

Documento di Analisi

Release 0.8

November 21, 2009

Firenze



Approvazione, redazione, lista distribuzione

approvato da	il giorno	firma
Marco Tinacci	November 21, 2009	

redatto da	il giorno	firma
Francesco Calabri	November 21, 2009	
Manuele Paulantonio	November 21, 2009	
Massimo Nocentini	November 21, 2009	

distribuito a	il giorno	firma
Daniele Poggi	November 21, 2009	
Niccoló Rogai	November 21, 2009	
Marco Tinacci	November 21, 2009	

Contents

I	Requisiti del prodotto	4
1	Requisiti obbligatori	6
1.1	Generali (2.1)	6
1.2	Diagrammi WBS, Gantt e Task Network (2.2)	6
1.3	Diagrammi specifici (2.3)	6
1.4	Generazione di immagini e doc (2.4)	7
1.5	Documentazione (2.5)	7
2	Semplificazioni, Metriche	8
2.1	Semplificazioni e requisiti aggiuntivi	8
2.2	Metriche	8
II	Domain Model	9
3	Overall diagram	11
4	Aspects descriptions	13
4.1	Task	13
4.2	TaskBox	14
4.3	Strip	14
4.4	Chart	16
4.5	Dependency	16
4.6	UserOption	17
4.6.1	<i>UserOption's</i> Instances	18
4.6.2	<i>UserOptionsChoice</i>	19
4.7	ReportSection	19

Introduzione

Descrizione dell'acronimo: **cdns** sta per "come **d**escritto **n**ella **s**pecifica".

Part I

Requisiti del prodotto

Release 1.0

Chapter 1

Requisiti obbligatori

1.1 Generali (2.1)

Come requisiti fondamentali, **PMango 3.0** sarà visualizzabile e usabile con le ultime versioni *Internet Explorer 8* e *Mozilla Firefox 3.0*.

Le nostre modifiche e aggiunte saranno distribuite senza costi di licenza, in quanto si tratta di estensioni di un progetto GPL

Ci assumiamo la responsabilità di essere conformi ai punti *d)*, *e)*.

1.2 Diagrammi WBS, Gantt e Task Network (2.2)

- a) implementato nello use case **1.3 Show Project Page**.
- b) implementato negli use case **1.2 Make NodeTaskbox** e in **2.2 Make GanttTaskbox**.
- c) implementato nello use case **1.5 Show UserOptions**

1.3 Diagrammi specifici (2.3)

- a) implementato nello use case **1.5¹ Show UserOption**
- b) implementato nello use case **4.2 Create Critical Path Table**
- c) implementato nello use case **1.5 Show UserOptions**
- d) implementato nello use case **1.5 Show UserOptions** e descritto in modo dettagliato nella sezione **7.1 UserOption's Instances** del documento **Domain Model**

¹replace this number with the relative use case within the tasknetwork package

- e) implementato nello use case **1.5 Show UserOptions**

1.4 Generazione di immagini e doc (2.4)

- a) implementato negli use case **1.10 Refresh Chart**, **1.9 Make PDF**
- b) implementato nello use case **1.8 Add to Report UserAction**
- c) implementato nello use case **1.3 Show UserOptions** e descritto in modo dettagliato nella sezione **7.1 UserOption's Instances** del documento **Domain Model**
- d) vedi punto *a)*
- e) implementato nello use case **1.12 Open in New Window**

1.5 Documentazione (2.5)

Ci assumiamo la responsabilità di essere coerenti a quanto richiesto nei punti *a)*, *b)*, *c)* nei momenti in cui verranno effettivamente implementati.

Chapter 2

Semplificazioni, Metriche

2.1 Semplificazioni e requisiti aggiuntivi

- a) il nostro gruppo **non** prevede lo sviluppo di requisiti aggiuntivi, preferendo implementare correttamente il processo di sviluppo adottato per raggiungere i requisiti richiesti dal committente.
- b) abbiamo deciso di creare **solo** oggetti **gif** per usarli in modo interscambiabile sia nella visualizzazione da browser web, sia per aggiungerli in documenti PDF. Questo ci porta alcuni vantaggi:
 - ci interfacciamo con una sola libreria, avendo così modo di capirne a fondo il comportamento e eventualmente aggiungere quelle funzionalità di helper che potrebbero servirci, ma che attualmente non vengono fornite.
 - ci riduce il carico di lavoro, questo non preclude che se arriviamo in anticipo con un prodotto finito e che rispetta la specifica richiesta, potremo proporre una integrazione dell'offerta sviluppando le funzionalità native per la rappresentazione in PDF.

2.2 Metriche

Part II

Domain Model

Release 1.1

Chapter 3

Overall diagram

Questo diagramma comprende tutti i concetti che abbiamo identificato durante la prima iterazione del blocco di analisi.

Nella figura abbiamo una visione di insieme che può essere utile a fini di codifica e progettazione del piano delle prove. Finchè si rimane invece nella sfera della progettazione (analisi inclusa) potrebbe produrre dei dubbi in quanto propone molti concetti; mentre si sta cercando di raffinare le varie relazioni secondo noi è necessaria una vista più in dettaglio di composizioni di pochi concetti che sono legati tra loro, lasciando tutti gli altri ad una loro commento separato.

Procediamo nel seguito del documento nella descrizione di piccole composizioni in modo da chiarire i motivi per cui sono stati creati concetti e relazioni fra essi.

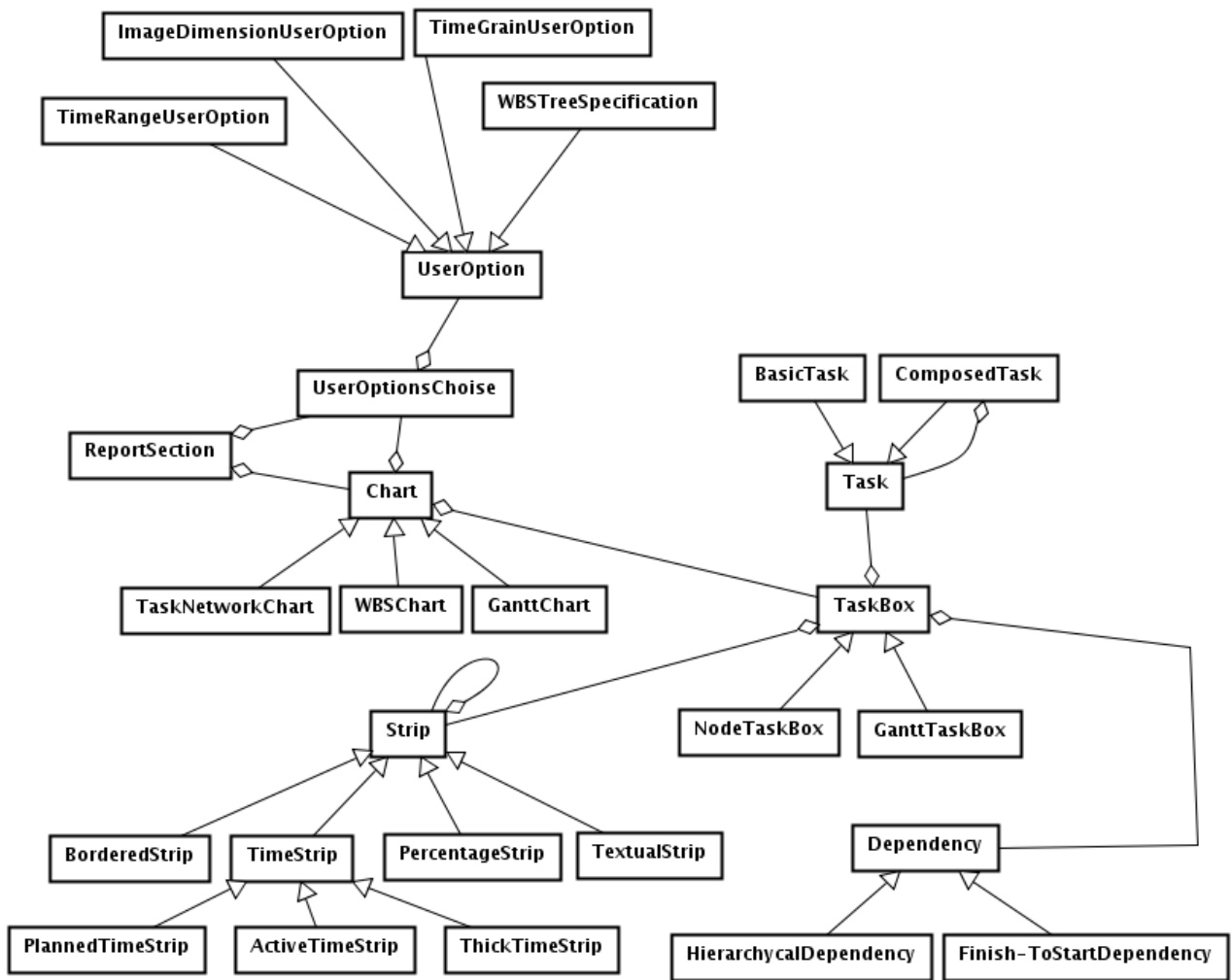


Figure 3.1: Overall UML diagram

Chapter 4

Aspects descriptions

4.1 Task

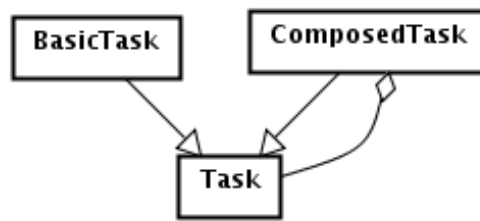


Figure 4.1: task and its relations

Molto probabilmente il concetto di *Task* esiste già nell'attuale versione di **PMango 2.2.0**. Quello che abbiamo pensato è di introdurre un *glue layer* che ci permette di non apportare modifiche al codice esistente di mango, ma lavorare con uno strato di intermezzo per essere il meno intrusivi possibile e poter portare avanti il lavoro dipendendo solo dalle nostri oggetti, facendo il minor riferimento al codice già esistente.

Vogliamo rendere trasparente il concetto che un *Task* sia un'attività singola (non scomponibile in sottoattività) che una attività scomposta.

Costruiamo la relazione \rightarrow che lega questi due concetti:

- *BasicTask* \rightarrow attività di base, non ulteriormente scomponibili
- *ComposedTask* \rightarrow attività che sono composte da sotto attività

In questo modo possiamo trattare questi due tipi di attività in modo interscambiabile e del tutto trasparente. Usando l'astrazione *Task* non ci importa se abbiamo una attività base o composta, in quanto così le abbiamo portate ad avere interfacce compatibili.

4.2 TaskBox

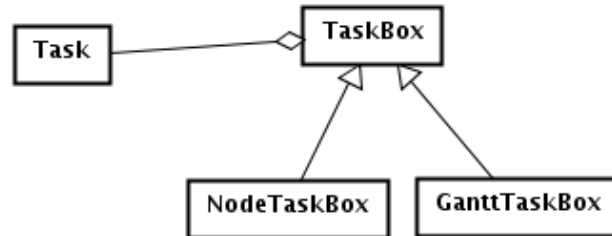


Figure 4.2: taskbox and its specializations

Il *TaskBox* è la rappresentazione grafica di un *Task* (Figure 4.1). Questo concetto astrae su queste specializzazioni:

- *GanttTaskBox* che ci permetterà di costruire la rappresentazione in un *GanttChart* conformi alle norme fissate nel documento di specifica.
- *NodeTaskBox* che ci permetterà di costruire la rappresentazione in un *WBSChart* e in *TaskNetworkChart* alle norme fissate nel documento di specifica.

Abbiamo usato il principio di incapsulare il concetto che varia, modellando il concetto astratto di *TaskBox* per avere questi vantaggi:

- non legare un *Chart* specifico a una rappresentazione specifica
- aggiungere una nuova rappresentazione consiste nel modellarla e dichiarare che si tratta di una specializzazione di *TaskBox*
- potremo cambiare a runtime il tipo di rappresentazione voluta nel disegno di un *Chart*, magari inserire in un *WBSChart* una rappresentazione pensata per i *GanttChart*

4.3 Strip

La *Strip* modella il concetto di "striscia": lo possiamo vedere come il building block di più basso livello di tutta la nostra analisi. Si possono osservare queste relazioni:

TaskBox composition *TaskBox* contiene delle *Strip*, indipendentemente dalla rappresentazione dedicata ad uno specifico *Chart*. Abbiamo costruito questa relazione in quanto per costruire un *TaskBox* sarà sufficiente comporre un insieme di strip, tante quante sono necessarie per la corretta visualizzazione del *Chart* che si sta disegnando.

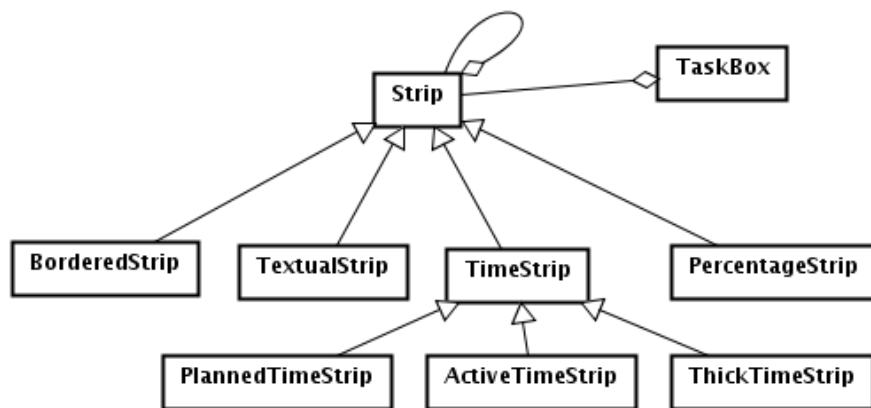


Figure 4.3: kinds of strips

russian doll *Strip* contiene a sua volta delle *Strip*: questo è un concetto che pensiamo possa essere molto potente. Vogliamo rendere la *Strip* un contenitore trasparente rispetto ad oggetti del suo stesso tipo. Questo ci permetterà di disegnare *Strip* annidate, decidendo a runtime sia la **profondità** di annidamento sia l'**ordine** con cui vengono annidate. Otteniamo così l'effetto di *russian doll*, dedicandoci a modellare solo alcune semplici specializzazioni di *Strip*, necessarie per l'implementazione delle notazioni richieste nel documento di specifica, limitandoci poi ad ottenere rappresentazioni complesse annidando quelle semplici.

specializations abbiamo un primo livello di specializzazione:

- *PercentageStrip* rappresenta una quantità (l'intera *Strip*) e la percentuale di completamento (una parte di *Strip* di colore diverso). Può essere composta da un *GanttTaskBox* per indicare la percentuale di completamento relativa al *Task* rappresentato.
- *TextualStrip* permette di inserire delle stringhe di caratteri all'interno della *Strip*. Questa possiamo utilizzarla ad esempio nel *GanttChart* sulla destra della relativa *TaskBox* per indicare l'effort oppure le risorse.
- *BorderedStrip* permette di costruire un bordo, in modo che possiamo implementare la notazione per *field* come richiesto per *NodeTaskBox*
- *TimeStrip* permettono di rappresentare informazioni relative a un intervallo di tempo. Queste sono i building blocks per *GanttChart*. Possiamo specializzare ulteriormente questo concetto:
 - *PlannedTimeStrip* per costruire la parte superiore del *GanttTaskBox* cdns
 - *ActualTimeStrip* per costruire la parte inferiore del *GanttTaskBox* cdns

- *ThickTimeStrip* per costruire la parte superiore del *GanttTaskBox* nel caso la rappresentazione sia di un *ComposedTask* cdns

4.4 Chart

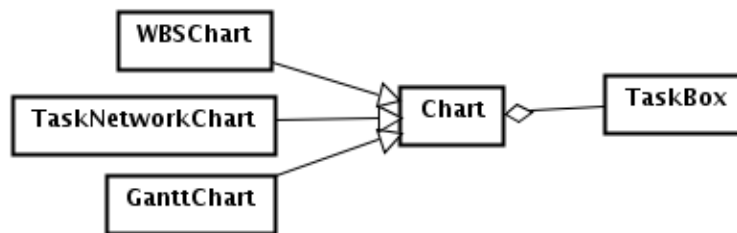


Figure 4.4: chart and building blocks

Il *Chart* modella il concetto di grafico generico. Per implementare la specifica abbiamo queste specializzazioni:

- *GanttChart* che permette di avere la rappresentazione delle attività nel tempo
- *WBSChart* per avere una vista gerarchica delle attività e di come sono state raffinate e decomposte in sotto attività
- *TaskNetworkChart* per rappresentare le dipendenze di tipo *finish-to start* fra coppie di attività

Per costruire un *Chart* è sufficiente assemblare *TaskBox* in base ad alcune preferenze del client che richiede la generazione.

4.5 Dependency

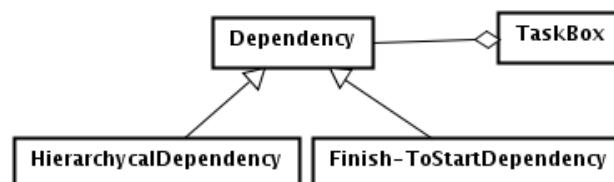


Figure 4.5: dependencies

Dependency modella il tipo di dipendenze che possiamo rappresentare in un *Chart*. Per implementare la specifica abbiamo bisogno di incapsulare queste varianti:¹

- *Finish-ToStartDependency*: siano a, b due *Task* tali che b non può iniziare finché a non sia completato. Questa relazione è catturata da questa specializzazione.
- *HierarchycalDependency*: siano a, b_i con $i = 1, \dots, n \in N$, *Tasks* tali che a è scoposto in b_i *Task*. Questa relazione è catturata da questa specializzazione.

4.6 UserOption

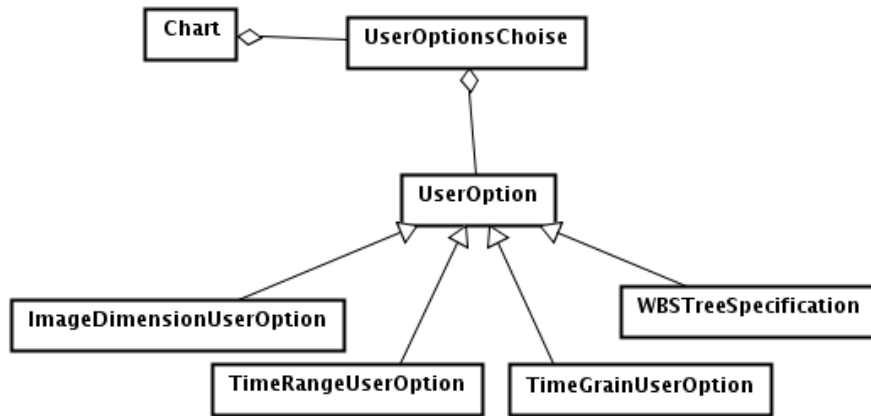


Figure 4.6: UserOptions and choice

Quando un generico client (potrebbe essere sia una persona fisica che un oggetto astratto) della nostra implementazione della specifica vuole generare un *Chart* può guidare la generazione decidendo alcuni fattori che sono di suo interesse. Questi fattori vengono modellati dal concetto di *UserOption*.

Ogni *Chart* espone una lista di *UserOption* per dare al client la possibilità di esprimere quali informazioni guidare. Questa lista varia da *Chart* a *Chart*².

Il concetto di lista di *UserOption* è catturato in *UserOptionsChoice*.

Procediamo per passi: nelle prossime subsection osserviamo due aspetti che trattarli insieme potrebbe non essere sufficiente per esporli in modo chiaro.

¹dire in quali Chart vengono utilizzate

²creare un reference dove vengono mappate questa relazione: potrebbe essere un appendici di questo documento??

4.6.1 *UserOption*'s Instances

Questa è stata una decisione non molto facile da prendere. Il problema è questo: nella specifica abbiamo che per ogni *Chart* il committente ha dichiarato quali *UserOption* mostrare. Queste però non rappresentano un concetto che vogliamo catturare nel nostro modello, ma allo stesso tempo sono *istanze* (un insieme discreto quindi) di elementi che fissa il committente.

Per questo motivo decidiamo di codificare questo insieme discreto in questo documento e la successiva enumerazione è da considerarsi parte integrante del diagramma inserito come figura.

Rappresentiamo il concetto espresso sopra indicando due descrizioni con questa struttura di codifica:

- *istanze*, dove inseriamo tutte le possibili *UserOption* che non possono essere ancora raffinate
- *specializzazioni* dove inseriamo tutte le possibili specializzazioni di *UserOption* che possono essere ancora raffinate, ripetendo in modo ricorsivo questa struttura di codifica

Le successive descrizioni sono relative al concetto di *UserOption*:

istanze ³ *WBSExplosionLevelUserOption, ActualTimeFrameOption, CompletionBarOption, PlannedDataOption, ActualDataOption, AlertMarkUserOption, ReplicateArrowUserOption, FindCriticalPathUserOption, WBSUserSpecificationUserLevel, WBSUserSpecificationUserLevel, ResourcesDetailsOption, TaskNameOption, CompleteDiagramUserOptions*

specializzazioni

- *WBSTreeSpecification*
istanze *LevelSpecification, UserCustomSpecification*
specializzazioni nessuna
- *TimeGrainUserOption*
istanze *WeaklyGrain, MonthlyGrain*
specializzazioni nessuna
- *ImageDimensionUserOption*
istanze *CustomDim, FitInWindowDim, OptionalDim, DefaultDim*
specializzazioni nessuna
- *TimeRangeUserOption*
istanze *CustomRange, WholeProjectRange, FromStartRange, ToEndRange*
specializzazioni nessuna

³inserire qui la il mapping sui vari Chart?

4.6.2 *UserOptionsChoice*

Questo concetto è, secondo la nostra analisi, molto importante in quanto ci permette di astrarre dal client che richiede una generazione.

Il motivo per cui abbiamo introdotto questo concetto è di poter lavorare lato server usando *UserOptionsChoice* per controllare quali informazioni il client vuole guidare. In questo modo non siamo vincolati ad accedere ai dati inviati per *POST*, *GET* dalla form HTML, ma possiamo direttamente guardare in *UserOptionsChoice*. Queste ci permette di disaccoppiare il processo di generazione della maschera di input di una pagina HTML.

Se vogliamo utilizzare il processo di generazione (che comunque è server side) scrivendo un programma client (GUI o da riga di comando) che costruisce una HTML request ad hoc (dovremo definire una grammatica e attribuire la semantica ai contesti, questo è necessario, non ch  scrivere un parser), usando *UserOptionsChoice* e il suo disaccoppiamento ci sar  possibile farlo.

Una volta ricevuta la response possiamo maneggiare la pagina inviata come una response HTML valida e usarla per i nostri obiettivi (possiamo richiedere l'immagine generata, o il file PDF generato, salvandolo in locale, oppure visualizzando lo stesso con un browser, ma possiamo anche inserire in un db oppure farci dei test sopra...).

Dovremo quindi costruire un oggetto che si incarica di costruire *UserOptionsChoice* in base al tipo di richiesta ricevuta (da una pagina html come   il caso di PMango, oppure una richiesta da un client indipendente scritto in un qualche linguaggio). Una volta costruito l'insieme delle *UserOption*   possibile iniziare la generazione. Questo sar  delegato alla fase di progettazione.

4.7 ReportSection

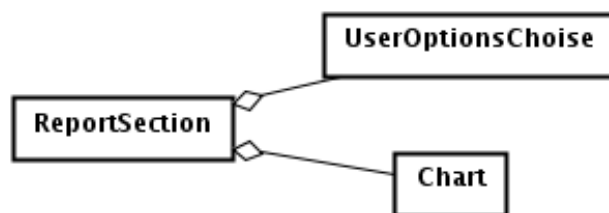


Figure 4.7: report section

Abbiamo da implemetare un requisito che vuole la possibilit  di aggiungere alla reportistica un determinato *Chart* con le relative *UserOption* scelte dall'utente. Modelliamo quindi il concetto di *ReportSection* per realizzare questo requisito. Come si vede dalla figura, *Report-*

Section associa *Chart* e *UserOptionChoice*. Utilizziamo direttamente la lista delle scelte⁴ in modo da non doverla costruire nella funzionalità di assemblamento del report.

⁴che viene costruita lato server