Domain Model

Release 1.1

November 16, 2009
Firenze

Approvazione, redazione, lista distribuzione

approvato da	**il giorno**	**firma**
Marco Tinacci	16/11/2009	

redatto da	**il giorno**	**firma**
Francesco Calabri	16/11/2009	
Manuele Paulantonio	16/11/2009	
Massimo Nocentini	16/11/2009	

distribuito a	**il giorno**	**firma**
Daniele Poggi	16/11/2009	
Niccoló Rogai	16/11/2009	
Marco Tinacci	16/11/2009	

Contents

1 Overall diagram **5**

2 Task **6**

3 TaskBox **6**

4 Strip **7**

5 Chart **8**

6 Dependency **9**

7 UserOption **10**
7.1 *UserOption*'s Instances 10
7.2 *UserOptionsChoice* 11

8 ReportSection **12**

Introduzione

1 Overall diagram

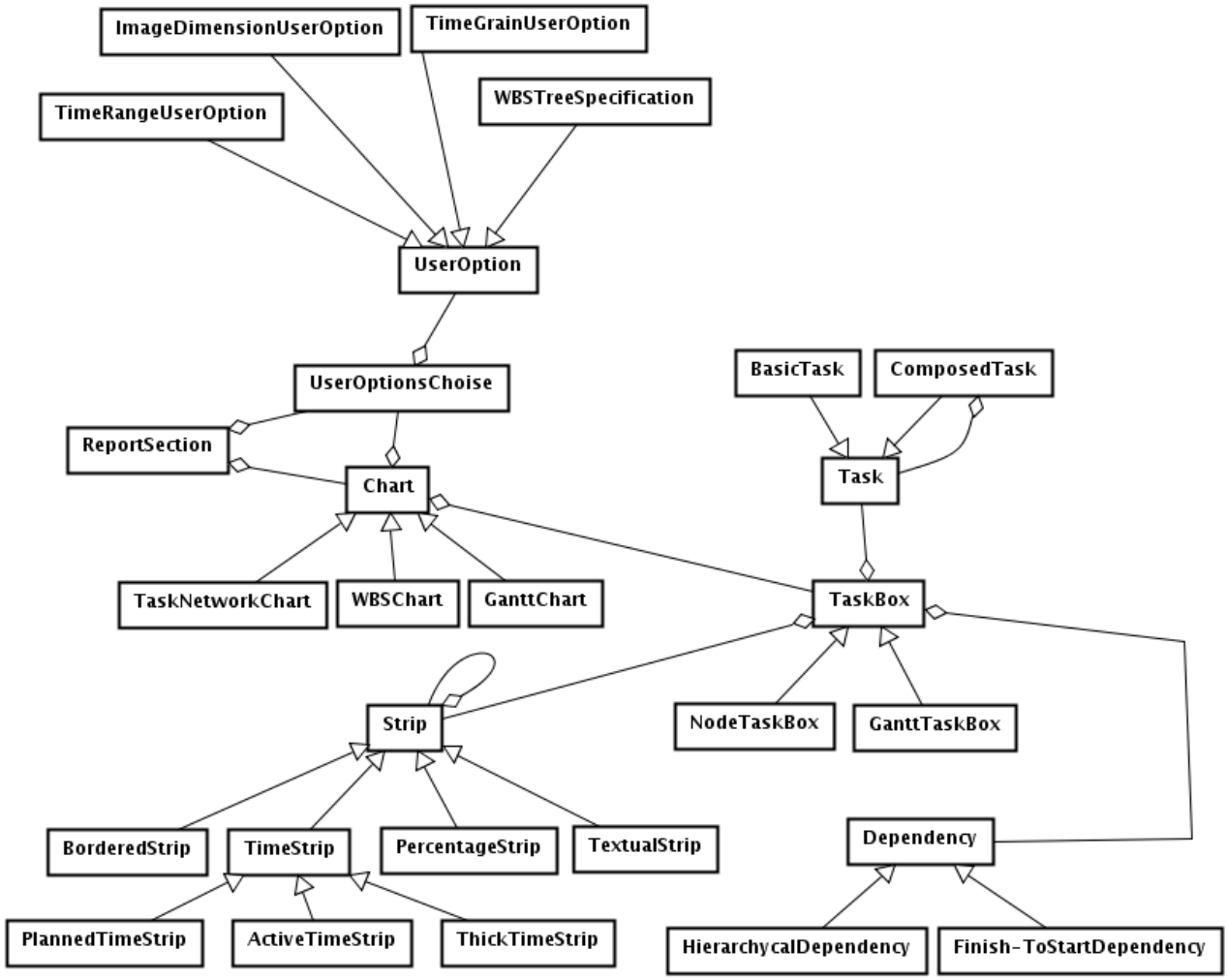

Figure 1: Overall UML diagram

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.

2 Task

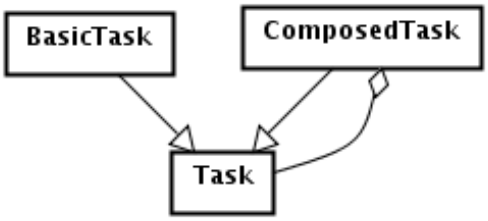

Figure 2: task and its relations

Molto probabilmente il concetto di *Task* esiste gia 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 gia esistente.

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

Costruiamo la relazione → che lega questi due concetti:

- *BasicTask* → attività di base, non ulteriormente scomponibili
- *ComposedTask* → 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 cosi le abbiamo portate ad avere interfaccie compatibili.

3 TaskBox

Il *TaskBox* è la rappresentazione grafica di un *Task* (Figure 2). 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.

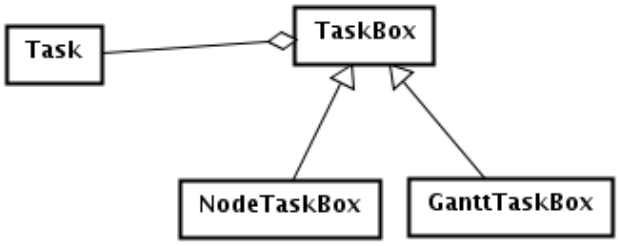

Figure 3: taskbox and its specializations

- *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 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.

russian doll *Strip* contiene a sua volta delle *Strip*: questo è un concetto che pensiamo posso 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 cosi 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.

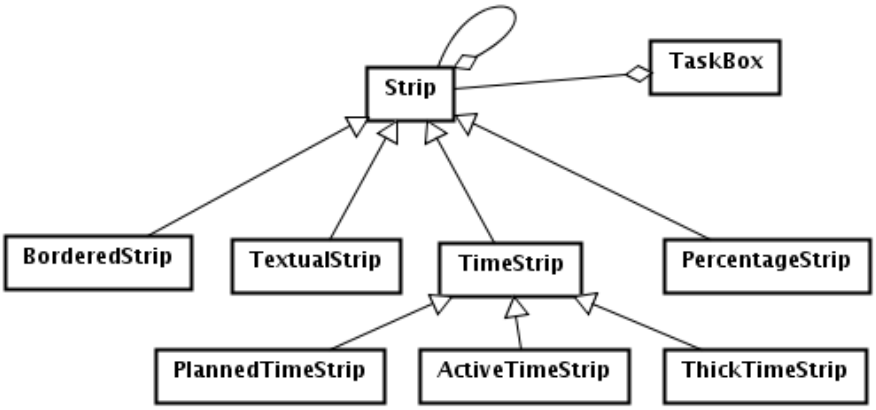

Figure 4: kinds of strips

specializations abbiamo un primo livello di specializzazione:

- *PercentageStrip* rappresenta un quantità (l'intera *Strip*) e la percentuale di comletamento (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

5 Chart

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

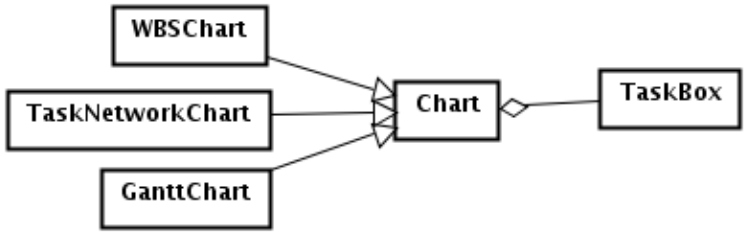

Figure 5: chart and building blocks

- *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.

6 Dependency

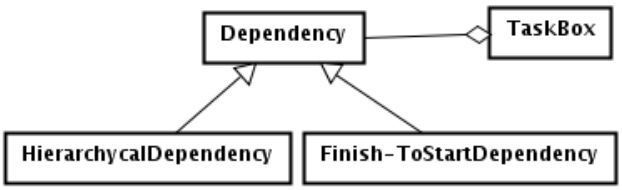

Figure 6: dependencies

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 $i = 1, \ldots, n \in N$, *Task*s tali che a è scoposto in b_i *Task*. Questa relazione è catturata da questa specializzazione.

[1] dire in quali Chart vengono utilizzate

7 UserOption

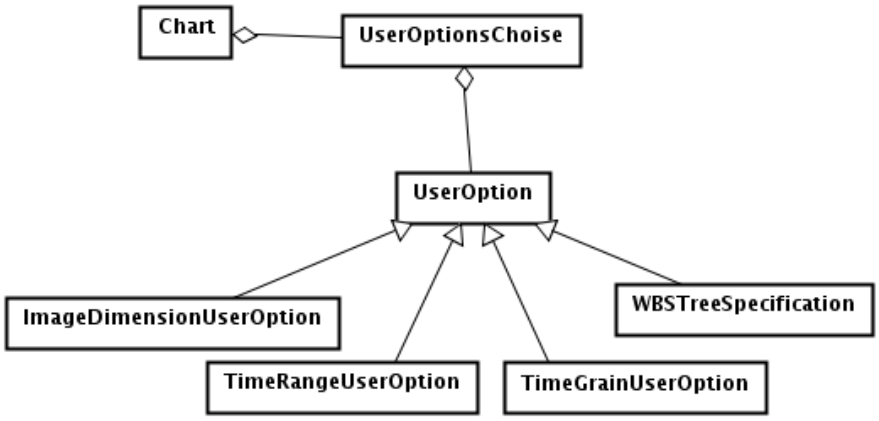

Figure 7: UserOptions and choice

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.

7.1 *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.

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

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, CompletitionBarOption, 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

7.2 *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

[3] inserire qui la il mapping sui vari Chart?

non siamo vincolati ad accedere ai dati inviati per *POST, GET* dalla form HTML, ma possiamo direttamente guardare in *UserOptionsChoice*. Queste ci permette di disaccopiare 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 chè scrivere un parser), usando *UserOptionsChoice* e il suo disaccoppiamento ci sarà possibile farlo.

Una volta ricevuta la response possiamo maneggiare la pagina inviata come una response HTML valida e usarla per i nostri obiettivi (possiamo rihiedere l'immagina 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 è il caso di PMango, oppure una richiesta da un client indipendente scritto in un qualche linguaggio). Una volta costruito l'insieme delle *UserOption* è possibile iniziare la generazione. Questo sarà delegato alla fase di progettazione.

8 ReportSection

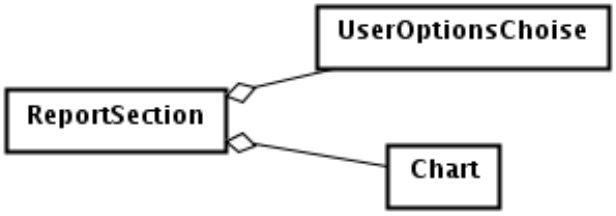

Figure 8: report section

Abbiamo da implemetare un requisito che vuole la possibilità 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, *ReportSection* associa *Chart* e *UserOptionChoice*. Utilizziamo direttamente la lista delle scelte[4] in modo da non doverla costruire nella funzionalità di assemblamento del report.

[4]che viene costruita lato server