cognitive load because they have built-in mnemonics: their meaning can be either be perceive directly or easily learned.

A symbol is *semantically immediate* if a reader is able to infer its meaning from its appearance alone.

A symbol is *semantically opaque* if the relationship between its appearance and its meaning is purely arbitrary.

A symbol is *semantically perverse* if a reader infers a different or even opposite meaning.

A simple solution to improve the semantic transparency is the use of icons. Icons resemble the concepts they represent, speed up recognition and recall, and make diagrams more accessible: people prefer real objects to abstract shapes.

Another aspect to be considered is the semantic transparency of the relationships between the elements. Notations in general use effective connection lines that, however, provide few clues to their meaning as they can be used to represent almost any type of relationship.

On the other hand, certain spatial arrangements of visual elements predispose people toward a particular interpretation of the relationship among them even before the meaning of the elements is known. This conveys the relationship among the entities in a more semantically transparent way than using arrows, so is more likely to be interpreted correctly and more easily remembered (Figure 3.2).

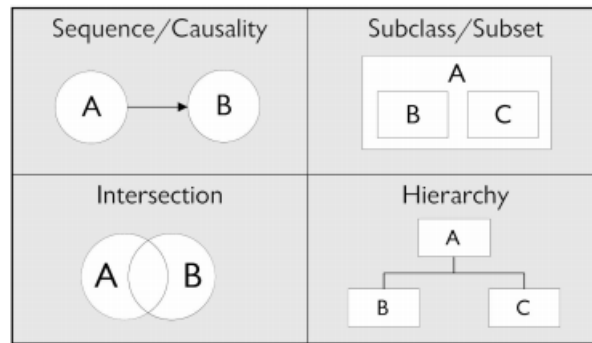


Figure 3.2: Semantic transparency of relationships between elements

3.5.4 Complexity Management

Complexity management is required to create a notation capable to represent information without overloading the human mind. Notation complexity can be measured by the number of elements on a diagram, not to be confused with the number of symbol types in a notation, that is called graphic complexity.

The key is to keep a notation as simple as possible because complexity has a major effect on cognitive effectiveness than on the amount of information conveyable. In fact the latter is limited by human perceptual and cognitive abilities.

Perceptual ability is the cleverness to discriminate between diagram elements and it increases with diagram size. By cognitive ability we mean the number of diagram elements that can be comprehended at a time. This capability is strictly correlated to working-memory capacity. Complexity management is particularly important when dealing with novices.

Some software engineering visual notation lack mechanisms for complexity management and, as a result, really complex diagrams are often produced. Modularization and hierarchically structuring can be practices of great impact trying to reduce complexity. The former divides a large system into more cognitively manageable smaller parts or subsystems. The latter represents a system at different levels of detail.

3.5.5 Cognitive Integration

This principle applies when multiple diagrams are used to represent a system. It is closely related to complexity management. Multi-diagram

representations, in order to be cognitively effective, must include mechanism to support conceptual and perceptual integration. Conceptual integration is a mechanism to help the reader assemble the information from separate diagrams in a coherent mental representation of the system.

A good technic is the contextualization: it consists of the inclusion of all directly related elements from other diagrams as foreign elements. Including the overlaps between diagrams allows to better understand each element in the system of diagrams in terms of its relationships to all other elements. Perceptual integration is achieved using perceptual clues that simplify navigation and transitions between diagrams such as clear labeling, level numbering and a navigational map.

3.5.6 Visual Expressiveness

In a notation, information-carrying variables are variables used to encode information. Variables not formally used are called free variables. The more are the information-carrying variables used in a visual notation, the most it is expressive. The inverse measure is the number of free variables, and it is called the degrees of visual freedom. Having eight visual variables means that there are eight degrees of both visual expressiveness and visual freedom. A notation that uses eight degrees of visual freedom is called nonvisual or textual, while a notation with eight degrees of visual expressiveness is visually saturated.

Being visual expressiveness a measure of the graphic design space, a notation using only a visual variable is said visually one-dimensional. Visual expressiveness can be used to improve discriminability by choosing the right number of variables (degree of expressiveness) to represent a certain quantity of information.

The choice of the variables to use must be made on their capacity and power compared to the information we want to represent. The capacity of a variable is the number of its perceptible steps and it should be greater than or equal to the number of values required. The power of a variable is the highest level of measurement that can be encoded and so it should be greater than or equal to the measurement level of the information (Figure 3.3).

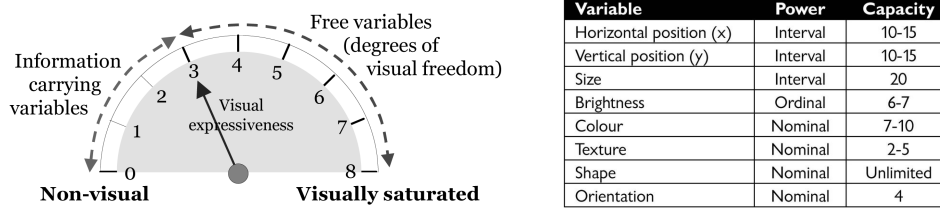


Figure 3.3: Visual Expressiveness

3.5.7 Dual Coding

The principle of dual coding is about reinforcing the cognitive effectiveness of the notation by adding complementary textual descriptions to graphic symbols. When information is presented both verbally and visually, representations of that information are encoded in separate systems in working memory and referential connections between the two are strengthened. Textual encoding is more effective when it is used to supplement rather than to substitute graphics.

One particular technique to achieve dual coding is the annotation, which consists in adding textual explanations directly on the diagram near to the symbol.

3.5.8 Graphic Economy

The graphic complexity of a notation is the number of graphical symbols of that notation. It can also be seen as the size of its visual vocabulary. Graphic complexity differs from notation complexity because it is at the type level rather than the token level. This means that a notation with a high graphic complexity can have a low notation complexity and vice versa.

Dealing with a lot of symbols affects novices the most, because they have to consciously maintain meanings in working memory. Empirical studies show that the human ability to discriminate between perceptually distinct alternatives is around six categories. So this defines an upper limit to graphic complexity. Software engineering notations tend to increase in graphic complexity in the effort to increase their semantic expressiveness by adding new constructs. An unwished effect of this trend is the loss of cognitive effectiveness.

Managing graphic economy could be quite challenging. The main cause of high graphic complexity is a big number of semantic constructs as different

constructs are usually represented by different symbols. Reducing semantic is therefore a simple way for graphic complexity reduction.

Another common practice is to introduce symbol deficit choosing not to show some constructs graphically. This directly reduces complexity without affecting semantics.

A third way consists in increasing human discrimination ability: this can be achieved by increasing the number of perceptual dimensions on which stimuli differ. This means that the six-symbol upper limit only applies if a single visual variable is used. Using multiple visual variables to differentiate symbols can increase human discrimination ability.

3.5.9 Cognitive Fit

The principle of cognitive fit states that different tasks and different audiences require different representations of information. Multiple visual dialects are necessary to overcome the differences between the problem solving skills of experts and novice, and deal with the task characteristics.

Representations must be understandable by both business and technical experts. While notation experts develop diagram schemas in long-term memory which largely automates the process of diagram interpretation, non-experts are slower to understand the notation and tend to make errors. Novices face difficulties in symbols discrimination, they are more affected by complexity and have to consciously remember the symbols meanings.

It is not possible to use a single notation optimized for one of the two sides: it can reduce its effectiveness for the other side (expertise reversal effect). The best solution is to have at least two different visual dialects: an expert and a novice one. Notations designed for novices will have more discriminable symbols (perceptual discriminability), reduced complexity (complexity management), more mnemonic conventions (semantic transparency), explanatory text (dual coding) and simplified visual vocabularies (graphic economy).

Another situation that requires different visual dialects is the use of different representational media: requirements for sketching on whiteboards or paper are different to those for using computer-based drawing tools. Some notational requirements must be taken into account: perceptual discriminability, due to the variations in how symbols are drawn by different people; semantic transparency, due to the difficulty in drawing pictures and icons

	Semiotic Clarity	Perceptual Discriminability	Semantic Transparency	Complexity Management	Cognitive Integration	Visual Expressiveness	Dual Coding	Graphic Economy	Cognitive Fit
Semiotic Clarity								±	
Perceptual Discriminability					+				+
Semantic Transparency		+							±
Complexity Management								-	+
Cognitive Integration		-		+				-	
Visual Expressiveness		+						+	±
Dual Coding									+
Graphic Economy		+		+			-		+
Cognitive Fit									

Figure 3.4: Interactions among principles

with respect to simple geometric shapes; visual expressiveness, because some visual variables like color and textures, are more difficult to use.

3.5.10 Interactions among principles

Some principles are in conflict, other support each other, so interactions should be studied in order to make trade-offs and exploit synergies. Moreover interactions are not necessarily symmetrical. Figure 3.4 summarizes these interactions among the principles.

Some important interactions are:

- *Semiotic Clarity vs. Graphic Economy*: symbol excess and redundancy increase graphic complexity but symbol overload and deficit reduce it.
- *Graphic Economy vs. Perceptual Discriminability*: increasing the number of symbols makes it more difficult to discriminate between them.
- *Perceptual Discriminability vs. Visual Expressiveness*: the use of more visual variables and a wider range of values increases both Perceptual Discriminability and Visual Expressiveness.
- *Visual Expressiveness vs. Graphic Economy*: Graphic Economy defines limits on Visual Expressiveness and Visual Expressiveness reduces graphic complexity.

- Perceptual Discriminability, Complexity Management, Semantic Transparency, Graphic Economy and Dual Coding improve effectiveness for novices but, according to the Cognitive Fit principle, Semantic Transparency can reduce effectiveness for experts.

3.6 OAuth

OAuth (Open Authorization) [6] is an open protocol that allows secure authorization in a simple and standard method from web, mobile and desktop applications. It allows authorized third parties applications to access users' data and perform actions in their place without knowing their usernames and passwords.

The actors involved in an OAuth authorization request are:

- *Service provider*: the web service in which the protected resources are stored.
- *User*: the one who wants to share his protected resources with a consumer without making them public.
- *Consumer*: the web application who wants to access to the user protected resources.
- *Protected resources*: the objects protected by the OAuth protocol, on which it grants the access and permits.
- *Access and request token*: the elements used in place of the user credentials.

The complete OAuth authorization flow (Figure 3.5) is as follows.

First of all the consumer must register itself with the service provider, obtain an API key and a shared secret and keep them saved. It is important that the shared secret is sent through a secure channel.

When the user makes a request to the consumer, the consumer prepares an OAuth request with its API key and redirects the user to the service provider. The service provider presents to the user a login page with the specifications of the consumer request and asks him the authorization to

proceed. If the user grants it, the service provider answers with a request-token and redirects the user back to the consumer page. At this point, the consumer sends its API key and shared secret with the request token, asking the service provider for an access token. The service provider verifies data and sends the access token.

Finally the consumer can use the token to perform requests of user protected data to the service provider.

The OAuth protocol introduces many advantages:

- it allows quick login without the need to create a new account
- users can control customer accesses and permits to their protected data
- there is no need to redesign a more robust authentication system every time
- the login simplicity and velocity encourages users to visit and test new websites
- it is a standard and all the data transfers take place on SSL
- it is well known and adopted by all the major social networks

However, OAuth protocol has some disadvantages:

- it lacks of anonymity, because users are always obliged to use their personal data
- accustoming users to login to different sites with their credentials can be dangerous and make them think that this practice is always safe
- since we are using a unique account for many sites, if we lose it or decide to close it, then there will be serious impacts across all these sites
- if it becomes a permanent standard, everyone would be forced to use this method without alternatives

3.7 Draw2D

Draw2D [7] is a JavaScript library that allows creating graphs, diagrams and workflows inside an HTML5 canvas. It is based on *jQuery* and *RaphaelJS*, it is platform independent, and it has a fast SVG rendering.

It is well supported by all modern browsers like *Chrome*, *Firefox*, *Safari*, *IE9+* and *Opera* and it is compatible with mobile devices. Compared to *Raphael*, *Draw2D* simplifies the creation and management of figures thanks to its powerful API that allows writing less code to obtain more flexible results. Elements are designed as classes with inheritance support, simulating an object oriented coding style, so it is really easy to extend figures, adding new functionality, and to modify their appearance.

Draw2D offers extendible basic vector shapes like *Rectangle*, *Circle*, *Oval* and *Diamond*. Each one can be graphically customize changing dimensions, background and border colors and other parameters.

All figures are treated like nodes inside the canvas. Each node can have input and output ports that allow to connect nodes with lines using the drag and drop paradigm. *Draw2D* also supports images and labels, and allows to import and export diagrams in JSON format, so it is easier to save and load processes from the database. It also implements some useful functionality like zooming in and out on the canvas and a commands stack to keep trace of performed actions that behave like undo and redo system.

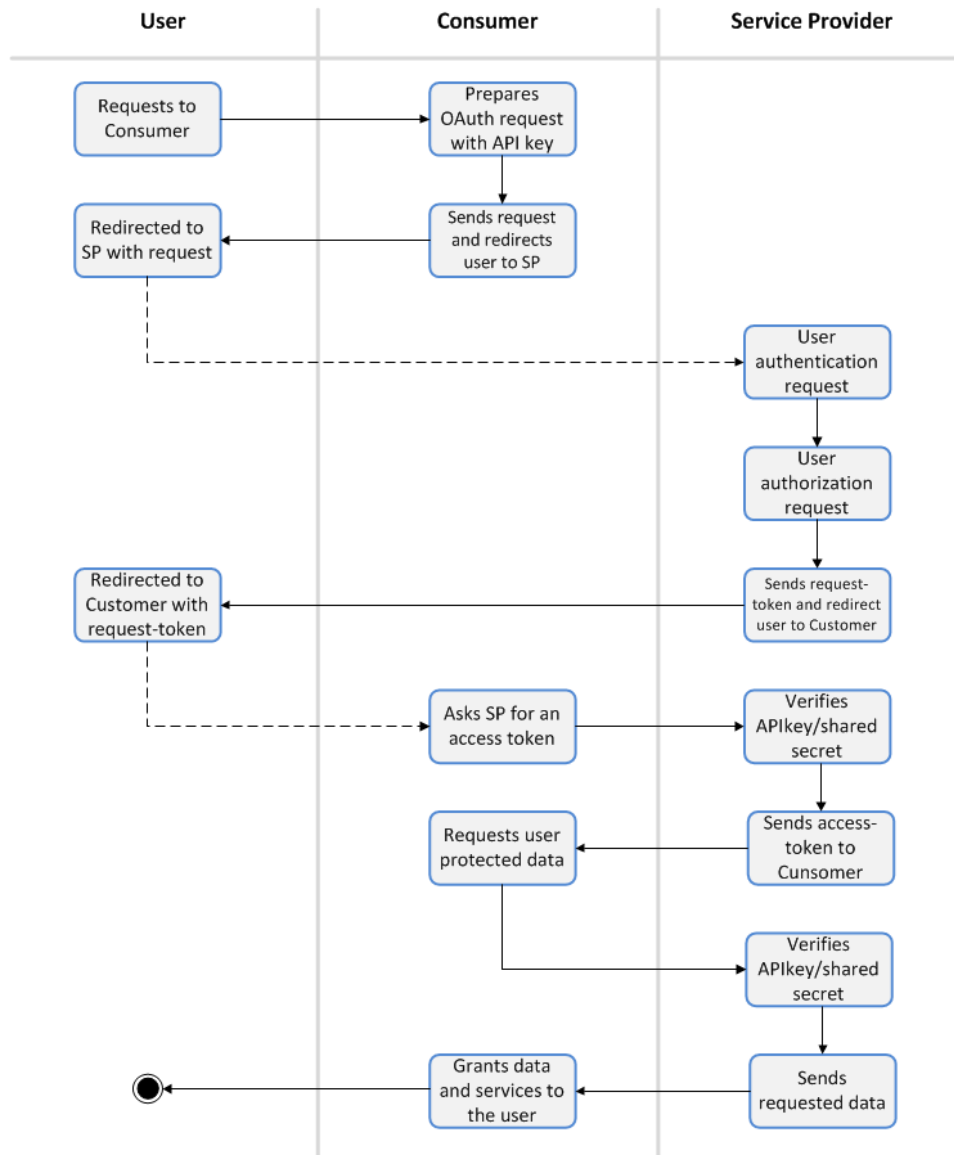


Figure 3.5: OAuth authorization flow

Chapter 4

Design

4.1 Description

In Chapter 2 we have listed and described many web applications for the management of workflows and to-do lists. In Chapter 3 we have introduced the BPM and its notation. From these analyses it results that the applications managing the to-do lists have many restrictions and do not allow specifying relationships between tasks other than the simple temporal sequence. Business process management notation instead is well structured, but it is too focused on enterprise environment and it is not immediate to common people. Our thesis will try to merge these two worlds and find a way to manage the personal processes with a simple but expressive notation.

Our idea is to develop a web application for the management of personal processes in collaboration with other people, exploiting social networks capabilities. Users can log into the application through the OAuth protocol, using one of the social networks they are already registered to. Once they are logged in, they can connect the application with their other networks and then they can import friends building their own contacts list. The added friends then will be available to be assigned to tasks.

At this point users can start creating a process. A graphical editor will be needed to let them visually model the actions and put them in a sequence, assign a friend to each task and maybe express conditions or impose time constraints. Once they have modeled the process, it will be saved in the database and will be ready to be started and executed.

When the user will start a process, the first task will be read by the

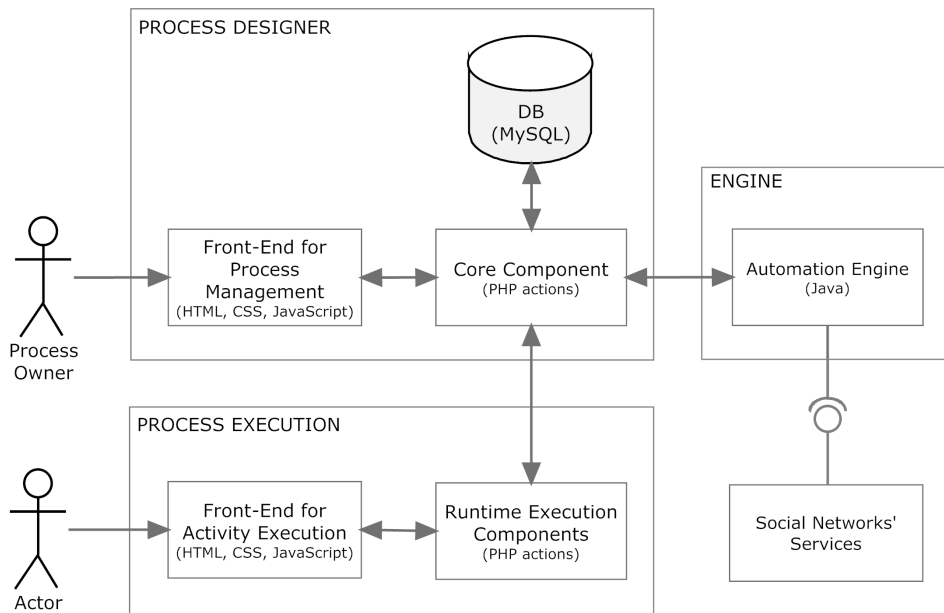


Figure 4.1: System architecture

system and a notification will be sent to the person assigned to it, informing him to do the particular action. Once performed, this person must confirm the completion by clicking on a special link. A workflow engine will be in charge of coordinate the execution flow of the process and send the notifications.

Figure 4.1 explains the architecture of the whole system.

4.2 Requirements

As introduced above, the goal of our thesis is to find the connecting link between the business process management and the to-do list management. In order to reach our purposes, some general requirements must be satisfied:

- easy, intuitive and expressive notation
- graphic modeling of the process
- automatic coordination and control of the execution flow of the process
- socialization must be exploited

To achieve the notation requirements we have planned to take as a reference the BPMN specifications and then to adapt them to our scope. We also have decided to keep in mind the analysis and the guidelines exposed by Moody in his work *Physics of Notation* in order to build a consistent and valid notation.

We have thought that a graphic editor would be the better solution to organize tasks and to model the processes: it offers a great level of interaction and it allows moving around the elements and positioning them in the way the user prefers. On the other hand, having too elements, the modeled process could be visually too heavy, and letting the users have too much freedom can be somewhat disorienting. These could represent critical problems, but since our targets are quite small processes we have thought that we can control them.

In order to coordinate the tasks and follow the right execution flow we have planned to use a workflow engine. The idea is to pass the modeled process to the engine, and then let it manage the notifications and the precedences between tasks.

Socialization can be achieved with social networks integration. A first step is to use the social login feature to access the web application. This is possible using the OAuth protocol implemented by all the major social networks. Using it, users have not to register a new account and then can benefit of contacts import and notifications. Then we can exploit also the API functionality offered by the social networks to request and exchange data.

4.2.1 Use case

There are three actors involved in the usage of the application:

- **User:** he is the user that visits the site for the first time. He can only choose the social network to use for the first login.
- **Logged User:** he is the user that has already logged in using a social network. He can join other social accounts, import and edit contacts, create and manage processes.
- **Actor:** the actors are people to whom tasks are assigned. They receive

activities to complete and they can only report tasks completions and submit required parameters' values.

Figure 4.2 shows the use case diagram of the actors' interactions with the system.

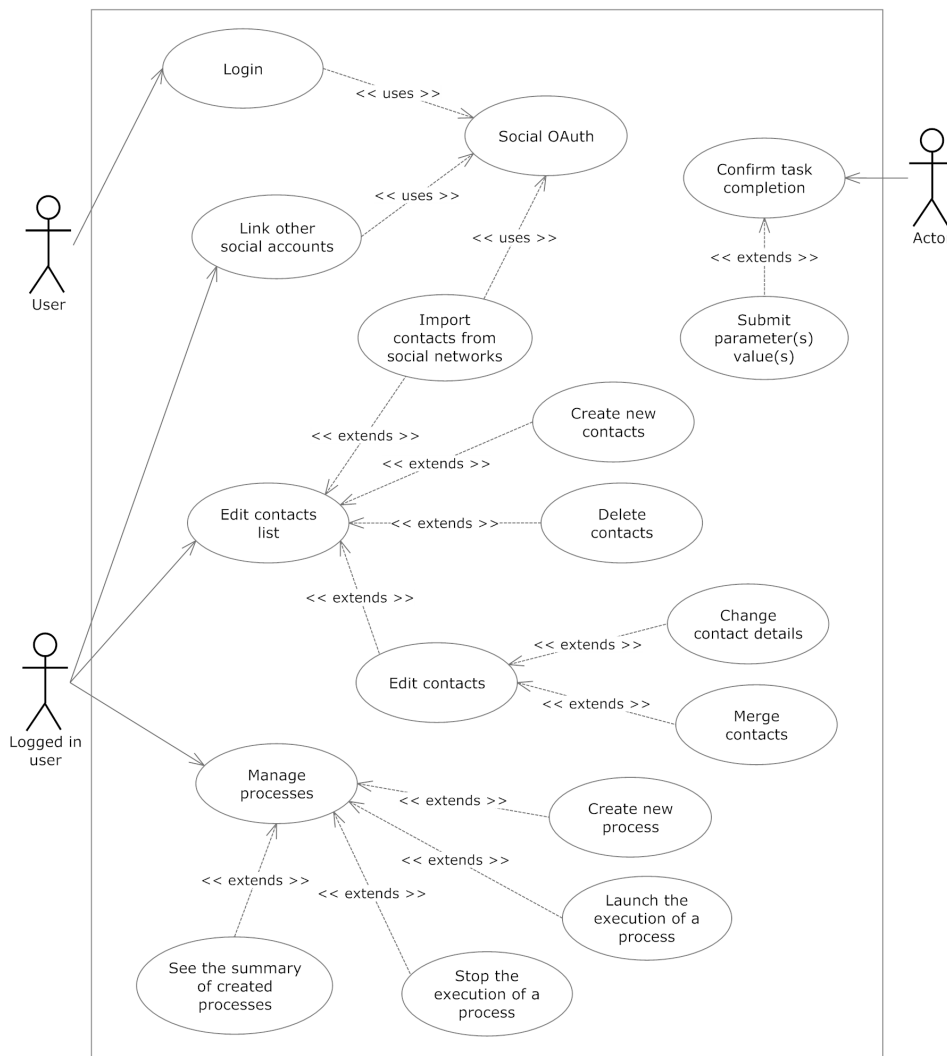


Figure 4.2: Use case

4.3 Modeling syntax

4.3.1 BPMN Analysis

The first step to define the PPM notation has been the analysis of the elements defined by the BPMN, in order to identify the conceptual features valid to reach our purpose.

The concept of the task stands at the base of the process modeling: a task represents an atomic action that, in a sequence together with other tasks, allows achieving the process goal. So, a task is the basic block around which the process is built and it is an obliged choice.

A task represents a state of the process at a given time. It receives an input, coming from a previous task, and produces an output. We have chosen to model inputs and outputs like optional parameters receivable and craftable by a task. These variables could be useful to interact with tasks and users assigned to tasks.

Another basic concept we have identified, is the concept of sequence between tasks: tasks have relationships between themselves, such as temporal priorities. The sequence concept represents the easiest type of relation and so we have chosen to maintain it.

A point of strength of the BPMN is represented by gateways. These elements allow expressing more advanced relationships among tasks: a gateway divides the linear execution flow of the process. There are three main types of gateways: parallel, conditional and loop.

The parallel gateway permits the simultaneous execution of tasks starting after it. This concept is powerful, and if applied in the right way, it allows saving a large amount of time. The parallel execution is easy to understand and for this and the previous reasons, it might be very useful in the modeling of personal processes.

The conditional gateway is a bit harder to comprehend, especially for people who are not confident with the concept of variables. This element allows splitting the execution flow along one or more of his branches, as long as the conditions declared on them are satisfied. There are some variations of this element available in the way it manages conditions: exclusive, inclusive and binary. Since these differences are too specific and could confuse a common person, we have decided to consider the inclusive conditional gateway only.

The last type of gateway is the Loop. This is the one who needs the most attention, mostly regarding the duration and deadlines management. The loop, in fact, allows the execution flow to go backward to a previous state of the process. If it is used in conjunction with the other two types of gateway, an execution flows overlap could occur. Although these drawbacks, we have decided anyway to include the loop in our notation for its expressive power. However, we have decided to impose some restrictions on it, in order to avoid or to reduce these problems.

Other types of elements used in BPMN are the intermediate events or triggers. Among all the types of events, we have chosen to keep only the timer event, because it could be useful to impose time constraints on the execution flow. In particular we have chosen to introduce two different time events: one to express the waiting of a generic period of time, one to express the waiting of a specific date or time.

BPMN provides also other elements, like for example annotations and swimlanes, that we judged to be unnecessary in the description of daily life processes and too difficult to be used by ordinary people.

4.3.2 PPM Notation

From the preliminary analysis we have defined in a more rigorously way the elements behaviors.

Task

A Task (Figure 4.3) defines an atomic activity included within a Process, and it must be assigned to one or more Actors. It is represented with a rectangle with the title of the activity written in it. Some icons tell which properties of the task have been set.



Figure 4.3: Task element

Attributes of a task are:

- Title (string): the name used by the admin to identify the Task.

- Actors (list): people to which the Task is assigned. They must complete the task following orders and respecting any time constraint.
- Body (text): the message for the actor(s) containing detailed description of what he/they have to do in order to complete the Task.
- Time constraint (optional): it indicates the maximum time Actors have to complete the job. It can be a quantity of time (minutes, hours, days) or a date.
- Parameter(s) (optional): if Actors are supposed to produce an input, a Parameter must be declared. It can be of two types: string or number.

Icons (Figure 4.4) used in Tasks are:



Figure 4.4: Task icons

- Human Shape/Shapes: it appears on a Task if there is/there are actor/s assigned to that Task.
- Alarm Clock: it indicates that a time constraint for the Task has been set.
- Box with arrow (only with global parameters): if a Task leads the Actor to create a global parameter, a box with an incoming arrow appears on it. On the contrary, if the box has an outgoing arrow, it means that a global parameter is read (its value is printed in the Task's body).

Sequence



Figure 4.5: Sequence connection

Sequence flow of execution is represented by arrows (Figure 4.5). An arrow connecting two tasks means that the task at the start of the arrow must be completed before starting the execution of the task at the end. The destination cannot be an element preceding the source.

Parallel Execution



Figure 4.6: Parallel opening and closure

Parallel Gateways create parallel flows for a simultaneous execution of tasks. A Parallel Gateway has a unique incoming arrow and two or more outgoing arrows. The execution of parallel branches starts when the task directly before the gateway is completed. Being possible to use one and only one End event per process, multiple flows must be joined before the end. In the case of parallel flows, this can be done with Parallel Closing Gateways. A Closing Gateway has two or more incoming arrows and one outgoing arrow (Figure 4.6). The execution of the Task to which the outgoing arrow is connected starts only when all the parallel branches are completed.

Conditional Execution



Figure 4.7: Conditional opening and closure

The only way to make conditional execution is to create conditional parallel flows using Conditional Gateway. A Conditional Gateway has a unique incoming arrow and two or more outgoing arrows. The execution of each parallel branch starts only when the task directly before the gateway is completed and only if the condition expressed on the branch is verified to be true. For the sake of simplicity, only simple conditions can be used: arithmetic operations or comparisons between two parameters are not allowed. Being possible to use one and only one End event per process, multiple flows must be joined before the end. In the case of conditional parallels

this can be done with Conditional Closing Gateways. A Conditional Closing Gateway has two or more incoming arrows and one outgoing arrow (Figure 4.7). The execution of the Task to which the outgoing arrow is connected starts only when all the parallel branches with true conditions are completed. In the case of global parameters, conditions can be expressed on all the parameters created by all the Tasks directly or indirectly connected to the Conditional Gateway. In the case of local parameters, conditions can be expressed on parameters created in the Task directly connected to the Conditional Gateway, or on parameters properly propagated through Tasks indirectly connected to the gateway.

Parameters

The Parameters are used to store values submitted by Actors. These values can be used in following Conditional Gateways, to be printed in following Tasks bodies or just to be stored. We have identified three possible ways to use parameters: global, one-local and multiple local.



Figure 4.8: Global parameters

Once they have been created, Global Parameters (Figure 4.8) are available in all the processes directly or indirectly connected after the creating Task. If one or more Global Parameters are created in a Task, they must be represented with an icon of a box with an incoming arrow. If they are read, a box with an outgoing arrow is used instead.



Figure 4.9: One-local parameter

One-Local Parameters are only available in Tasks to which they have been propagated. A small square at the base of the outgoing arrow means

that a single local parameter is being propagated from that Task. A small square where the arrow hits the Task means that a single local parameter is being received from that Task. The names of the parameters are written right above the squares (Figure 4.9).

Also Multiple Local Parameters are only available in Tasks to which they have been propagated. The difference is that in this case more than a parameter can be propagated. No icons or squares are needed: their names appear on the outgoing connection of the Task.

Apart from this distinctions, parameters can accept two types of values: textual and numerical. The type is chosen by the creator of the project.

Loop

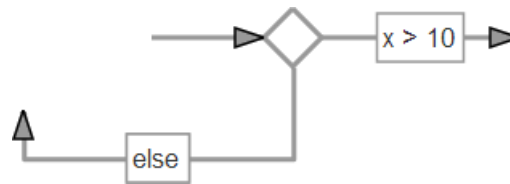


Figure 4.10: Loop element

The Loop is a specific tool to create backward flows. It has one input arrow and two output arrows, one going forward and one going backward. The expressed conditions are mutually exclusive: this means that the flow can continue only on one direction. Once a condition is declared on a branch, the other one is intended to be the complementary condition (Figure 4.10).

Events

An event is something that happens during the course of a process. There are three kinds of Events: Start, Intermediate and End.

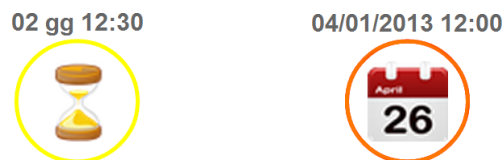


Figure 4.11: Intermediate events: Wait For and Wait Till

Intermediate Events are Time Events. They have one input arrow and one output arrow and they occur between a Start and an End Event. A

Time Event is activated when the Task before it is completed, and it will not start or terminate the process. There are two type of Time Events: Wait Till and Wait For (Figure 4.11). When the flow reaches one of these events, it will stop until the time or the date expressed has elapsed. Then it continues normally.



Figure 4.12: Start and End events

The Start Event indicates where a particular process will start. The End Event indicates where a process will end (Figure 4.12).

4.3.3 Restrictions

We have said that we want to achieve an easy notation, so we have imposed some constraints to avoid users making errors or coming up with invalid diagrams. These restrictions can be seen as easily remembered rules that guarantee the correct modeling of the process.

First of all we have imposed that each element, except for the Start and the End events, has only one input and one output ports. This means that each element can accept only one incoming connection and can have only one outgoing connection. Only gateways make exception of this rule.

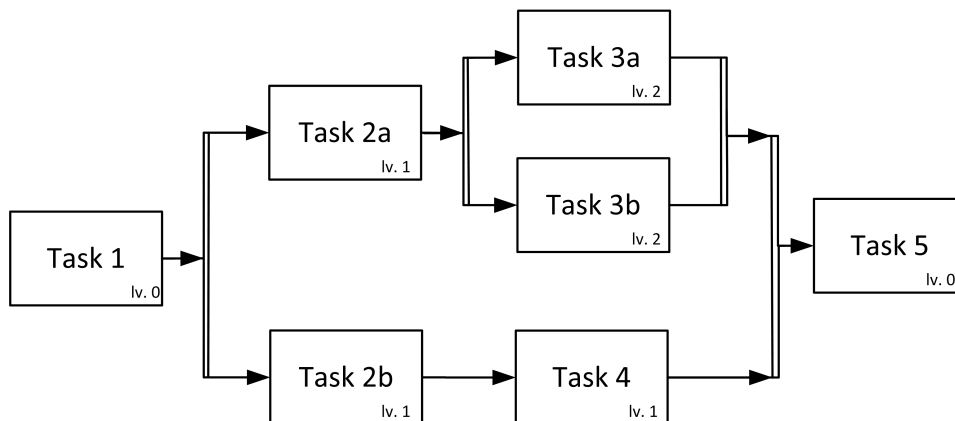


Figure 4.13: Blocks levels

An important constraint is represented by the block structure, along with depth levels (Figure 4.13). The Start and End events belong to the

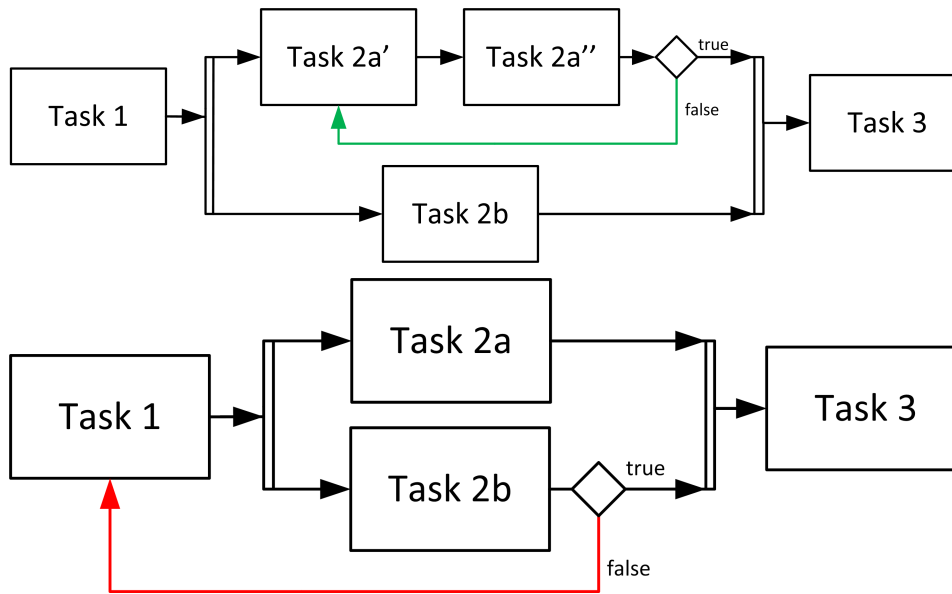


Figure 4.14: The first diagram is syntactically correct while the second presents a blocks structure violation: the loop backward connection points to a task with a different level

zero level. Whenever a parallel or conditional gateway is opened, a new block is created and the level of depth increases. On the contrary, when a parallel or conditional gateway is closed, the block is closed too and the level of depth decreases. So a block can be defined as a subpart of the process, comprised between a parallel or conditional gateway opening and its closure. Connections between elements of different blocks are prevented: you cannot connect a task of level one with a task of level three and vice versa. This is done in order to avoid possible inconsistent states or execution flow overlap. The Loop does not create new blocks neither it changes the depth level, but its outgoing branches must be connected with elements of its level.

Figure 4.14 shows some example of wrong modeling and highlight some of the problems just described related to the block structure.

As a consequence of the block structure, it follows that every gateway opening and its correspondent closure must have the same number of outgoing and incoming branches. If this does not happen, it means that there have been wrong connections together with a block violation or that there are blocks which have been created but no closed.

We have imposed restrictions also on parallel and conditional gateway

openings: it is not possible to open cascading gateways. Here we have to distinguish all the cases. Opening a parallel gateway right after another parallel gateway is useless: this scenario can be better modeled using only one parallel gateway which has the branches of both gateways. A cascade of conditional gateways can be resolved in the same way. Mixed sequences are trickier. A conditional gateway immediately followed by a parallel one, can be replaced by a unique conditional gateway with two or more branches with the same condition, so that the flow can proceed simulating a parallel execution. The last case is the parallel gateway followed by a conditional one. In this case, it is not possible to replace it with a single gateway and, more importantly, it becomes less immediate to use local parameters to express conditions. In order to maintain consistency, we choose to avoid cascade openings: at least a task must be placed between them. Cascading closure is instead accepted because they cannot lead to misunderstandings (Figure 4.15).

4.3.4 Moody's principles

Now that we have proposed the syntax, we have to analyze it from the point of view of Moody's Principles. First of all we analyze the basic shapes of the elements, listing their visual variables and observing if they carry any semantics (Table 4.1). From the resulting tables it is clear how the Visual Distance among all the elements is respected giving a good value of Perceptual Discriminability. In fact, each element has completely different basic shapes and each sub-event differs in color and texture. A good result in this direction is also helped by the global simplicity of the target, in particular by the small number of semantics that must be encoded.

As said in a previous chapter, there are three types of event. Table 4.2 is an analysis of such variations.

Semiotic Clarity, the first Moody's Principle, is respected by all the elements. Events share the circular shape, while tasks are rectangular. Parallel flows are represented with their own gateways, a vertical bar and a diamond respectively, and connections among these elements are represented with arrows. Furthermore, each event subtype has its own color and its own texture. Events' colors and textures are also in compliance with the Principles of Perceptual Discriminability and Semiotic Transparency. The Start has green color and a triangular symbol, properties that are both commonly


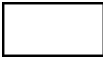



Metaclass	Symbol	Visual variables values	Semantics carrier
Event		(x,y): variable shape: circle color: yes brightness: NA size: fixed orientation: NA texture: border line	no yes yes NA no NA no
Activity		(x,y): variable shape: quadrilateral color: black/white brightness: NA size: fixed orientation: NA texture: border line	no yes yes NA no NA no
Sequence Flow		(x,y): variable shape: arrowhead link color: black/white brightness: NA size: variable orientation: NA texture: single line	no yes yes no no no no
Parallel Flow		(x,y): variable shape: quadrilateral color: black/white brightness: NA size: variable orientation: NA texture: border line	no yes no NA no NA no
Conditional Flow		(x,y): variable shape: quadrilateral color: black/white brightness: NA size: fixed orientation: NA texture: border line	no yes no NA no NA no

Table 4.1: Elements analysis





Metaclass	Symbol	Visual variables values	Semantics carrier
Start		(x,y): variable shape: circle color: green brightness: NA size: fixed orientation: NA texture: start triangular icon	no yes yes NA no NA yes
End		(x,y): variable shape: circle color: red brightness: NA size: fixed orientation: NA texture: stop squared icon	no yes yes NA no NA yes
Wait Till		(x,y): variable shape: circle color: orange brightness: NA size: fixed orientation: NA texture: calendar icon	no yes yes NA no NA yes
Wait For		(x,y): variable shape: circle color: yellow brightness: NA size: fixed orientation: NA texture: hourglass icon	no yes yes NA no NA yes

Table 4.2: Events analysis

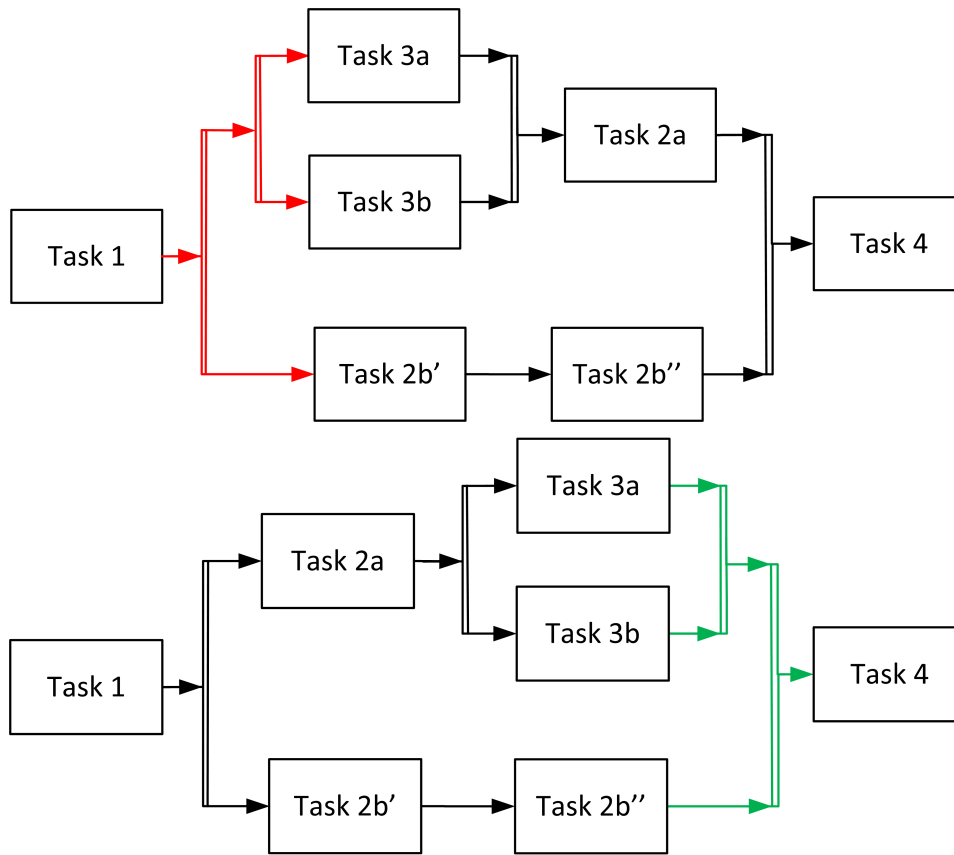


Figure 4.15: Wrong cascading gateways openings and correct cascading gateways closures

used to represent the “start” concept. The End is in red with a squared symbol and, also in this case, those are elements commonly used to represent the “end/stop” notion. Wait For and Wait Till are colored in yellow and orange, colors that communicate “warning” and so they are suitable for events that put the process in pause waiting for something. The Wait For icon is an hourglass and it clearly expresses the notion of waiting for a determined quantity of time. The Wait Till icon is a calendar and it is a good choice to express the concept of waiting till a chosen day. Arrows to model sequence flow of execution are a good and quite foregone choice. This is true also according to Winn’s experiments on spontaneous interpretation of diagrams. According to the Dual Coding Principle, text elements are used to enrich graphical elements of information. Tasks have titles, local parameters and conditions are written as labels near arrows, and Time Events have

the amount of time to wait for or the day to wait till written above the element. Complexity Management and Graphic Economy Principles are not a problem because of the requirements. All the work is based on the idea to create a tool for personal processes or small business processes and usable by all with simplicity. The other Principles are irrelevant for this thesis.

4.3.5 Building the four syntaxes

Above we have listed all the elements that we have thought to be useful in our notation. In order to better understand if the syntax we have identified is valid and suitable, we need to let it tested by users.

Here we face a delicate situation, which is how to submit the elements to the users so that they can evaluate them in the best way. Using a unique syntax with all symbols, for all users, is not possible. In fact there are three types of parameter management and they cannot be used at the same time. Moreover, taking into account the target of our experiments, using all the elements may be too confusing. A possible solution is to fix a number of elements and then take all the possible combinations, but having too much alternatives could be a problem. In fact we would need a large number of people to test all of them. For these reasons, we have decided that a good compromise would be to have four syntaxes.

All the four syntaxes have the basic elements in common: the start and the end events, the task and the sequence routing. We built a linear syntax with global parameters, events, parallel routing but not conditional and loop gateways. We also built a more complex syntax with multiple local parameters, conditional routing but no parallel gateway. The other two syntaxes were instead more balanced. Being the Loop probably the most critical element, we chose to include it only in one syntax, excluding the time events to prevent problems in time management.

The four syntaxes are:

- Syntax 1: it makes use of sequence and parallel flows, events, global parameters, but no loops and no conditional flows.
- Syntax 2: it makes use of sequence, parallel and conditional flows, global parameters, loops but no events.
- Syntax 3: it makes use of sequence, parallel and conditional flows, events, one local parameter, but no loops.

Element	Syntax 1	Syntax 2	Syntax 3	Syntax 4
start	x	x	x	x
end	x	x	x	x
task	x	x	x	x
global parameters	x	x		
one local parameter			x	
multiple local parameters				x
events	x		x	x
sequence	x	x	x	x
parallel gateway	x	x	x	
conditional gateway		x	x	x
loop gateway		x		

Table 4.3: The four syntaxes

- Syntax 4: it makes use of sequence and conditional flows, events, multiple local parameters, but no loops and no parallel flows.

In Table 4.3 are listed the four syntaxes with their constitutive elements.

4.3.6 Application Behavior

Each task must have at least one actor. During the execution of the process, actors will receive notifications in the form of messages through social networks or emails. Task orders will be sent to actors in the right moment telling them to start the activity and how to do it. In the message there will be a link to follow in order to report the task completion. If the actor is asked to insert any value, there will be a link to a page where he can write any input and report that task as completed.

As we said, a task can be assigned to more than one actor. From the executional point of view, multiple assignment can be handled in three different ways:

All. The task must be performed by all the designated actors. The next task will be available only when all the actors have fulfilled the current activity. This scenario is nothing but a simplified notation of a parallel execution of the same task done by different people. The first starting. In

this scenario the admin assigns the task to multiple actors but only the first one who confirms its execution will perform it. In this case it is necessary the implementation of a mechanism to let users to accept and confirm the task execution. Also it is necessary to introduce a timer within which users must confirm the activity. Furthermore if no one accepts the activity, the entire process may be interrupted and the admin have to choose a random user and assign him the task. Due to these issues, this scenario is discarded. The first ending. In this last scenario the admin assigns the task to multiple actors and all can start to work on it. The task is completed when an actor finishes it first, though other users are still performing it. This case is quite inefficient and so it is discarded too.

In conclusion, multiple assignment can speed up the modeling process by simplifying the representation of multiple execution of the same task by different people. However it makes parameters handling more complex. If a task with multiple actors asks to insert a value, there are three ways to do it. The first one is to handle users' inputs as an array of parameters but this will make its usage too complex, especially in building conditions. A second way is to admit only one input from all the actors. This solution inevitably assumes cooperating actors, and would fit better with a multiple "first ending actor" assignment. A last solution could be to keep the last value submitted, admitting a sort of parameter overwriting. Also this solution needs actors' cooperation and it is necessarily less tight.

Speaking about parameters, another critical issue exists. If a parameter is created in a task inside a block opened by a conditional gateway, it is not obvious to assume its existence outside of that block. In fact its existence is certainly known at runtime only, when the gateway condition will be evaluated and the parameter eventually created. This aspect is really tricky, but a way to resolve it does exist: the introduction of a default value for every parameter. The default value must be set by the creator of the process and it will allow maintaining consistency.

Up to this point we have assumed that an actor will perform its assigned task in the set time interval. But what happens if he doesn't? A simple solution might be to send again the original notification, or a message informing him that the time available is running out. In the event that the actor does not complete its task, the unique solution is to inform the admin, so that he can assign the task to an other people or stop the process. Some more

advanced technique might consist in introduce safety times between tasks, letting the actors and the admin have more time to manage this kind of situations, or give the possibility to choose in advance some possible substitutes for each task.

4.4 Engine

We said that notifications and process execution flow would be managed automatically by the system. In particular this will be possible using a workflow engine. Instead of building a new engine we have planned to choose it between some already available implementations. They are typically built in Java and require a mapping of the process in order to correctly work.

The execution of a process instance is similar to a finite state machine, where it responds to external events changing its state.

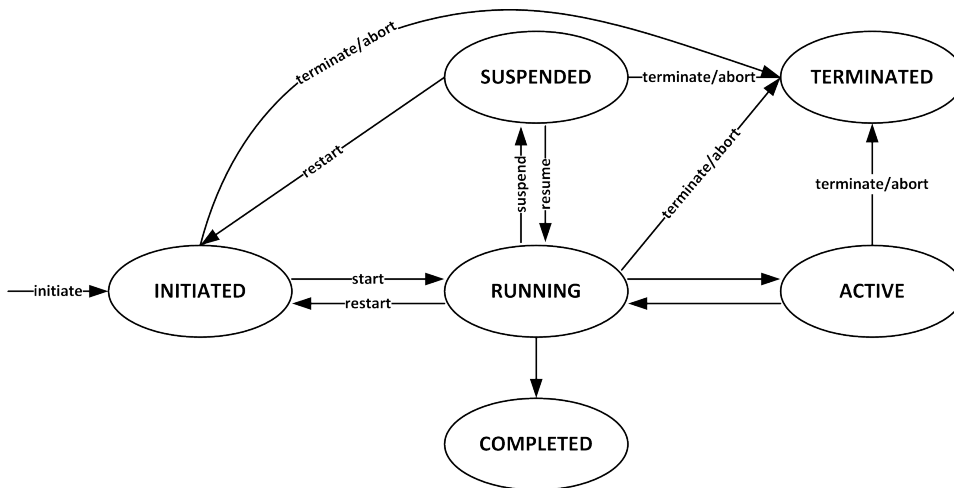


Figure 4.16: Process states

At any time during the execution, as shown in Figure 4.16, the process can be in one of these states:

- **Initiated:** the instance of the process has been created but the execution has not started yet.
- **Running:** the instance of the process has started the execution and any of the activities may be started.
- **Active:** one or more activities of the process instance have been started.

- Suspended: the entire execution has been suspended and no activities can be started until the process returns in the running state.
- Completed: the process instance has fulfilled the conditions for completion and no more activities will be executed.
- Terminated: the execution of the process has been stopped before it could reach the completed state.

Also tasks can switch through different states like shown in Figure 4.17.

- Inactive: the task is not currently been processed.
- Active: the task instance is running.
- Suspended: the task instance is blocked and is waiting to be resumed.
- Completed: the task instance has been correctly fulfilled.

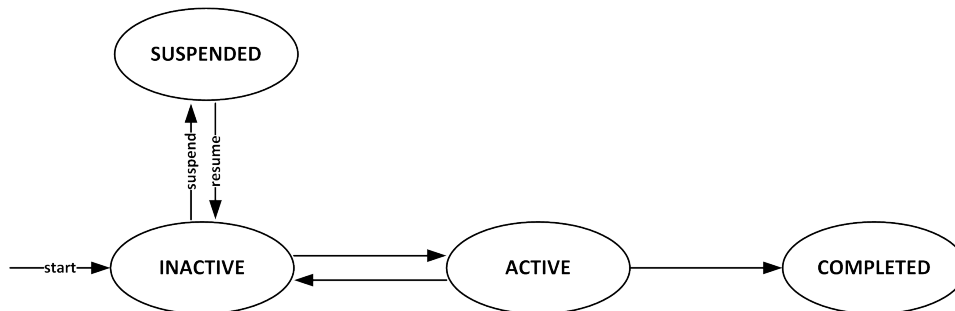


Figure 4.17: Task states

Above we have said that we have four syntaxes with different behaviors in the parameters management. This means that four versions of the engine are required. Since we are more interested in the notation evaluation, we will not implement the engine.

Chapter 5

Implementation

5.1 The Web Site

We have realized the web site in PHP, HTML and CSS and we have made large use of JavaScript and jQuery to increase the interaction and the responsivity of the system.

The site has a really simple structure. It is divided into three main sections: Processes, Contacts and Profile. To access them, a user must first authenticate himself. The authentication is done with the OAuth mechanism: users can so choose among some of the most used social networks. In our case, we have implemented login systems for Google+, Facebook, Twitter and LinkedIn.

Once the user has been authenticated for the first time, the application creates a new database entry in the users table with his name and avatar. The respective social ID is also saved, while the OAuth token is temporarily stored in session.

Figure 5.1 shows the database schema to manage the registered users, their contacts and the created processes. We decided to store social details for both users and contacts in the table Social. It contains the ID of the account in the social network, the name of the person and the avatar. Facebook and Twitter make use of numeric IDs, while LinkedIn and Google+ use strings. For this reason in Social there are social_id and social_token. The first one is used for integer IDs and the second one for string IDs.

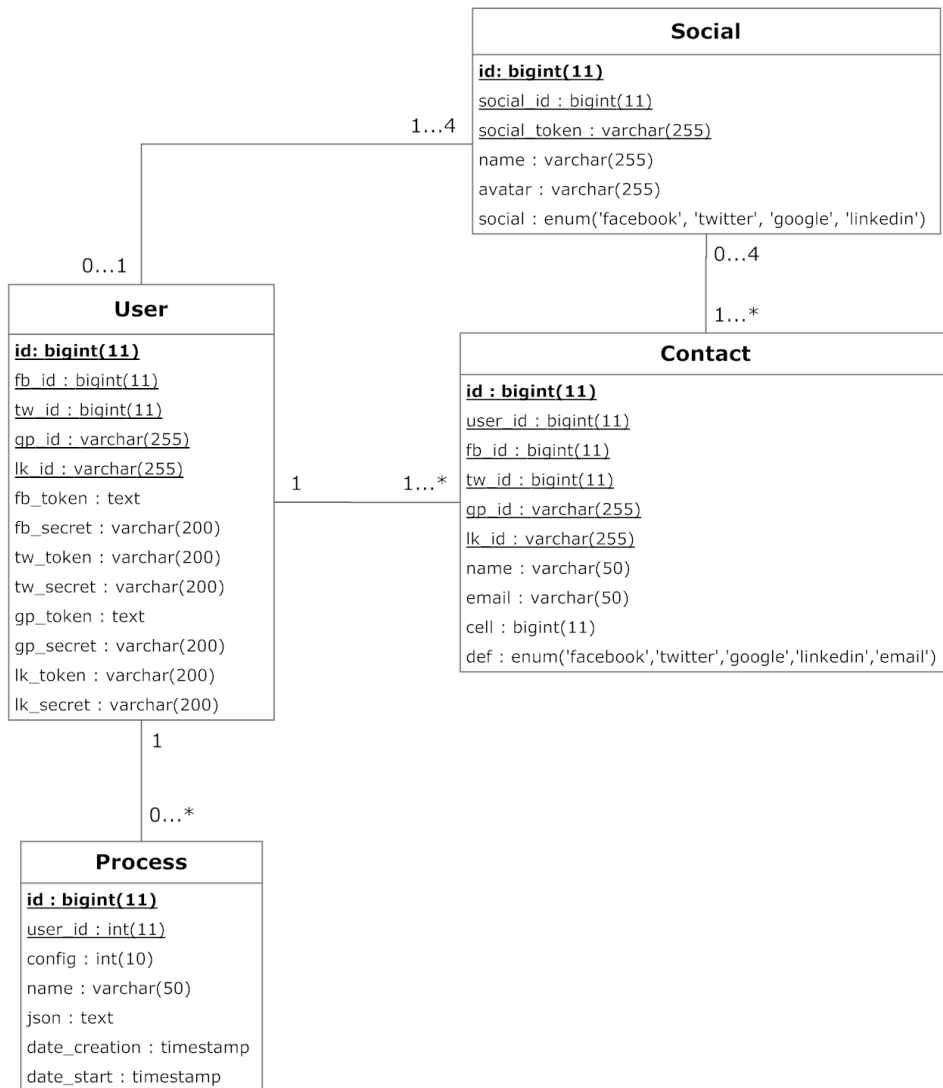


Figure 5.1: ER diagram

5.1.1 Profile

In this page the user can join other social accounts with the one he has used for the first authentication (Figure 5.2). He can also see a list of all his joined accounts. For each one of them, the username, the user's avatar and the related social network icon are shown. An account can also be disjoined and consequently removed by clicking on the cross button. This action will remove also contacts imported with that account.

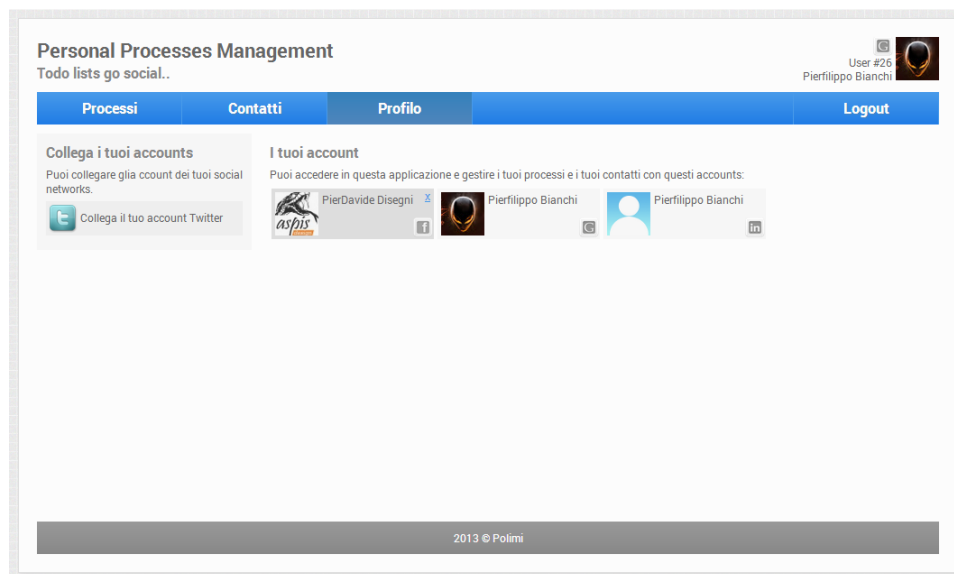


Figure 5.2: Profile page

On the left side of the page users can choose more social networks to join. When he clicks on one of them, an OAuth process is started through the APIs of that social in order to authenticate him. If the authentication successfully ends, his name, his avatar and his user ID are added to his account, and the token is saved in session. From now on, the user will be able to import and use contacts from the contacts' list of that social.

5.1.2 Contacts

In the Contacts page users can import contacts from their socials' friends' lists (Figure 5.3). On the left side of the page there are buttons to require the import. When the user clicks on one of them, an AJAX request is sent to the chosen social through its specific APIs, asking for a complete friend

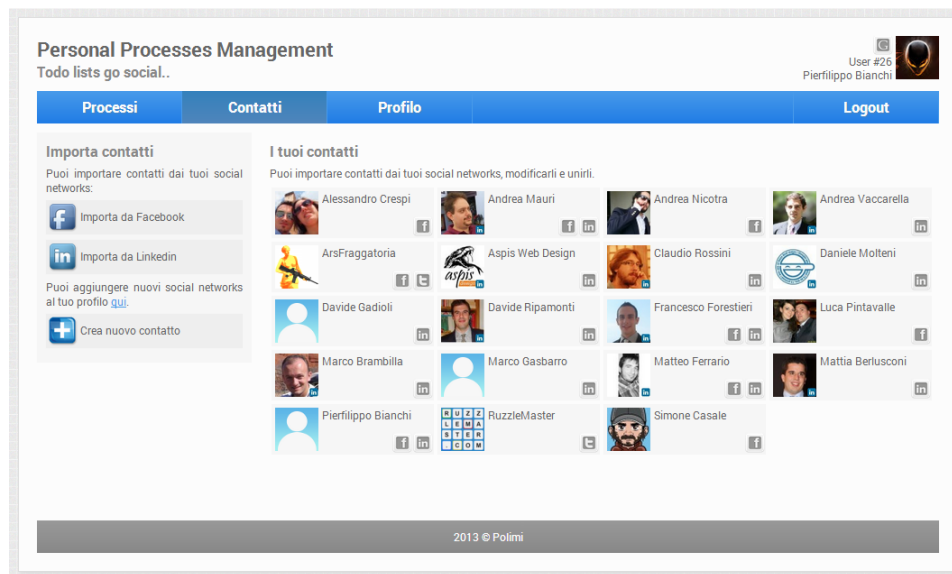


Figure 5.3: Contacts page

list. When the list is received, the user can select which contacts to import (Figure 5.4).

Due to Google+ APIs restrictions it was not possible to import contacts from Google+ circles. Imported contacts are listed in the rest of the page. Users can also create new contacts from scratch writing name and email, or modify the existing ones. We also have implemented a mechanism for contacts merging that allows merging data of two different contacts in the list (Figure 5.5). Therefore if the user is connected to a friend through different socials, he can put his friend's socials all together in only one contact.

Some icons show which social networks are available for each contact. Users can also express a preference for each contact, setting which social to use as default.

5.1.3 Processes

In the processes page users can create and manage their own processes (Figure 5.6). Created processes are listed in a table showing creation dates and their status. Processes can be deleted and opened in the editor to have a detailed view or to modify them.

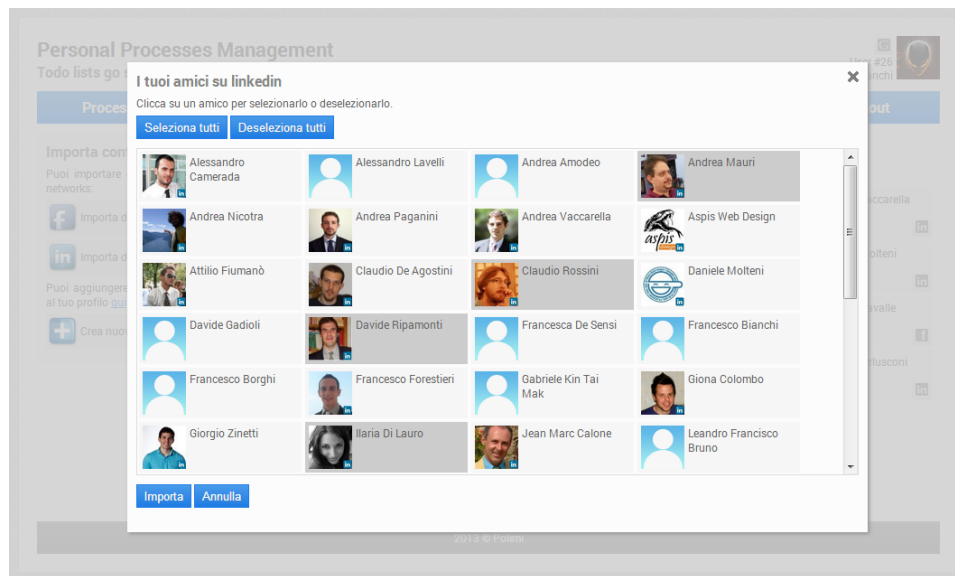


Figure 5.4: Social network's friends import panel

5.1.4 Social APIs

We have used social networks integration to manage the application login and contacts lists. Each social has its API to make requests and exchange resources. First of all we have registered our application to the chosen social networks obtaining the application keys and secrets which permits to execute valid calls.

Although the interaction mechanism is similar, every social network differs in the way the requests are built. In order to make things easier, there are libraries and official developer's kits of functions to use. Usually the first operation to do is the instantiation of the object that will manage the calls, setting the proper application key and secret.

For example in Facebook it is done in this way:

```
$facebook = new Facebook ( array (
    'appId' => FB_CONSUMER_KEY,
    'secret' => FB_CONSUMER_SECRET,
) );

$facebook->setAccessToken( $user->fb_token );
```

while in Twitter:

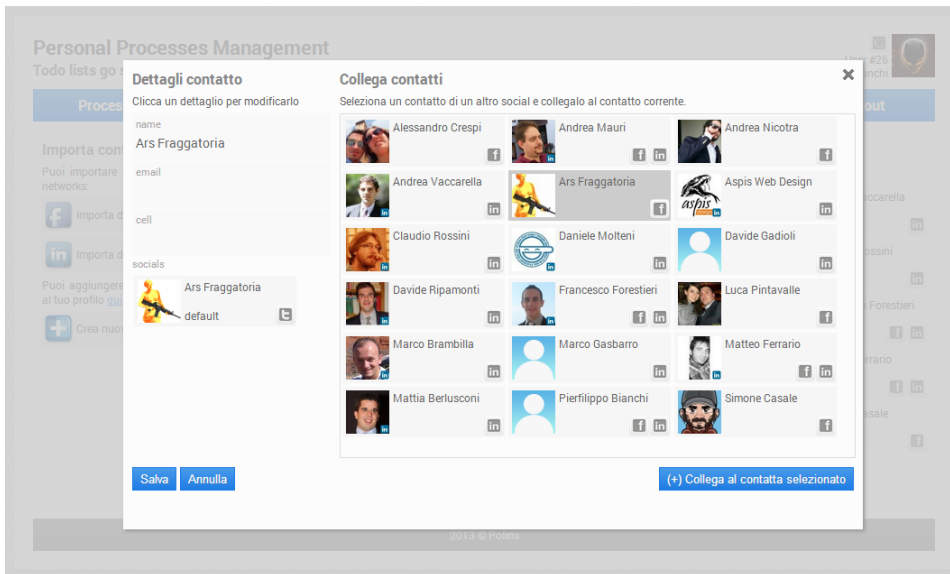


Figure 5.5: Friend's details and contacts merging

```
$twitter = new TwitterOAuth( TW_CONSUMER_KEY, TW_CONSUMER_SECRET,
$user->tw_token, $user->tw_secret );
```

The API interaction consists in a HTTP GET or POST request sent to the social network, and is followed by its response, usually received under the form of a JSON file. Usually, when asking for the user friends list, more than one API call is necessary: a counter variable, coded in each JSON file, tells if other requests must be made to obtain the complete list. This is the case of Twitter, which need the use of a loop of calls to correctly collect all the user friends. First of all it is necessary to request the array of friends' ids, and then use it to formulate another request to get the corresponding profiles. Since in Twitter there is the distinction between followers and followings, and not just the concept of friends, we have decided to return data of both of them. The procedure is the following:

```
$followers_ids = $twitter->get( 'followers/ids',
                                array( 'user_id' => $user->tw_id ) );
$friends_ids = $twitter->get( 'friends/ids',
                              array( 'user_id' => $user->tw_id ) );
$contacts = array_merge( $friends_ids->ids, $followers_ids->ids );
$contacts = array_unique( $contacts );
$contacts = array_chunk( $contacts, 100 );
$friends = array();
```

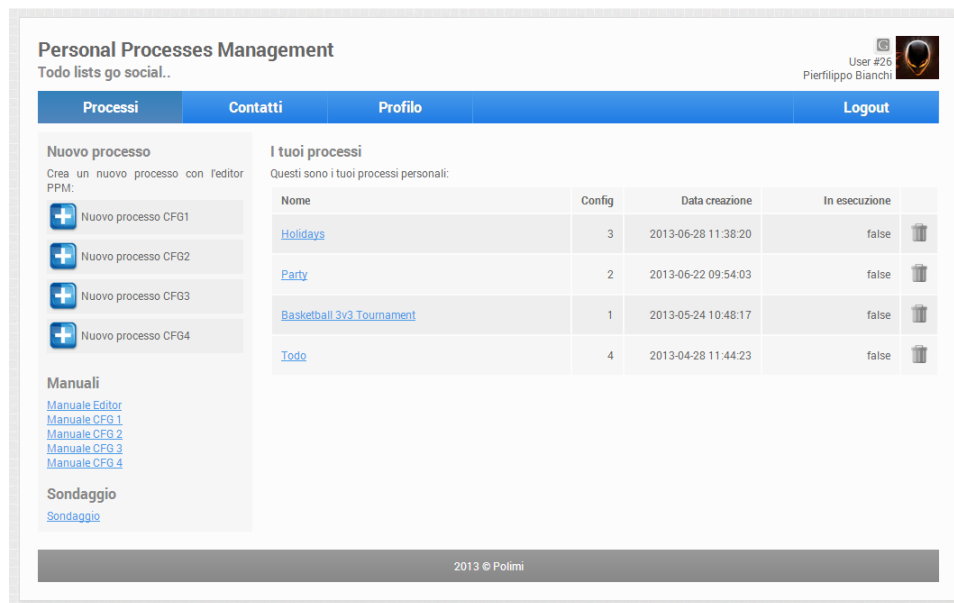


Figure 5.6: Process page

```

foreach ( $contacts as $c ) {

    $contacts_str = implode( ',', $c );

    do {
        $friends_lookup = $twitter->post( 'users/lookup',
                                           array( 'user_id' => $contacts_str ) );
    } while( count( $friends_lookup ) == 0 );

    foreach ( $friends_lookup as $r ) {
        array_push( $friends, array( $r->id, $r->screen_name,
                                     $r->profile_image_url ) );
    }
}

```

Facebook appears to be more immediate and allows making requests using a query language:

```

$params = array(
    'method' => 'fql.query',
    'query' => "SELECT uid, name, pic_square FROM user WHERE uid IN
               (SELECT uid2 FROM friend WHERE uid1 = me())"
);
$result = $facebook->api( $params );

```

5.2 Graphical Editor

We have chosen to develop the graphical editor in JavaScript using the canvas potentialities. The canvas is an HTML5 element that allows to draw graphics on web. It is only a container and it requires scripts to actually draw the graphics, so JavaScript represents the obvious choice.

Since the functions to create figures are of low-level and they require a lot of code lines, even for the easiest things, in order to better manage the creation, deletion, customization and movement of the elements, we have relied on a library. In particular we have chosen to use the Draw2D library [7], which is based in its turn on Raphael [8], another JavaScript library, and simplifies the drawing and the interaction. It in fact supports the drag and drop of the elements with mouse and all the other events like click, double click and keypress.

We have developed the editor in a popup window, separated from the website, in order to remove the browser's menu and status bars and maximize the space. We have organized the layout using a jQuery plugin so that it becomes more responsive on window resizing. The editor is divided in two parts: a toolbar on top and the canvas in the middle (Figure 5.7). The toolbar shows the process name, which can be edited, and the buttons to clear the canvas, save the process, validate the diagram, delete a selected element and zooming.

5.2.1 Classes

A peculiarity of Draw2D is that it simulates an object oriented coding style and its strength resides in the possibility to inherit and to extend classes. Draw2D makes available some basic elements that we extended for our purposes.

Figure 5.8 shows the class diagram which explain the class hierarchies.

The most important object is the VectorFigure class. It is the base class for all vector based figures inside the canvas. It inherits from the Node class which is instead the base class for all figures which can have ports. Ports are the anchors for connection lines.

Some basic geometric shapes are also available, that inherit from VectorFigure. Since we needed three kinds of shapes, the rectangle, the circle and the diamond, we extended those classes.

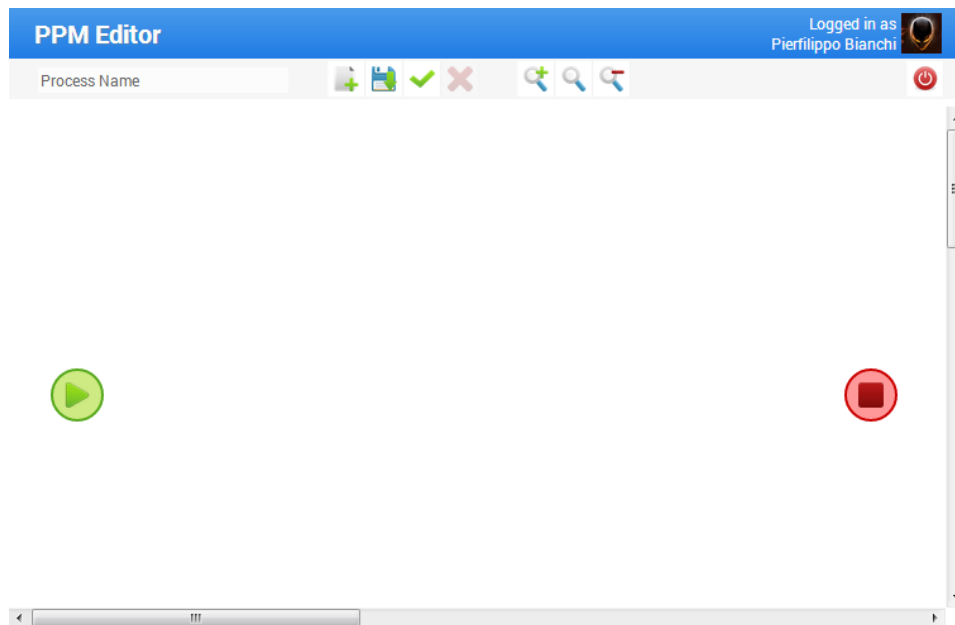


Figure 5.7: Graphical editor interface

We have extended the `Rectangle` class to create the task element and the parallel opening and closure gateways. While the task has fixed dimensions, the parallel gateways have variable height depending of the position of the connected elements. We have extended the `Diamond` class to create the loop and the conditional opening and closure gateway. These three gateways have the same exact shape: a loop can be seen as a modified version of the conditional gateway. We have extended the `Circle` class to create the start, the end and the intermediate events. This last one was specialized in the two types of time event: wait for and wait till. These four events differ in size, colors and in the icons we loaded inside them.

The `VectorFigure` class has been enriched with the functions useful for each elements like the `onDrag()` method which is called whenever an element is been moving in the canvas.

5.2.2 Ports and connections

Tasks elements and intermediate events present two ports, one for the input and one for the output connections. The start event instead has only the output port while the end event has only the input port. The input port is always positioned on the middle of the left side of the element while

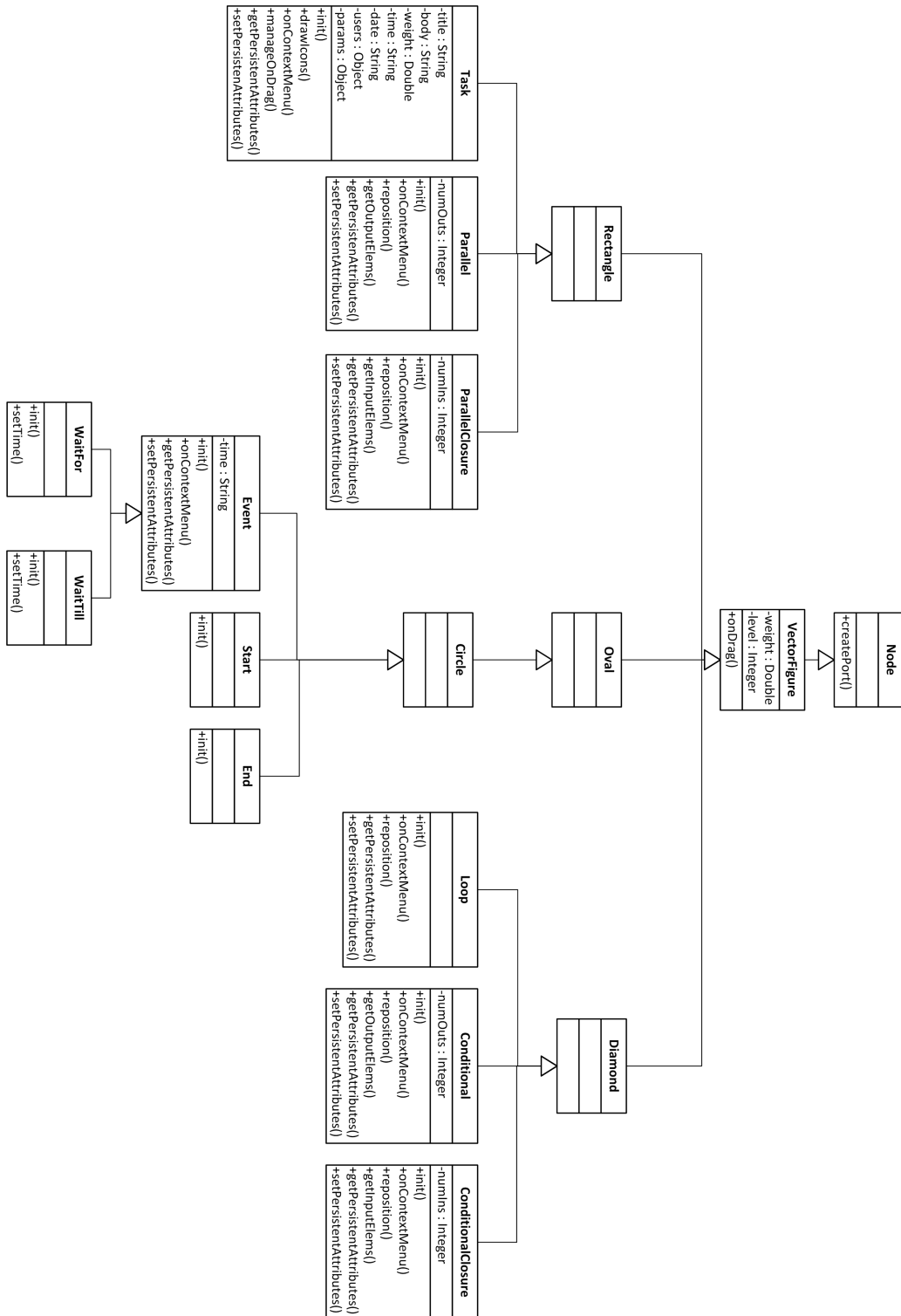


Figure 5.8: Editor class diagram

the output port on the middle of the right side. When a connection has been created, the source and target ports became invisible, because these elements can have only one connection per port Figure 5.9.



Figure 5.9: Connection creation

For gateways the situation is a little different. In fact, they must handle many connections. In order to facilitate the drag and drop to create connections from and to the gateway, we have introduced a particular type of port always fixed at the center of the gateway, on left side for incoming connections, on right side for outgoing connections. This port is always visible and available, since once a connection has been created, the link is drawn between a new hidden port created on the gateway and the connected element Figure 5.10.

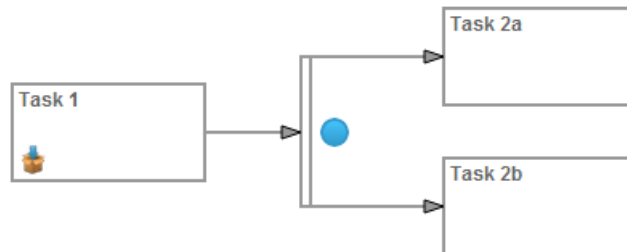


Figure 5.10: Parallel gateway

5.2.3 Gateways positioning

We have imposed some movement restrictions in order to keep the diagram more orderly, to avoid overlaps and to better shift gateways. Elements connected to gateways openings cannot be moved toward the right over a certain limit. The same applies for elements connected on the right of gateways closures: they cannot be shifted too much to the left. Gateways can be moved only in the space between the previous and next connected elements. In this way the block structure of the diagram is kept legible. If more space is needed in the canvas, groups of elements can be selected together and moved in a new position.

5.2.4 Elements management

We have managed the creation of elements in two ways. Right clicking on the canvas shows up the context menu to create a new task or an intermediate event (Figure 5.11). The start and end events are already present in the canvas, since they cannot be duplicated or deleted.

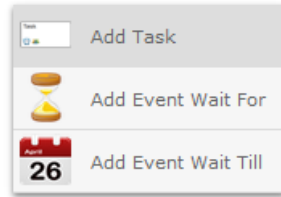


Figure 5.11: Context menu for the creation of task and events

Gateways instead are created right clicking on tasks' and events' output ports (Figure 5.12). We have adopted this solution for two reasons. First of all it allows us a better control on the validation of the modeling: when the user clicks with the right button on an output port, a menu shows up listing both the available gateways openings and specially the available closures. In this way the user is guided and is less likely to make mistakes. The second reason is that gateways cannot be detached from elements: they must have at least an input or an output connection. This is done again to help the user keeping the control on the diagram and to reduce errors.

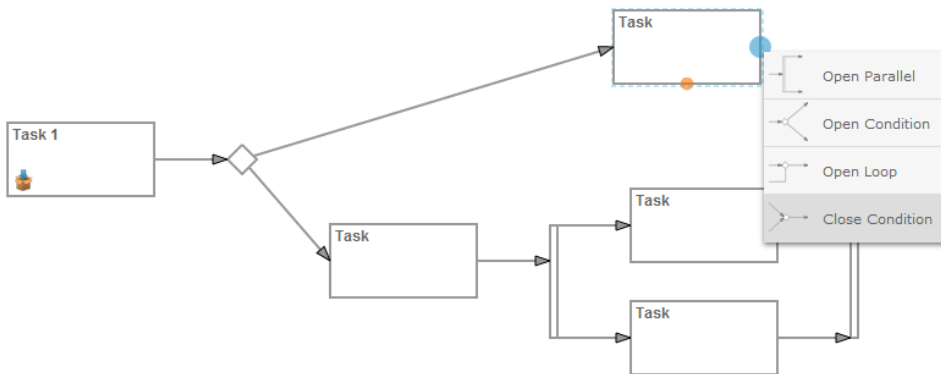
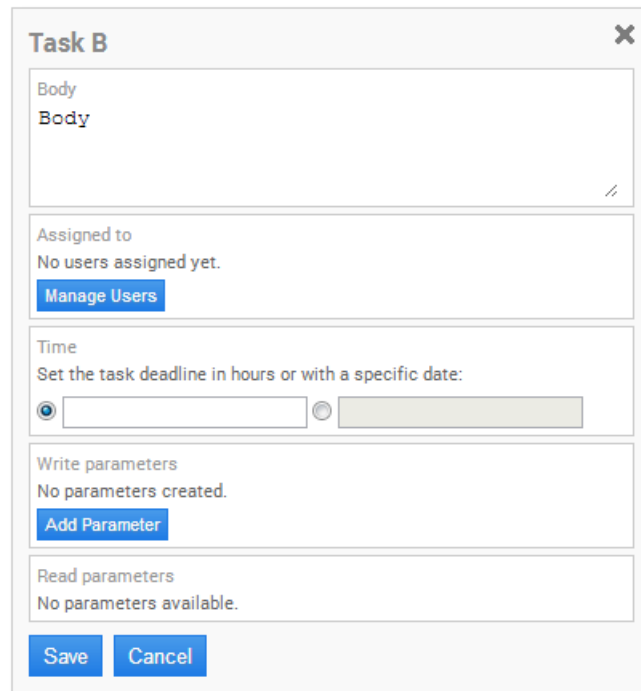


Figure 5.12: Context menu for the creation of gateways

All elements, except for start and end events, can be deleted from their context menus. Gateways are automatically deleted when their input connection or their previous element are deleted.

Tasks, intermediate events and conditional branches context menus allow also editing the element. We have built a draggable panel which appears on foreground listing all the information.



The image shows a window titled "Task B" with a close button (X) in the top right corner. The window is divided into several sections:

- Body:** A text area containing the word "Body".
- Assigned to:** A section with the text "No users assigned yet." and a blue button labeled "Manage Users".
- Time:** A section with the text "Set the task deadline in hours or with a specific date:". Below this text are two radio buttons. The first radio button is selected and is followed by a text input field. The second radio button is followed by a date input field.
- Write parameters:** A section with the text "No parameters created." and a blue button labeled "Add Parameter".
- Read parameters:** A section with the text "No parameters available."
- Buttons:** At the bottom of the window are two blue buttons labeled "Save" and "Cancel".

Figure 5.13: Task editing panel

Figure 5.13 shows the task editing panel. The admin can set the task name, the body message used for the notification and the execution time deadline. Clicking on the “Manage Users” button, a window with admin’s friends is loaded, and from it he can choose one or more actors to assign to the task. Depending on the current parameters management, the admin can also create, accept and propagate parameters or print them in the body message.

Intermediate events editing panels simply allow the admin to set the waiting time intervals.

Clicking on a conditional branch, it is possible to express conditions on parameters. As shown in Figure 5.14, the admin must first select an available parameter, then, depending on its type, he can choose a logical operator and insert a value.

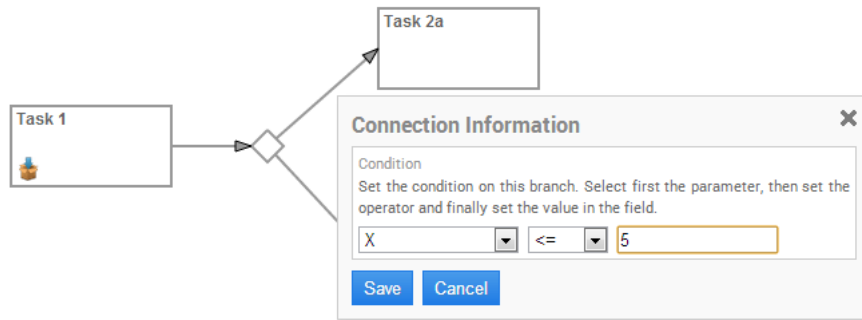


Figure 5.14: Conditional branch editing panel

5.2.5 Control Algorithms

We noticed that it would have been difficult for a novice user to create an entirely correct and working process. For this reason we decided that the introduction of control systems was essential. We have thus implemented a validation system that can be used by users every time they want, while modeling a process, to check its status. We have implemented also real-time controls on creation of connections between elements. The algorithms we have built for these controls make use of two key concepts we have devised. The first one is the block structure described in chapter 4, based on elements levels. The second one is the notion of element weight. We will now describe how the weights are generated and how we use them. We will then describe the details of each control algorithm.

Elements Weights

The idea is to give a weight to each element of the graph. Each weight value depends on the weights of the previous elements. The only element having a predefined value is the Start, with weight equal to 1. The algorithm we wrote explores the graph starting from the Start and computing weights one by one according to the way the elements are connected. The rule is that if an element B comes after an element A and they are in sequence, B gains the weight of A. Otherwise if they are connected by a gateway opening (parallel or condition), B gains the weight of A divided by the number of the gateway output connections. It follows that elements in nested blocks have different weights or, in other words, the higher the level, the lower the weight.

If we state that the execution of a process must start only from the Start and that the End Event is the only element that can symbolize the end of a process, it is clear that a process can be considered complete if and only if the weight of the End is one.

Graph Traversal

Weights computation, connection controls and process validation require a lot of graph exploring. In order to simplify our algorithms we have created a set of basic functions for the execution of recurrent operations. Here follows the list of these functions with a short description for each one:

- *getApertura* : *function(elem, level)*: given an element and its level, the function returns, if it exists, the Opening Gateway that delimits the block in which the element is contained. A gateway is considered belonging to the block which contains it, and not the block delimited by that gateway.
- *getChiusura* : *function(elem, level)*: given an element and its level, the function returns, if it exists, the Closing Gateway delimiting the block in which the element is contained.
- *getGateway* : *function(closure, level)*: given a Closing Gateway, the function returns the corresponding Opening. It assumes that all the incoming branches start from the same Opening. This is possible thanks to constraints and controls that avoid cross-block connections and Opening-Closing decoupling.
- *getChiusuraGateway* : *function(opening, level)*: given an Opening Gateway, the function returns the corresponding Closing.
- *getStart* : *function(canvas)*: this function returns the Start element object.
- *getEnd* : *function(canvas)*: this function returns the End element object.
- *getRoot* : *function(elem, canvas)*: given an element, this function returns the root element of the graph in which the element is contained. Note that creation of subgraphs is allowed, although they must be connected together to form a correct process.

- *sonoInLoop* : *function(source, target)*: given potential source and target elements of a connection, it checks if they are in loop. In other words, if the target element is currently preceding the source inside the graph, the function returns true.
- *sonoConnessi* : *function(source, target)*: it checks if potential source and target elements of a connection are already connected.

Connection Control

Each time a user tries to create a connection between two elements, the function *validaConnessione* : *function(source, target, canvas)* is called. This function receives source and target elements and checks if the user is trying to create a valid connection.

First of all, it checks if the source is a backward port of a loop and then if the potential connection is valid. Rules for a valid loop are:

- The source element of a loop must be reachable from the target.
- Tasks are the only type of element allowed as target for a loop.
- The source and the target of a loop must be of the same level.
- The source and the target of a loop must reside in the same block (*getApertura* : *function(elem, level)* returns two different Gateways for the source and the target).

If the source of the connection is not a backward port it means that the user is trying to connect two elements in sequence. In this case the rules for a valid connection are:

- The source cannot have an already outgoing connection and the target cannot have an already incoming connection.
- If the target is the End, the source must have level equal to zero and it must not be a Gateway Opening. Otherwise this would cause End to have weight less than one.
- The connection must not make a Closing Gateway the closing of an Opening Gateway different from its corresponding opening.

- If the source has level equal to zero, the target cannot be a Closing Gateway.
- The source must not be reachable by the target.
- The connection cannot directly connect the Start and the End.

If the user tries to connect two elements breaking one of the previous rules, the application detects the error and prevents the connection from being created. The user is warned by an alert. On the contrary, if the connection is valid an arrow from the source to the target element is drawn and elements levels are updated by calling the function *ricalcolaLivelli : function (elem, canvas, isFirst)*. Given an element, this function updates levels starting from the root element of the subgraph in which the element is contained. In fact a connection can alterate only element's levels of its containing subgraph.

Validation Algorithm

The validation algorithm checks if the process is rules compliant. We have said that we have imposed many constraints from a syntactical point of view, in order to address the user to a good solution and to prevent excessively faulty processes.

The validation algorithm scans the entire process searching for a pre-defined set of errors. Every aspect making the process incomplete or not restrictions compliant is considered an error. When invoked, the algorithm starts analyzing the diagram element by element. It searches for any element not reachable from the Start. It scans gateways to ensure that each Closing has at least two incoming connections and that each Opening has at least two outgoing connections. Tasks and events are scanned to ensure that their mandatory properties have been set. At this point the algorithm calls *azzeraPesi : function(canvas)* and *ricalcolaPesi : function(elem, nCiclo, nTot)* and element's weights are updated. *azzeraPesi()* is required because of the structure of the recursive function *ricalcolaPesi()*, which assumes all elements having weight equal to zero. *nCiclo* and *nTot* parameters are used in order to avoid redundancy in the exploration of the graph. Otherwise redundancy could occur with gateways. Once the weights have been updated, the algorithm checks the level and the weight of the End. As we have already said, if the End has weight less than one, it means that the flow is not

entirely routed in the End and the process is either invalid or incomplete. Furthermore, if the End has level greater than one it means that the user was able to end the process without closing all the opened blocks.

Load and Save

Processes are stored in the database under the form of JSON strings. Elements' properties can be divided into two categories. Some of them are variable and they change from an element to another, some other are equal to all the elements of the same type. Only the variable attributes are saved of course. Each element class has its own method to extract those data, *getPersistentData()*. The save method collects all the persistent data of all the elements one by one, and then it converts them into JSON. Lastly a PHP action assigned to JSON storing into the database is called.

Some of the attributes to save are different depending on the element class. Name of the class of course, element ID, coordinates, size and level are common to all the classes. For tasks are saved also the weight and all the attributes set by the user like title, body, time constraints, actors and parameters. Also for events the weight is saved while gateways have the number of output ports in the case of Openings or the number of input ports in the case of Closings. A special case is the connection that only needs a source ID and a target ID.

5.3 The Logging System

In order to evaluate the four syntaxes, we decided to implement a logging system able to collect more data as possible during the usage of the application. Logging was limited to the graphical editor. The logger was made up of a PHP action, responsible for logged actions storage into the database, and of a JavaScript file, responsible for data collection from each log request and used for building an AJAX request to be sent to the PHP action.

First of all the JavaScript logger must be instantiated:

```
logger = new Logger( userID, syntax, processID );
```

The parameters accepted by the constructor are the user ID, the syntax number, and the process ID. Once the logger has been initialized, a log request can be done calling the function "log":

```
logger.log( 'create', 'task', task.getId() );
```

This function makes an AJAX request and the PHP action inserts new data in the database. Each log row contains a code name for the tracked action, a field used for additional info, and the target of the action. In Table 5.1 are listed all the actions we decided to record with the respective names and codes used in log rows. Error codes and invalid connections codes are explained in Table 6.3 and Table 6.4

Being logs stored in a database, desired data can be easily extracted by means of SQL queries. We prepared a set of basic queries capable for the extraction of:

- time required to create each process
- number of times an element has been edited
- number of elements created
- number of times an element has been moved
- number of delete
- number of process savings
- number of validation requests
- number of invalid connections

Starting from these queries then we could easily build composite queries extracting derived data useful for comparison and evaluation of the syn-taxes.

Description	Coding
The user starts the creation of a new process by opening the editor.	<i>Name:</i> start <i>Info:</i> NA <i>Target:</i> NA
The user creates a new element in the canvas.	<i>Name:</i> create <i>Info:</i> task, wait_for, wait_till, and_open, or_open, and_close, or_close, connection, loop <i>Target:</i> elementID
The user tries to create a not allowed connection.	<i>Name:</i> invalid_connection <i>Info:</i> invalid connection error code <i>Target:</i> NA
The user deletes an element of the canvas.	<i>Name:</i> delete <i>Info:</i> task, wait_for, wait_till, and_open, or_open, and_close, or_close, connection, loop <i>Target:</i> elementID
The user opens the info panel of an element.	<i>Name:</i> open_info <i>Info:</i> task, wait_for, wait_till, connection <i>Target:</i> elementID
The user edits an attribute of a task.	<i>Name:</i> edit_task <i>Info:</i> title, body, actors, deadline, param_add, param_remove, param_rename, param_prop, param_recv, param_unprop, param_unrecv, param_type, param_print <i>Target:</i> taskID
The user edits the time of an event.	<i>Name:</i> edit_event <i>Info:</i> time_waittill, time_waitfor <i>Target:</i> eventID
The user edits a condition.	<i>Code name:</i> edit_connection <i>Info:</i> cond_add, cond_remove, param, operator, value <i>Target:</i> connectionID
The user closes an info panel of an element.	<i>Name:</i> close_info <i>Info:</i> close, cancel, save <i>Target:</i> elementID
The user moves an element in the canvas.	<i>Name:</i> move <i>Info:</i> task, wait_for, wait_till, and_open, or_open, and_close, or_close, loop, start, end <i>Target:</i> elementID
The user clicks on the “Validate” button asking for a validation of the created process.	<i>Name:</i> validate <i>Info:</i> validation error code (if invalid) <i>Result:</i> True / False
The user clicks on the “Save” button saving the process.	<i>Name:</i> save <i>Info:</i> NA <i>Target:</i> NA
The user clicks on the “Shut Down” button to conclude the experiment.	<i>Name:</i> end <i>Info:</i> NA <i>Target:</i> NA

Table 5.1: Logged actions

Chapter 6

Experiments

6.1 Preparation of the experiments

In Chapter 4 we have introduced the graphical notation and we have identified four different syntaxes, each one consisting of a combination of some of the notation elements.

The idea was to submit these syntaxes to real users and to conduct experiments in which they used one of these syntaxes to model a process inside the graphic editor we have developed. By observing how users model the processes, and then analyzing their outcomes, we expect to determine which is the best configuration.

To do this, we have invented some simple scenarios based on situations that may happen to face in real life, being this the goal we want to achieve. Every scenario describes a process with suggested activities and some general constraints. We have asked people to model the process, imagining that they need to coordinate a group of friends and colleagues, assigning persons to tasks.

These are the three scenarios used during the experiments along with the test scenario used to explain the editor functionality:

Scenario A - Holiday with friends

You are organizing a holiday with your friends in the first week of August. You have decided to visit London and set the budget to €1200 per person. First of all you must decide the transport, choose the hotel and plan some possible tours. Then you have to draw up the final budget and if it is

included in the planned costs, you can proceed with reservations. The week before the departure, you will meet together to define the last details.

Scenario B - Association party

You are a member of an association which wants to organize a party on July 1st with an outdoor dinner and music to celebrate its 10 years of activity. Participation at the event is free for anyone but it is necessary to take a reservation. First of all you need to publicize the event, and once at least 50 reservations are taken, you have to inform the municipality and get the authorization. Then you can contact the catering service and engage the band. The event will start with the dinner at 8:00 pm followed by the concert at 9:30 pm if the dinner has finished.

Scenario C - Warehouse management

Your company works with a lot of raw materials stocked in the warehouse. Your goal is to optimize the buying and selling of the materials to avoid wasting it, maintaining always 1000 units constant. Every week, the total quantity of material stocked in warehouse must be checked and then, according to the amount, you need to buy new raw material or sell it. Each order must be first approved by the accounting department. Once arrived, new stock must be placed and catalogued.

Scenario Test - Basket tournament

You are organizing a basket tournament. You need to collect the registrations, schedule the matches, rent the playgrounds and find the sponsors.

In every scenario we have asked users to model the process trying to reduce organization times and involving as many persons as possible.

Having four syntaxes and three scenarios, overall there are twelve possible couples, called use cases, shown in Table 6.1.

We have decided to submit two use cases to each user. It should be noticed that the submission order of the two use cases is relevant because, trying to solve the second test, the user will apply the knowledge gained during the first one.

use case	syntax	scenario
1	1	A
2	2	A
3	3	A
4	4	A
5	1	B
6	2	B
7	3	B
8	4	B
9	1	C
10	2	C
11	3	C
12	4	C

Table 6.1: Use cases

In order to overcome unintended effects and to have balanced experiments, we have studied all the possible use cases pairs taking in consideration both the order and the scenario-syntax coupling.

Applying the Graeco-Latin square theory to the use cases, we obtained the result shown in Figure 6.1.

Each cell of the table represents a single experiment, made up by two use cases to be assigned to a single user. In each row of the table, every syntax and every scenario appears the same number of times, and every use case appears one time in the first position and one time in second position.

Invalid experiments are highlighted in red. We consider invalid an experiment which has the same scenario and/or the same syntax in both tests. For example, the experiment [2,3] is invalid because of the repetition of scenario A, while the experiment [5,9] is invalid because of the repetition of the first syntax.

Excluding the red cells, the remaining green cells can be grouped as shown in Figure 6.2.

The second and the fifth rows have six cells each that are mirrored repetitions of the other six cells ([3,5] and [5,3], [4-6] and [6,4], etc.). However, it was possible to combine them to obtain two new rows in compliance with

1,1	2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	11,11	12,12
2,3	3,4	4,5	5,6	6,7	7,8	8,9	9,10	10,11	11,12	12,1	1,2
3,5	4,6	5,7	6,8	7,9	8,10	9,11	10,12	11,1	12,2	1,3	2,4
4,7	5,8	6,9	7,10	8,11	9,12	10,1	11,2	12,3	1,4	2,5	3,6
5,9	6,10	7,11	8,12	9,1	10,2	11,3	12,4	1,5	2,6	3,7	4,8
6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
7,1	8,2	9,3	10,4	11,5	12,6	1,7	2,8	3,9	4,10	5,11	6,12
8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2
9,5	10,6	11,7	12,8	1,9	2,10	3,11	4,12	5,1	6,2	7,3	8,4
10,7	11,8	12,9	1,10	2,11	3,12	4,1	5,2	6,3	7,4	8,5	9,6
11,9	12,10	1,11	2,12	3,1	4,2	5,3	6,4	7,5	8,6	9,7	10,8
12,11	1,12	2,1	3,2	4,3	5,4	6,5	7,6	8,7	9,8	10,9	11,10

Figure 6.1: Graeco-Latin square of use case couples

the properties we have described above.

Let's call the remaining rows A, A', B, B', C and C' (Figure 6.3). We can see that A' has the same couplings of A but with use cases in the inverse order. The same is for B'-B and for C'-C. In conclusion, we can say that taking in consideration multiples of twelve experiments following the previous technique, we have all the scenarios and syntaxes configurations balanced. This means that in our set of experiments we will have an equal number of times each scenario and each syntax, as well as each use case will appear in first or second position a number of times equal to the other use cases, and consequently this will be valid also for scenarios and syntaxes.

According to these considerations, good sets for the experiment are represented in Table 6.2

1	10,7	11,8	4,5	1,10	2,11	3,12	8,9	5,2	6,3	7,4	12,1	9,6
2	3,5	4,6	1,11	2,12	7,9	8,10	5,3	6,4	11,1	12,2	9,7	10,8
3	4,7	1,12	6,9	7,10	8,11	5,4	10,1	11,2	12,3	9,8	2,5	3,6
4	6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
5	7,1	8,2	9,3	10,4	11,5	12,6	1,7	2,8	3,9	4,10	5,11	6,12
6	8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2

Figure 6.2: Valid experiments

A	10,7	11,8	4,5	1,10	2,11	3,12	8,9	5,2	6,3	7,4	12,1	9,6
A'	4,7	1,12	6,9	7,10	8,11	5,4	10,1	11,2	12,3	9,8	2,5	3,6
B	6,11	7,12	8,1	9,2	10,3	11,4	12,5	1,6	2,7	3,8	4,9	5,10
B'	8,3	9,4	10,5	11,6	12,7	1,8	2,9	3,10	4,11	5,12	6,1	7,2
C	1,7	2,12	3,5	4,10	5,11	6,4	7,9	8,2	9,3	10,8	11,1	12,6
C'	1,11	2,8	3,9	4,6	5,3	6,12	7,1	8,10	9,7	10,4	11,5	12,2

Figure 6.3: Valid experiments

Notice that it would be better to take all the mirrored rows of those currently taken, instead of just a part of them. For instance, if we take A, B and C, it is better to choose also A', B' and C' instead of A' only.

However, having found about twenty people available to test, we have decided to choose A and B rows.

6.2 Experiment procedure

Each experiment has been conducted following this exact procedure in five steps:

1. **Introduction.** First of all, the user was informed about the tool and

	A	B	C
A	-	ok	ok
B	ok	-	ok
C	ok	ok	-

Table 6.2: Experiments sets

its purposes. Then he was told that we wanted to test the effectiveness and ease of use of the editor and of the adopted modeling syntax to model small processes, in daily life or in small and medium-sized enterprises.

2. **Registration.** The user logged in to the web site using one of his social accounts and then he imported a maximum of 15-25 contacts from his socials. If it was possible, the user merged his contacts when there were multiple accounts of the same person.
3. **Instruction.** The user read the main manual of the editor which describes how to make the basic actions of creation, modification and deletion of elements. Then, a short test of 3 minutes took place under our supervision in order to take confidence with the basic functionality of the tool. This was done using a test scenario, equal for all the users, written specifically for this phase.
4. **Experiment.** Each user had to model two use cases. In this step the user could read the manual of the first assigned syntax followed by the description of the coupled scenario. At this point he could start modeling the process. The experiment was supervised and the user could read manuals or ask for help while modeling. Once the first process was modeled, the user could proceed with the second use case in the same way described before.
5. **Closure.** At the end of the test, after the user had submitted both models, he was asked to answer a questionnaire about some personal information, his knowledge in the computer science field and about the ease of understanding the scenarios and modeling them with the assigned syntaxes. We asked him also to report the lack of elements or functionality both in the syntax and in the editor.

6.3 Data analysis

During the experiments we have logged all the users' actions, as explained in Chapter 5, and we have taken some notes on processes modeling by direct observation. All these data, together with the opinions and feedback collected from the questionnaire, must be analyzed to understand each syntax's pro and cons, and hopefully, identify a good compromise.

6.3.1 Outliers

We have paid particular attention to timings and especially to the execution time of each experiment. We have found out that two users have taken an amount of time greater than the one acceptable, with respect to the others, for both the tests. In fact, it was greater than the average time plus the double of the standard deviation. So we have decided to exclude their data and to repeat those experiments with two other new people.

Then we have updated the averages and the standard deviations to check if the new data had created new outliers. Fortunately, there were no problems, so we could proceed to analyze them.

6.3.2 Durations

From the analysis of the times of the single experiments (Figure 6.5) we notice that processes modeled with the first syntax are those done faster, with an average duration of 16'27". On the contrary the fourth syntax is the slower, with an average time of 22'02", about 34% slower than the first (5'35" more). The second and the third syntaxes are instead quite similar between them, about 16% and 15% slower than the first. Looking at the standard deviations, the fourth and first syntaxes have higher values. All the delivered processes are correctly validated except one made with the second syntax. It's interesting to notice that 19 times over 24, the second test has been modeled in less time than the first one, with an average of 7'30" less. This is probably due to the fact that during the first test users need more time to take confidence with the editor. In the remaining 5 times, users took an average of 3'20" more. Also the syntaxes play a role in the difference of time between the two tests. In fact, users who play first with the fourth syntax, and then with the first syntax, take on average 12'43" less, while users who first used the first syntax and then used the second

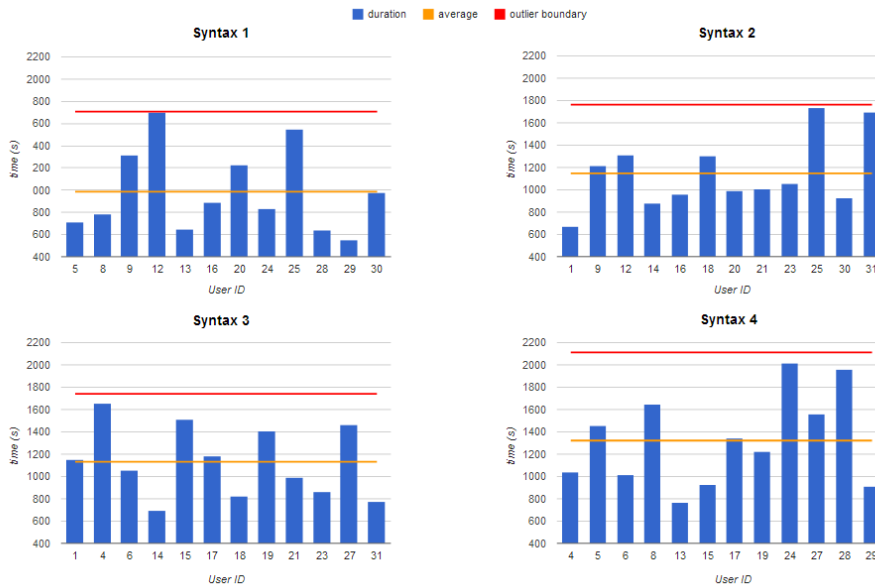


Figure 6.4: Durations per syntax in seconds

syntax, take on average 1'27" less. It seems that the fourth syntax is more powerful than the first but is also heavier to use, so it takes more time. The first syntax is simpler, offers minor possibilities and is faster to use.

6.3.3 Number of elements' creations and deletions

Looking at the number of elements (Figure 6.6) created and deleted per syntax, we notice that the processes created with the first syntax have fewer elements and so they are simpler and smaller. Instead, the fourth syntax has the most number of creations and is the richer one. The second and third syntaxes seem to have a similar quantity of elements used. The number of deletions is quite constant for all the syntaxes, a little less in the first and a little more in the second.

The most used elements are of course tasks and connections. In Figure 6.7 they are omitted in order to better highlight other elements variations. The Wait For events are rarely used and have a high percentage of deletions in all the syntaxes in which they appear, respectively 60%, 80% and 67%, while the Wait Till events are preferred and more used. It is interesting to notice that overall the *Parallel Gateway* is less used than the *Conditional Gateway*. In the first syntax there is only the *Parallel Gateway*,

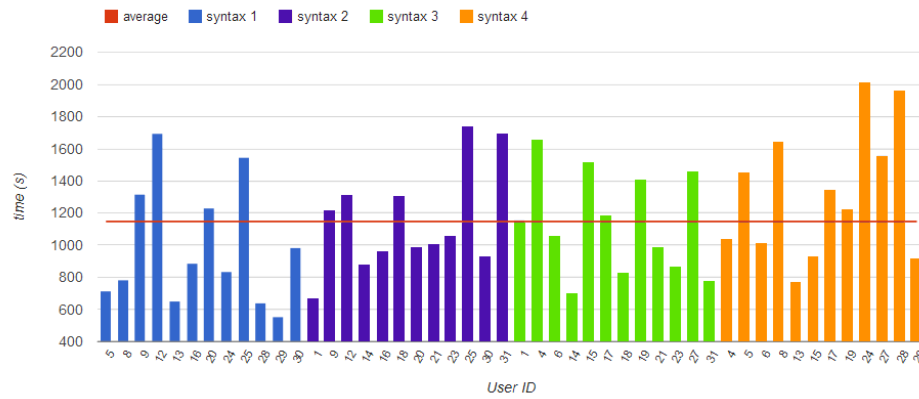


Figure 6.5: Experiments' duration per single user

in the fourth there is only the *Conditional Gateway*, while in the other two they appear both. In the second syntax the *Parallel Gateway* is used twice as often as the *Conditional*. In relation to these data we also need to point out that syntaxes three and four are those with a greater number of created elements overall.

The *Loop*, with an average use of 1.33 per process, is the most used element in the processes where it is available, namely those modeled with syntax two.

6.3.4 Validation requests

During all the tests, users have used the validation button to check their processes correctness. The diagram in Figure 6.8 shows the average number of requests per test done by users for each syntax.

Generally, the processes were correct but sometimes errors were found and users had to revalidate at least another time. Globally, all the syntaxes have almost the same number of validations.

The highest percentage of validations with errors is the one related to the second syntax, immediately followed by syntax three. Syntax one and syntax four are those with the lowest percentages of false validations.

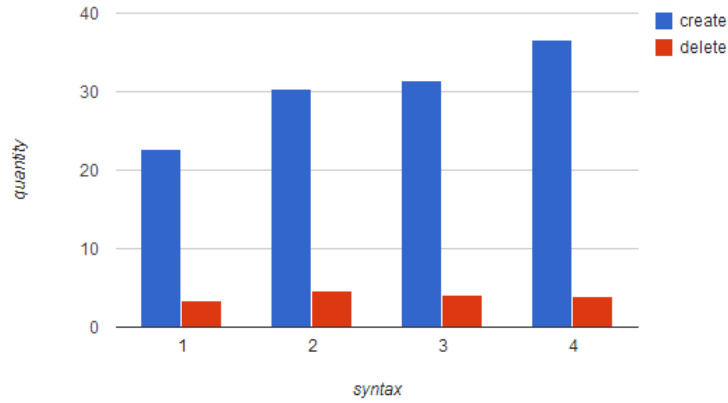


Figure 6.6: Average number of creations and deletions per process

6.3.5 Validation errors

Table 6.3 shows all the errors that could occur during validation. Each error has an identification code used for logging.

Figure 6.10 shows the average number of errors by type per single test. The number of occurrences of an error is divided by the number of all the tests in which it could occur. The second syntax presents the highest number of validation errors, with an average value of 1.33 per process, while the third has 1.08 errors and the first and the fourth only 0.75.

As shown in Figure 6.11, errors per process were really few. The most recurrent error is *invalid04* which occur about one time every three processes. It is followed by both errors *invalid03* and *invalid09* which occur one time every six processes.

Figure 6.12 shows the most relevant errors for each syntax. In the first syntax the most common error is caused by time events, which users generally forgot to configure. Syntaxes two and three are largely affected by error *invalid05*. In syntaxes three and four conditions over gateway connections were often forgotten, but not in syntax two. Another frequent forgetfulness in syntax four was the assignment of user to tasks. A curious fact in syntax two is that a lot of times users didn't connected the start to any task.

error code	description
<i>invalid00</i>	One or more elements are not reachable from the Start
<i>invalid01</i>	One or more Loop Gateways are not used
<i>invalid02</i>	One or more opening gateways do not have the right number of outgoing connections
<i>invalid03</i>	One or more Closing Gateways do not have the right number of incoming connections
<i>invalid04</i>	The End is not zero level
<i>invalid05</i>	The End element is not reachable by all the other elements
<i>invalid06</i>	One or more Task elements are not assigned to anyone
<i>invalid07</i>	One or more Time events do not have the time set
<i>invalid08</i>	One or more Conditional Gateway connections do not have the conditions set
<i>invalid09</i>	One or more Loop Gateway connections do not have the conditions set
<i>invalid10</i>	One or more Loop Gateway do not have the forward connection

Table 6.3: Validation errors codes

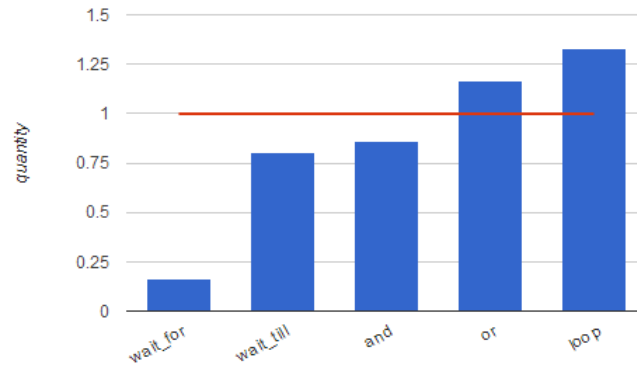


Figure 6.7: Average number of elements per process

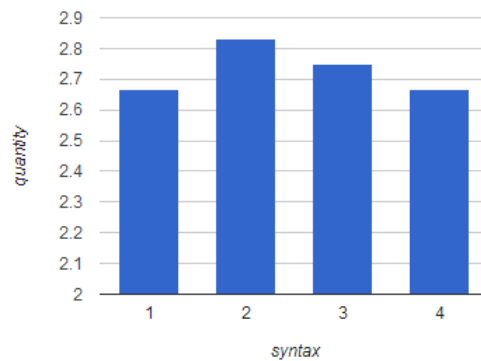


Figure 6.8: Average number of validation requests per single test

6.3.6 Wrong connections

In Table 6.4 are listed all the errors of invalid connections.

In general connection errors were few and there was less than one error per test. As shown in Figure 6.13, the first syntax has no errors at all, while the third and the fourth are those with more wrong connections with respectively eight and nine errors in twelve tests each.

Looking at Figure 6.14 we see that the only two significant errors are *seq01* and *loop02* in about one test over four, users tried to connect non zero level elements to the end, and with the same frequency they connected a *Loop* to elements with different level.

error code	description
<i>seq00</i>	Either the source has already outgoing connections, or the target has already incoming connections
<i>seq01</i>	The target element of the arrow is the End and source is a non zero level element
<i>seq02</i>	The source element of the arrow is an opening gateway while the target is the End
<i>seq03</i>	The arrow directly connects an opening gateway with a closing gateway
<i>seq04</i>	The user is trying to close different openings with a unique closing gateway
<i>seq05</i>	The source element is at level zero and the target is a closing gateway
<i>seq06</i>	The target comes before the source in the execution flow, but the source is not a Loop Gateway
<i>seq07</i>	The arrow directly connects the Start to the End
<i>loop00</i>	The target of the loop does not come before the source in the execution flow
<i>loop01</i>	Target element is not either task or event
<i>loop02</i>	Source and target levels are different
<i>loop03</i>	The source and the target are not in the same block

Table 6.4: Wrong connections codes

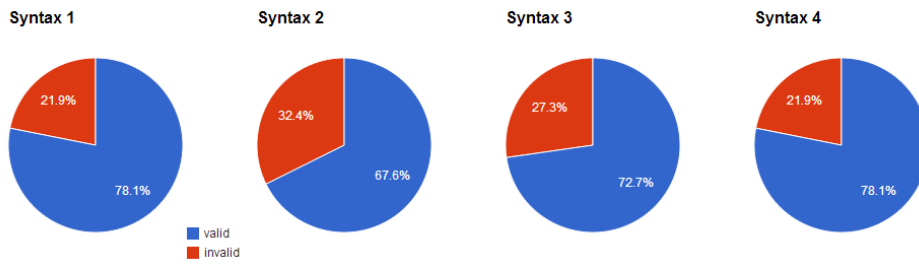


Figure 6.9: Percentage of validations per syntax

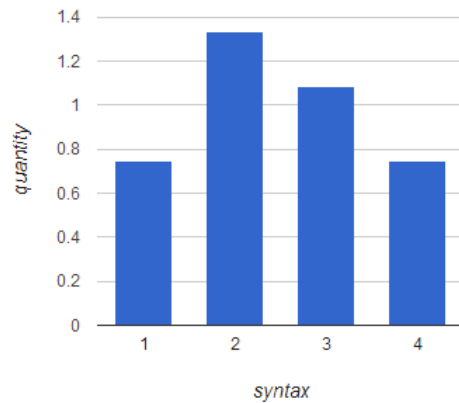


Figure 6.10: Validation errors by syntax per single test

6.3.7 Editing elements

The average time per process spent by users editing elements is about the same for all the syntaxes except for the second one that is significantly smaller.

On average, the time spent modifying single elements is more in the first syntax, though with higher standard deviation, while less time was spent on the fourth. This is not true for the *Wait For* element that, as previously said, is very little used.

On the other hand, the number of element changes is greater in the fourth syntax, while it's smaller for the first one. It's noticeable the high number of changes for syntaxes with local parameters characterized however by short editing times. These characteristics should be caused by the need to manually propagate local parameters. For what concerns events data are

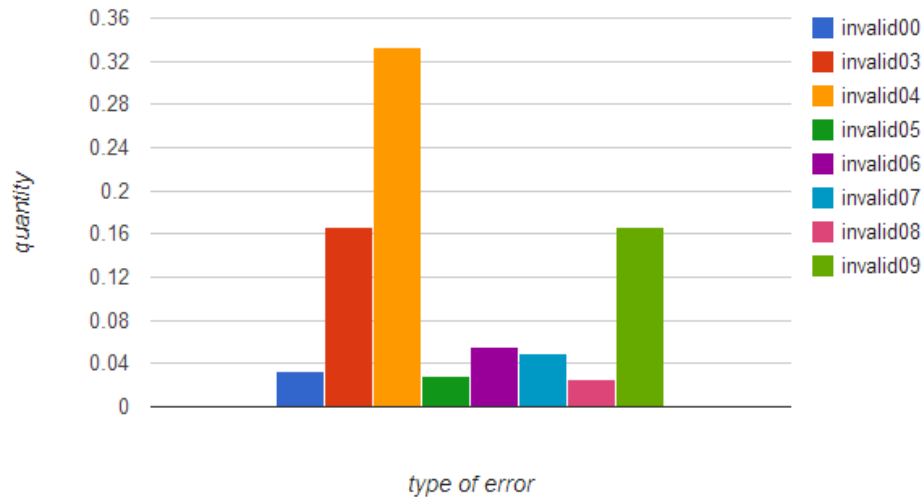


Figure 6.11: Validation errors per single test

quite the same. Conditions on average require more time to be configured with the syntax four.

6.3.8 Questionnaires

Demographic analysis

Overall, twenty-six persons tested the editor and the syntax. Actually, the data we are showing refers to twenty-four users, because two of them were considered outliers and so they were replaced by two other new people. Of the final twenty-four users, twenty-one were men, while only three were females. Nineteen users are in the 18-30 years old range, two users in the 31-50 range and three users in the 51-70 range. Half of the users are students, and seven of them have high skills in computer science. Nine persons define themselves as basic PC users. Concerning BPM knowledge, only two users claim to use it regularly, while eight of them know it without using it and fourteen have never heard of it.

Scenarios difficulty

Each scenario was tested sixteen times. Both scenarios A and B were judged mainly easy with 13 and 12 votes each and the remaining votes on

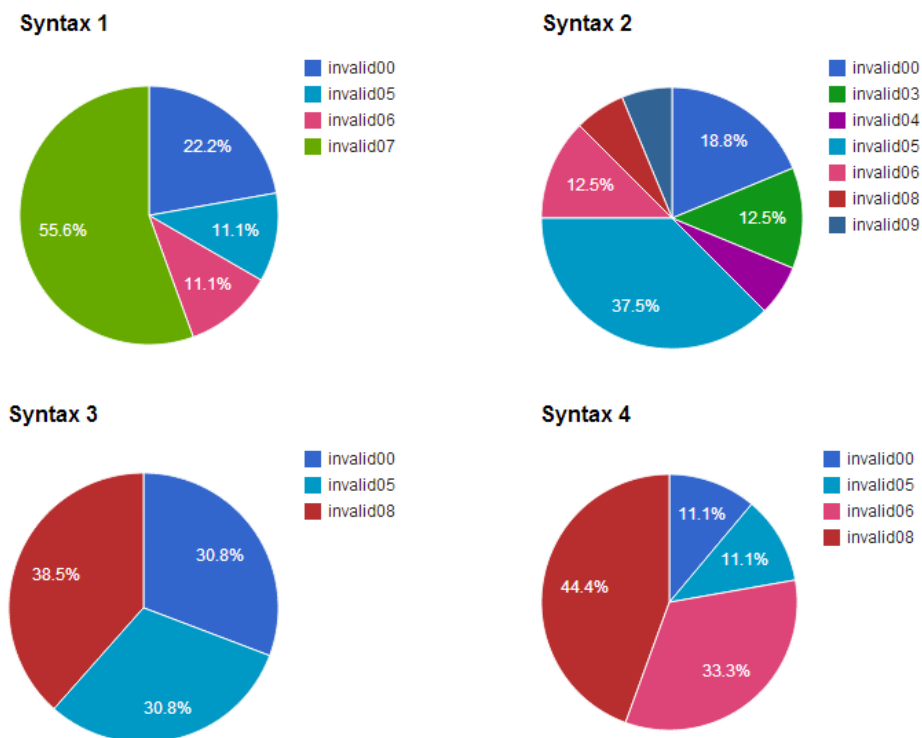


Figure 6.12: Errors' percentages per syntax

medium level. Scenario C resulted to be the hardest with only 8 votes on easy level, 7 on medium level and 1 on hard level.

Syntaxes difficulty

Each syntax was tested twelve times. Syntaxes 1, 2 and 3 were equally judged with 7 votes on easy, 4 on medium and 1 on hard difficulty. Instead, syntax 4 is perceived harder since users gave only 2 votes on easy, 8 on medium and 2 on hard (Figure 6.17).

Cases difficulty

The easiest combination of syntaxes and scenarios resulted to be syntax 3 and 4 with scenario A, and syntax 1 with scenario B. The worst was syntax 3 with scenario C.

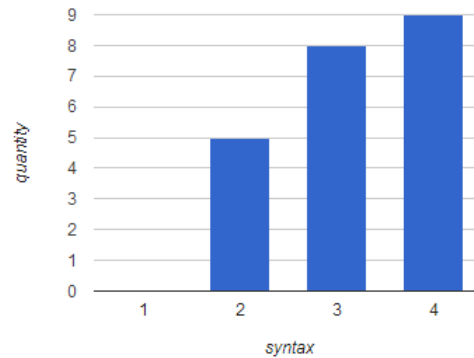


Figure 6.13: Total connection errors per syntax

Syntaxes deficiencies

The most noticed deficiency was the lack of the *Loop* element. In 15 tests, users claimed the necessity to have the *Loop* to correctly model the process they had in mind. Particularly, its absence was felt more in the third syntax.

The *Conditional Gateway* is included in all the syntaxes except for the first, and right in this, in 8 tests over 12, users wanted it.

The *Parallel Gateway* does not exist only in the fourth syntax and its absence was noticed only in 3 tests over 12. In some cases it is possible to overcome the parallel absence using a *Conditional Gateway* with replicated conditions over multiple connections.

Other interesting recursive deficiencies are those regarding the lack of a singular condition and the lack of an early termination. With singular condition we mean the possibility to use a *Conditional Gateway* with a branch that connects directly the opening with the closure without tasks in the middle. This can be useful when there is the necessity to execute a task only when a particular condition is satisfied, otherwise the task must be ignored and the flow of the process can continue.

With early termination we mean a particular element, it can be a task, that once reached, it blocks the process in an error state, or it directly terminates the process. Many times during the experiments it was necessary to replicate the same task representing an early termination of the process because for example some condition to correctly proceed with the process

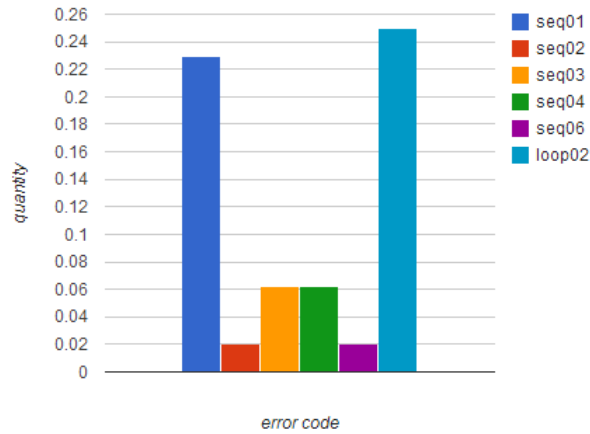


Figure 6.14: Connection errors per single test

was not verified. However, after these tasks, the process flow continued as there were no errors. Related to this problem there is the limitation imposed by the strict block structure: every gateway opened must then be closed with the same number of connections, and inside a block it is not possible to draw a connection to an element located in a different block. So it is not possible to connect a conditional branch directly to the end of the process, nor it is possible to let a task without outgoing connections.

The lack of the events has not gone unnoticed in the second syntax.

Speaking about the *Loop*, in some cases, users would have preferred to have the possibility to connect the backward connection not with a task but with another connection. For example this is useful to go back to a gateway, both parallel and conditional. However, this implies to exit from the current block.

Users noticed also deficiencies on parameters types: someone claimed Boolean parameters but then he resolved using the textual type; other users would have preferred to create lists of parameters and treat them as arrays. In some cases it would have been useful to compare two parameters inside a condition, or sum them or compare them with a value. Using the third syntax, in two cases users would have needed to propagate more than one local parameter, while in other two cases they would have preferred to have global parameters that are more comfortable to use. In the fourth syntax

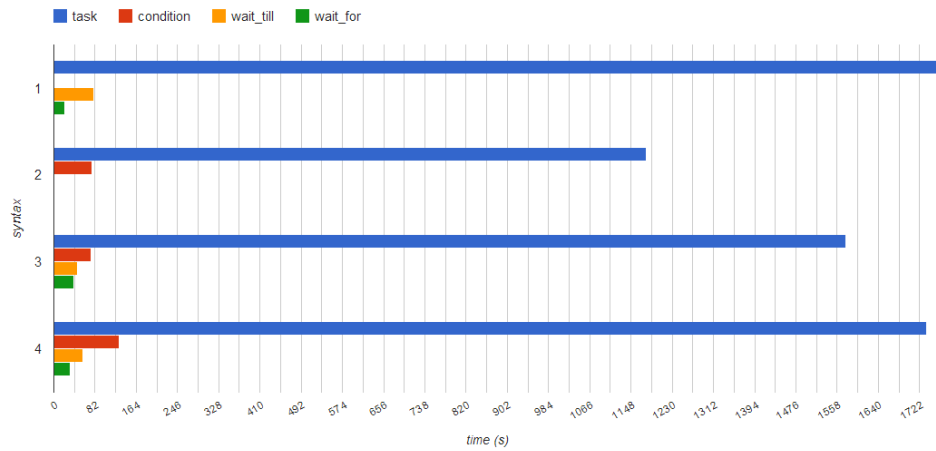


Figure 6.15: Average editing time per process by syntax and element

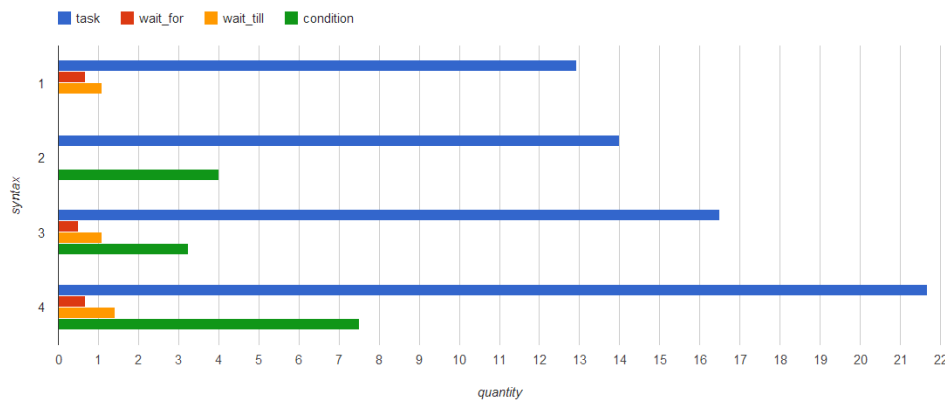


Figure 6.16: Number of info panels opened per single test

instead, in three tests the propagation of multiple local parameters was found too heavy.

After all, even if every syntax has its pro and cons, all the users have succeeded to model the processes, sometimes using alternative, weird but functional methods to bypass restrictions.

Editor deficiencies

Concerning the editor deficiencies and functionality that might have been useful, the most requested was a better time and schedule management of task durations. Particularly it was requested a way to quickly visualize all the deadlines and the temporal limits set. Other interesting features refer

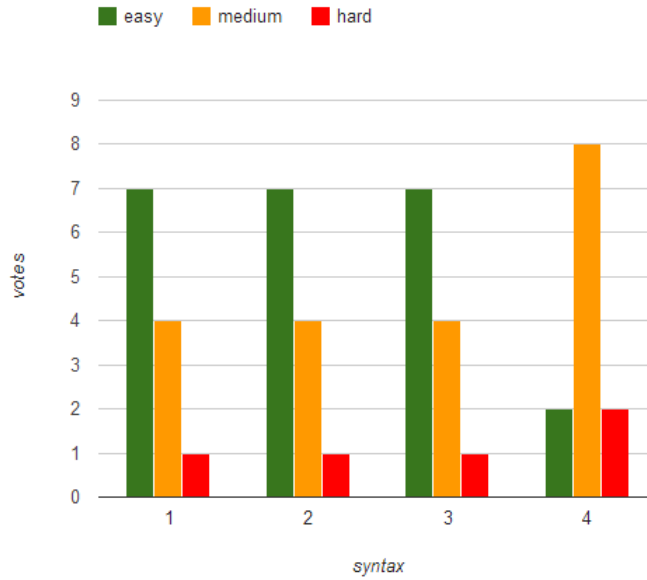


Figure 6.17: Syntaxes difficulty

to possibility to do undo e redo and copy and paste of tasks.

6.4 Conclusions

Analyzing all the data related to the different aspects of the modeled processes during the experiments, we can state that does not exist a syntax which is clearly better than the other ones. Each syntax has indeed its pro and cons.

Probably the best syntax would be the one achievable choosing the best elements of each of the four tested syntaxes, making sure to impose a greater control over the most critical elements that are the most likely to take the user to make mistakes.

Here follow the resuming considerations about the syntaxes, along with a pros and cons list of each one.

6.4.1 Syntax one

It is the quickest and leanest in the processes creation, but in many cases it has turned out to be too poor and ineffective. It is more suitable to de-

scribe very easy and linear processes and it is too limited if there is the need to increase the modeling detail.

Pros	Cons
<ul style="list-style-type: none"> - Small number of elements to learn and a few rules to use them. - Processes are modeled faster with a smaller number of elements and deletions. - Shorter process design time. - No connection errors were done during the experiments. - A few validation errors were done during the experiments. - Smaller number of validation requests and lower percentage of wrong models submitted for validation. 	<ul style="list-style-type: none"> - Lack of specific tools to express the conditional execution. - Harder in medium complex and complex processes modeling. - Longer task editing time. - Poor expressive power.

6.4.2 Syntax two

It is the only syntax which presents all the gateway types: particularly the Loop Gateway has proved to be really useful in the modeling of the more complex passages of the processes.

Pros	Cons
<ul style="list-style-type: none"> - The loop gateway makes possible to model iterative execution - Average modeling time - Number of used elements below the average. - Small number of connection errors 	<ul style="list-style-type: none"> - Lack of specific tools to express time events. - Higher percentage of validation errors. - Higher number of validation requests.

6.4.3 Syntax three

This syntax represents a good compromise between ease of use and descriptive power but the simplicity of a single parameter is, in practice, a great limit.

Pros	Cons
<ul style="list-style-type: none"> - Lower Wait Till and Conditions editing times. - Small number of Wait Till and Conditions changes. - Lower number of users' suggestions in the questionnaire. 	<ul style="list-style-type: none"> - Number of connection errors over the average. - Each time a parameter is added it must be manually propagated, if needed.

6.4.4 Syntax four

This syntax turned out to be the worse one, and was the one more criticized by users.

Pros	Cons
<ul style="list-style-type: none"> - Small number of validation errors - Lower number of validation request with lower percentage of invalidity. 	<ul style="list-style-type: none"> - Connection errors above the average. - Bigger number of elements used. - Each time a parameter is added it must be manually propagated, if needed. - Longer processes creation times. - Longer elements editing times. - Higher number of editing on elements. - The hardest to use according to questionnaire results. - Greater number of suggestions in the questionnaire.

6.4.5 Elements evaluation

To choose the best syntax, also the users' evaluations given in the questionnaire and the notes taken by us during the experiments must be taken in consideration. In fact, in some cases, data collected through the logger may have little meaning if not complemented by the users' impressions.

Keeping in mind all these aspects, we can give a resuming evaluation of the single elements of the four syntaxes.

- **Wait For Event.** It has been by far the least used element and the most deleted. Sometimes, instead of using it, users preferred to use the time constraints inside the tasks, misinterpreting their meaning.
- **Wait Till Event.** Compared to the Wait For event, the Wait Till event has been used and appreciated by the users, and it has been useful to resolve the proposed scenarios.
- **Parallel Gateway.** It is a basic element that has been frequently used when it was available and missed in the syntax where it was not. Sometimes it has been replaced by the Conditional Gateway using equal conditions on the branches.
- **Conditional Gateway.** It has turned out to be fundamental for its great descriptive power. Users have felt the lack of it when it was not available.
- **Loop Gateway.** Looking at the collected data, the Loop has been the most relatively used element, proving to be very useful to resolve the critical aspects of the modeling. When available, it has been really appreciated, while, when missing, users claimed its need and have found harder the process modeling. Conversely, the Loop element has introduced some more errors, so it requires a little more attention in its use.
- **Global Parameters.** Overall, global parameters were easy to understand and much appreciated, despite an initial difficulty of some users that were not confident with the concept of variable and parameter in computer environment (this consideration applies also to local parameters).

- **One Local Parameter.** It has shown its limit when there was the need to receive more than a parameter in a task. Users found unintuitive to propagate and receive the parameter.
- **Multiple Local Parameters.** They have been judged more useful than the one local parameter because they increased the modeling efficiency, but on the other hand, they were criticized because of the heaviness they introduce in the notation.

Chapter 7

Conclusions

7.1 Experience and discussion

In this thesis we have faced the problem of the integration of BPM methods into the management of personal processes. Our goal was the identification of a modeling visual notation powerful enough to describe tasks and their relationships, but also clear and immediate to use for common people.

First of all we have explored and analyzed similar existent applications in order to understand their points of strength and their critical aspects. We have realized that none of them allows to control the execution flow of actions. In fact they are basically to-do lists managing applications: tasks have no dependencies between them, they are just put in sequence, and the notion of process is not always clear.

Then we have studied the Business Process Management Notation with the purpose to identify a set of constructs suitable to describe personal tasks. For each one we have chosen the best graphical solution which is visually expressive and compliant with the Moody's principles. A particular attention was paid to the mechanisms of task assignments, time events, dependencies, and parallel and conditional routing. This was a central point because the introduction of non-basic constructs should not overburden the notation's intuitiveness and usability. Once we have defined the notation, we have prepared four syntaxes, each one made up of a combination of the original set of elements. The idea was to let users evaluate the proposed syntaxes and to identify the best compromise for the description of personal processes.

To do so we have built an application prototype with a graphical editor, which allows modeling daily life scenarios. We have also exploited social networks integration in the managing of user contacts and in the possibility to assign actors to tasks, in order to perform them in a participatory way.

With the help of a logger we have gathered interesting data about the actions performed by users during the experiments, in particular errors and invalid connections. Analyzing such information we have been able to better evaluate the pros and cons of each syntax.

Our study has laid the groundworks for the choice of a language for the description of personal processes, suited both for the context and for the targeted users. The experimentation has shown that this approach works, provided that a good equilibrium between expressive freedom, intuitiveness and ease of use is reached. The social network integration is as powerful as fundamental and must be exploited, specially for manage the notifications.

With respect to previous works on this topic, we have tried a richer notation and the results we have obtained were satisfactory. In fact, end users have correctly understood how to make use of elements, after the early impact. We can state that we have obtained a visual notation with a good and balanced expressive power. Also the graphical editor has worked well and has been appreciated.

7.2 Future work

Here we list some possible future works base on our thesis.

- A first and obvious development must be the engine implementation. To do so, a definitive notation must be chosen. This will make possible to test in a more accurate way the whole system and the validity of the notation. There are however some issues that arise. First of all, processes error states must be managed. This means, for example, that a solution to tasks not performed by the deadline or to unexpected behaviors must be found. A possible solution could be the implementation of a notification mechanism that warns first the actor, if the available time is ending, and then the process admin, if the time has passed without the task completion. Attention must be paid also to the parameters management and to the notification system.

- A second improvement could be done in the direction of social networks integration. First of all, the list of the available social networks can be expanded. Then the problem of tokens' lifetime must be solved. In fact, they must be always up to date in order to let the engine send out notifications in place of the admin.
- The editor and the notation could be enriched with the missing functionality spotted by users during the experiments, such as a summary of all the tasks deadlines or the event of early process termination. Time events can be improved making them parametric.
- It may be interesting the introduction of process templates. They can both serve as example for novice or starting points that allow quicker modeling. They also could be organized in categories.

Bibliography

- [1] Personal Process Management, Michael Rosemann, <http://www.michaelrosemann.com/uncategorized/113/>
- [2] Weber, I., Paik, H.-Y., Benatallah, B., Vorwerk, C., Zheng, L., Kim, S.: Personal Process Management: Design and Execution for End-Users. Technical Report UNSW-CSE-TR-1018, School of Computer Science and Engineering, the University of New South Wales, Sydney, NSW 2052, Australia (2010)
- [3] Marco Brambilla, Application and Simplification of BPM Techniques for Personal Process Management, <http://dbgroup.como.polimi.it/brambilla/personal-process-management-bpms2-paper>
- [4] BPM4PEOPLE, Social BPM Project, <http://www.bpm4people.org/cms/content/en/socialbpm>
- [5] Daniel L. Moody, The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering
- [6] OAuth, open authorization protocol, <http://oauth.net/>
- [7] Draw2D, javascript graphic library, <http://www.draw2d.org/>
- [8] Raphael, javascript graphic library, <http://raphaeljs.com/>