Adrien Naigeon S3D

Tom Fouquet

Doryann Becker

SAE Développement d'application

Rapport Final

Table des matières

Fonctionnalités

Fonctionnalités réalisées

- Trois affichage : tableau/liste/GANTT
- Ajouter/supprimer/modifier une tâche
- Dépendances
- Collaborateurs
- Sous-tâches
- Ajouter/supprimer/modifier une liste
- Drag and Drop
- Archivage
- Création d'un nouveau fichier
- Enregistrer un fichier / ouvrir un fichier
- Exporter un fichier en image

Répartition des tâches

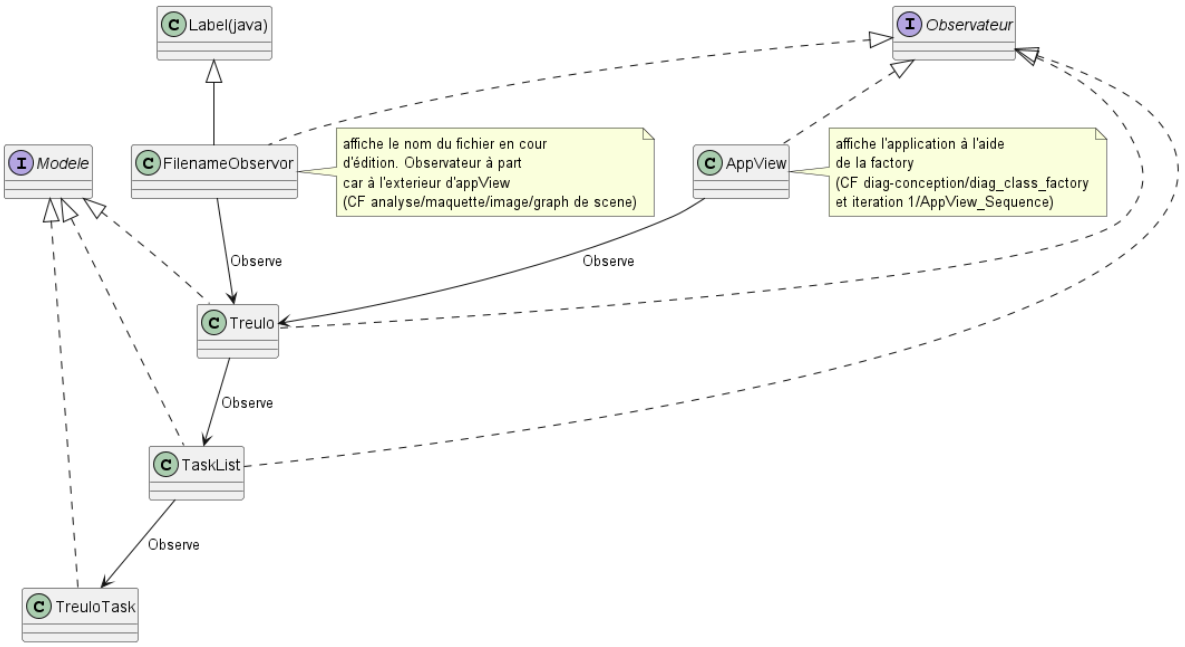
Lien vers le trello :
https://trello.com/invite/b/TWFbf2Vf/ATTI36e717da2909e99b85e7417f27de741aEB278C22/logiciel-dorganisation-de-taches-personnelles

Doryann	Adrien	Tom
-Création classe Tasklist et Task -Modifier/supprimer/créé liste de tâche -Modifier une tâche -Changer mode d'affichage -Affichage liste -Tâche en détail -Liste de dépendance dans vue de création de tâche -Vue tâche en détail -Edité liste de dépendance/collaborateurs -Sous-tâche avec durée cohérante -TextArea à taille variable -Edition/ajout de tâche -> ne pas afficher la tâche dans sa propre dépendance	-Factory d'affichage -Supprimer une tâche -Vue bureau -Vue liste -Drag and drop -Archivage -Gestion de fichier -Export en image -Edition/ajout de tâche -> ne pas afficher la tâche dans sa propre dépendance -Commentaires	-Création classe Treulo (modèle) MVC -Vue de création de tâche -Retour en arrière -Déroulé des tâche (vue liste) -Liste collaborateurs -Vue Gantt

Diagramme de classe

Voici un diagramme de classe (très) simplifié de l'application afin de mettre en lumière sa structure. Des diagrammes plus détailler son disponible dans Document/iteration6/diag_classe_automatique

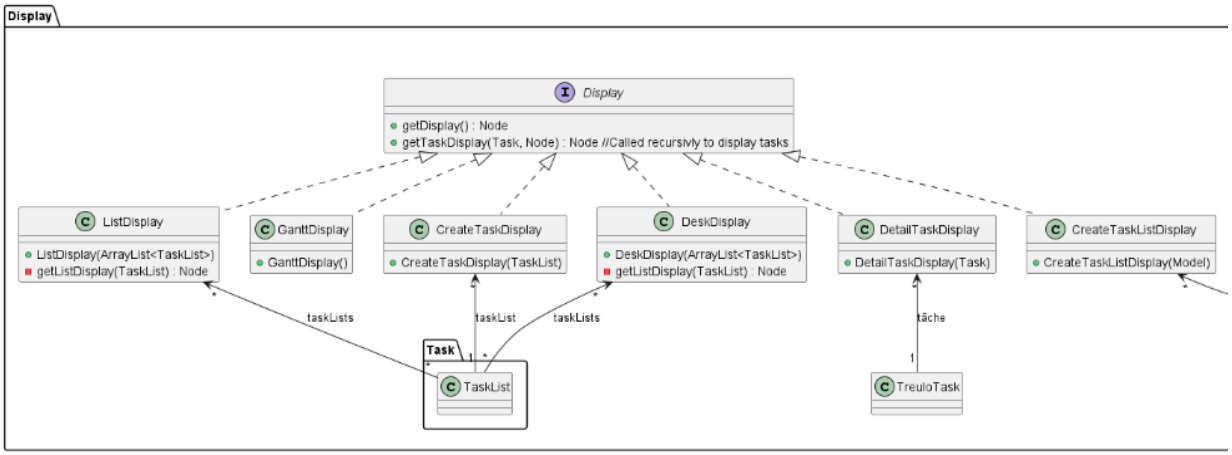
note : le diagramme sans contrôleur peut comporter des incohérence suite à la suppression des contrôleur

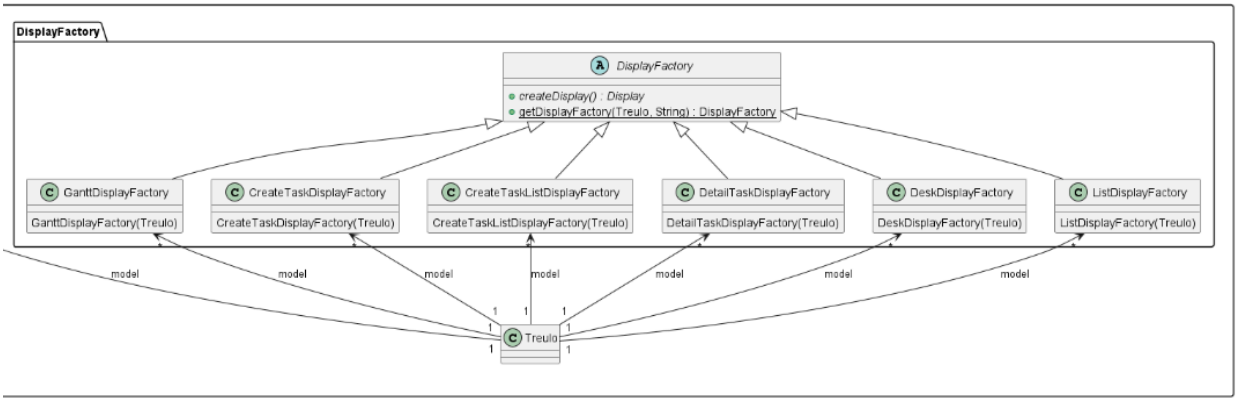


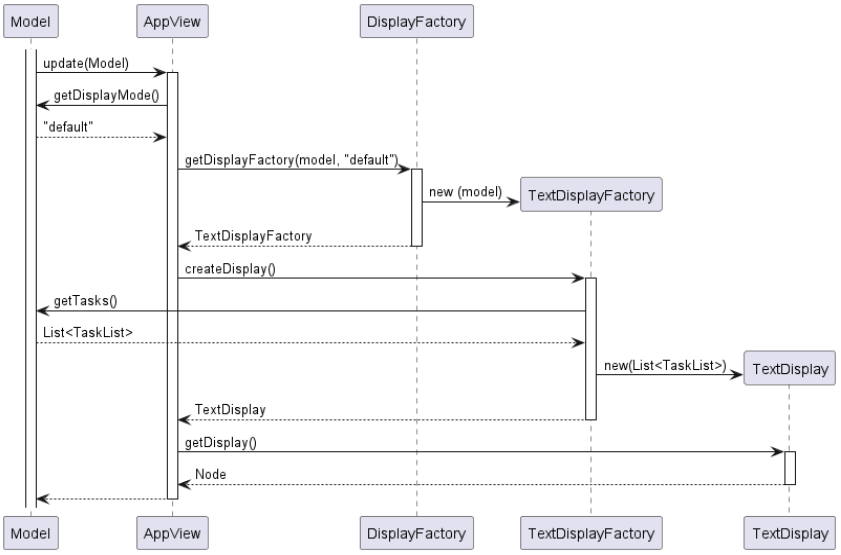
Eléments originaux dont nous sommes fières

Adrien - Factory et AppView

J'ai travaillé sur la partie génération de l'affichage de l'application : il est généré par une usine qui va construire le bon objet Display (afficheur) en fonction des besoins. Les afficheurs sont ainsi des classes à part qui peuvent être éditer sans risque d'interférer avec une autre partie de l'application, et surtout de manière très simple (puisque l'objet AppView qui utilise l'afficheur est déjà placé dans la scène, pas besoin de gérer cette partie en créant l'afficheur)

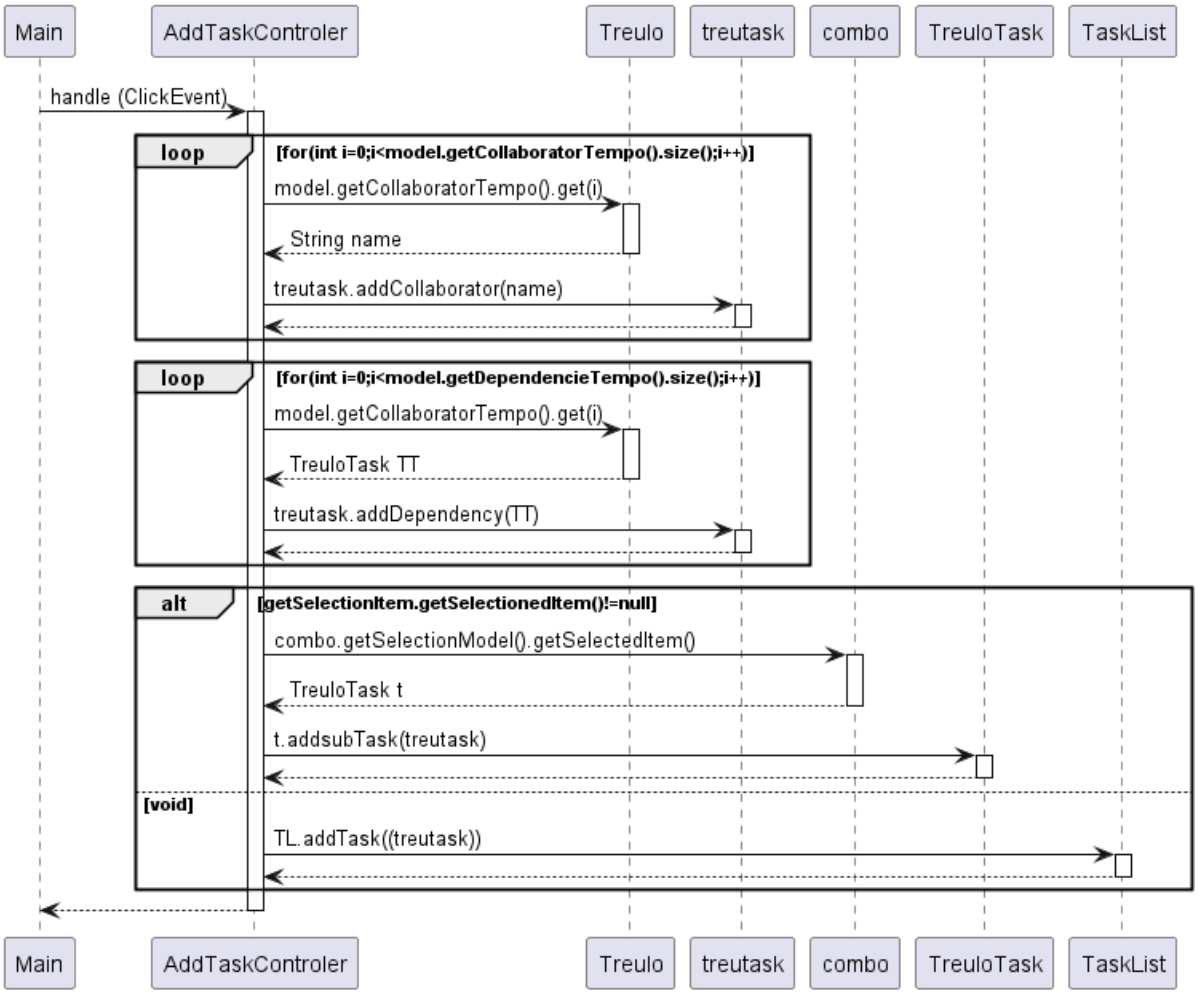






Tom - Ajouter une tâche :

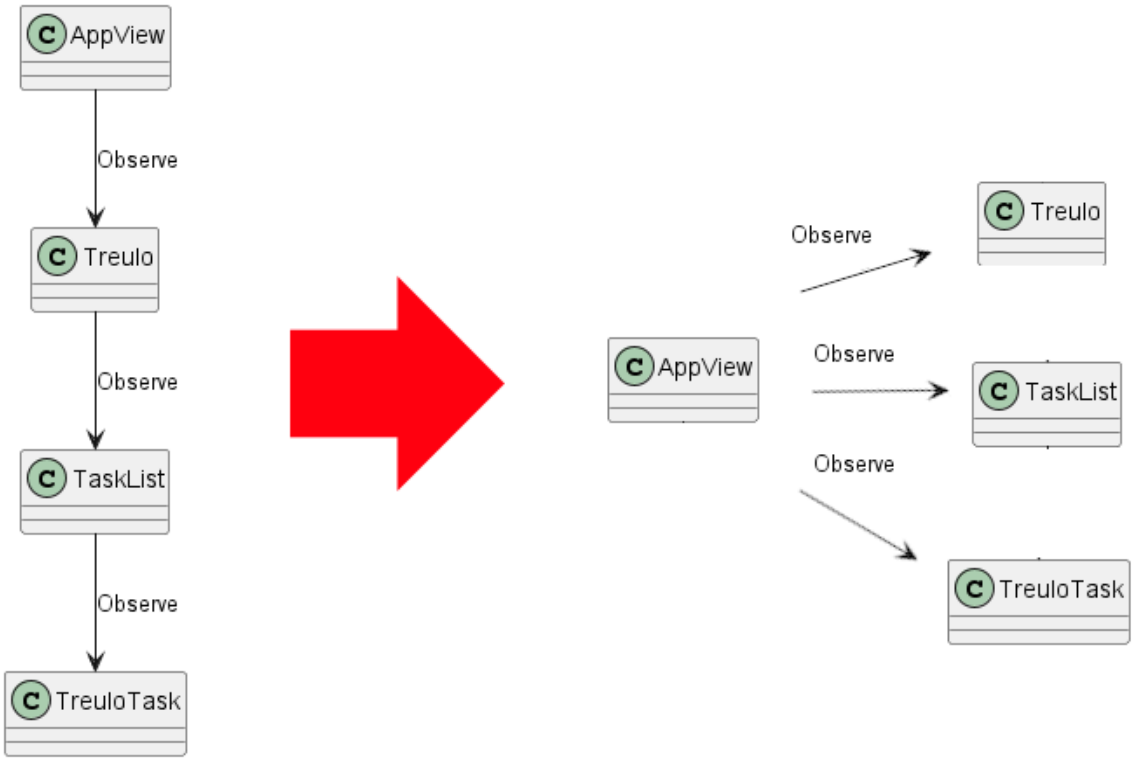
J’ai travaillé sur la création de tâches, celle-ci possède un affichage propre à elle afin de pouvoir ajouter tous les détails dont on a besoin, par exemple les Dépenses ou encore les Collaborateurs. J’ai dû créer des attribut temporaire dans le modèle afin de pouvoir ajouter toute les différentes dépendances et collaborer lorsque l'on appuie sur ajouter j’ai dû faire comme cela car la tâche n'existe pas encore. Le modèle possède aussi un attribut permettant de savoir dans quelle liste il veut ajouter la tâche.
L’action peut être représenter par le diagramme de séquence suivant :



Doryann

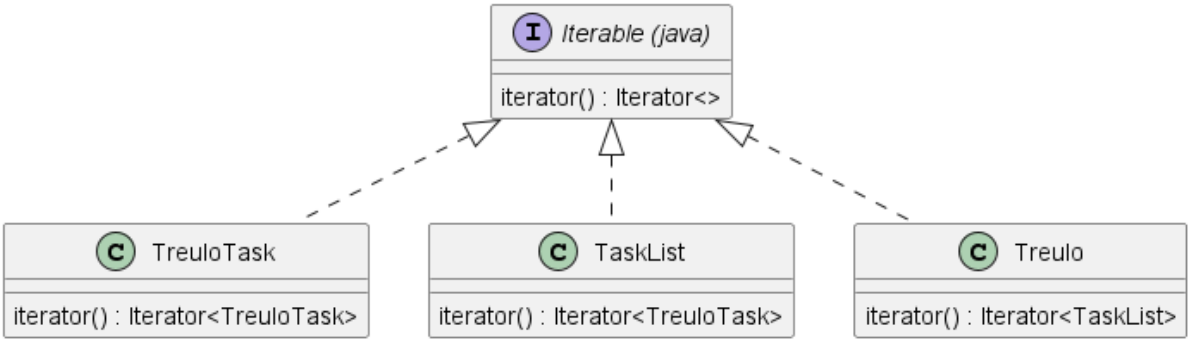
Modification par rapport à l'étude préalable

La structure de Treulo est une chaîne d'observateurs qui fait remonter les mises à jour à "AppView", l'affichage. Cependant cette conception est difficile à maintenir, en effet ajouter de nouveaux composants demanderais de l'ajouter à cette chaîne d'observation. Pour régler ce problème, nous avons envisagé de faire en sorte que AppView observe tous les composants, afin de pouvoir plus facilement en ajouter de nouveaux. Cette conception nous à néanmoins porté chance puisque pour la sérialisation, la chaîne d'observateur est sauvegardée, ce qui n'aurait pas été le cas avec la nouvelle conception.

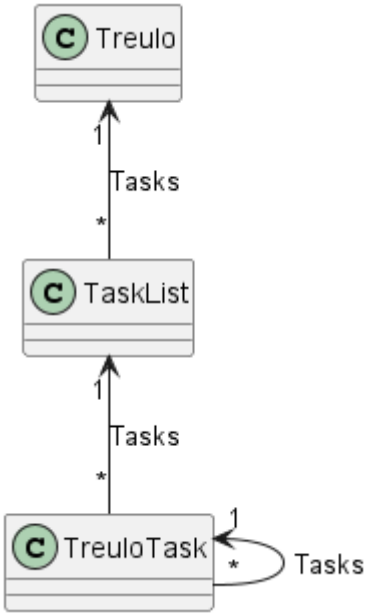

AppView
Observe
Treulo
Observe
TaskList
Observe
TreuloTask
Observe
Treulo
AppView
Observe
TaskList
Observe
TreuloTask

Patron/architecture utilisés

- Factory (Voir “élément dont nous somme fière / Adrien”)
- MVC (Voir “Modification par rapport à l’étude préalable”)
- Iterator



- Composite



Graphe de scène

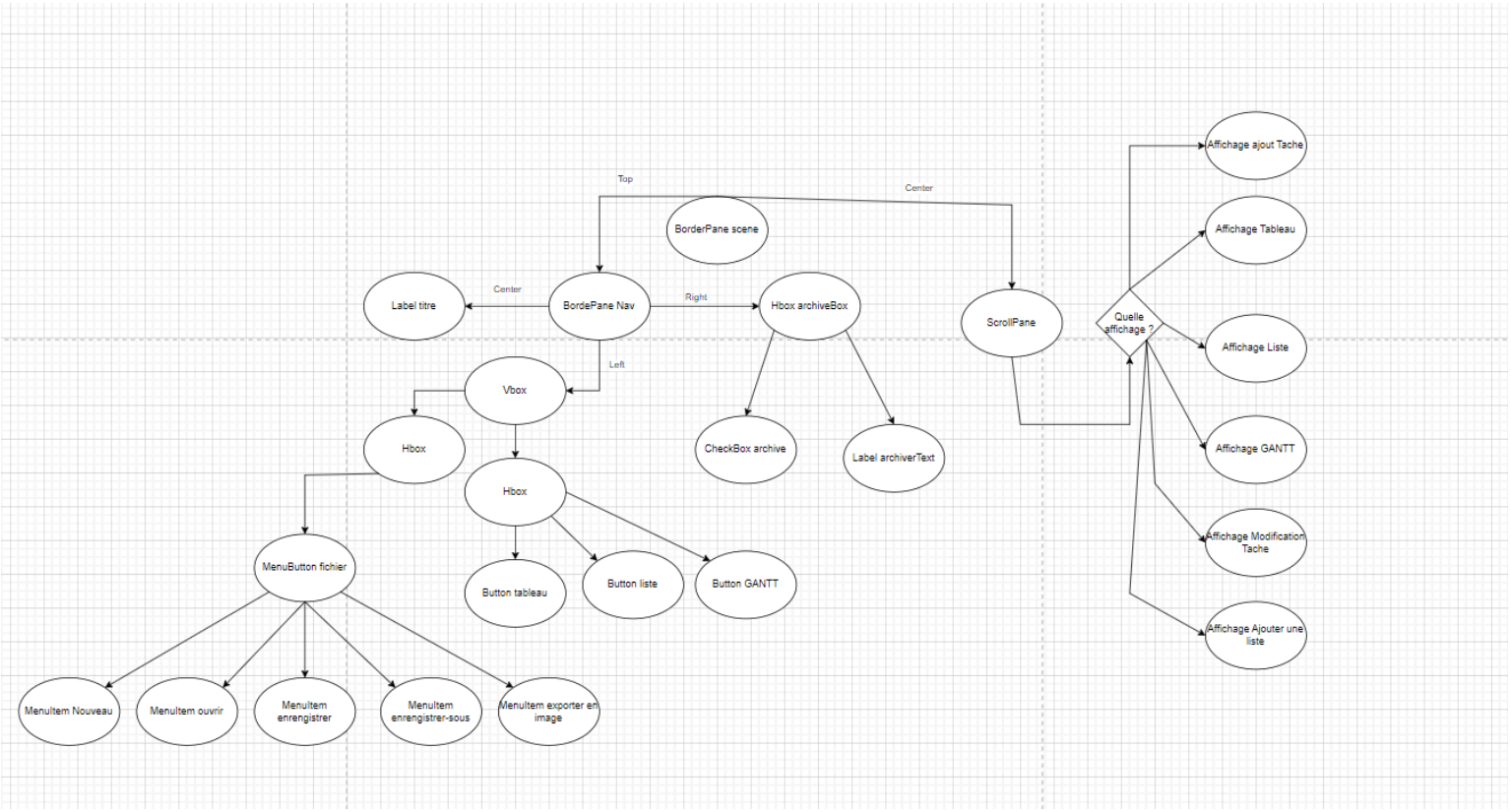
Changement de vue

Le changement de vue est effectué en modifiant la variable “displayMode” du modèle. La fabrique s’occupe ensuite de modifier l’affichage en fonction de ce paramètre. (CF “Éléments dont nous sommes fière / Adrien”)

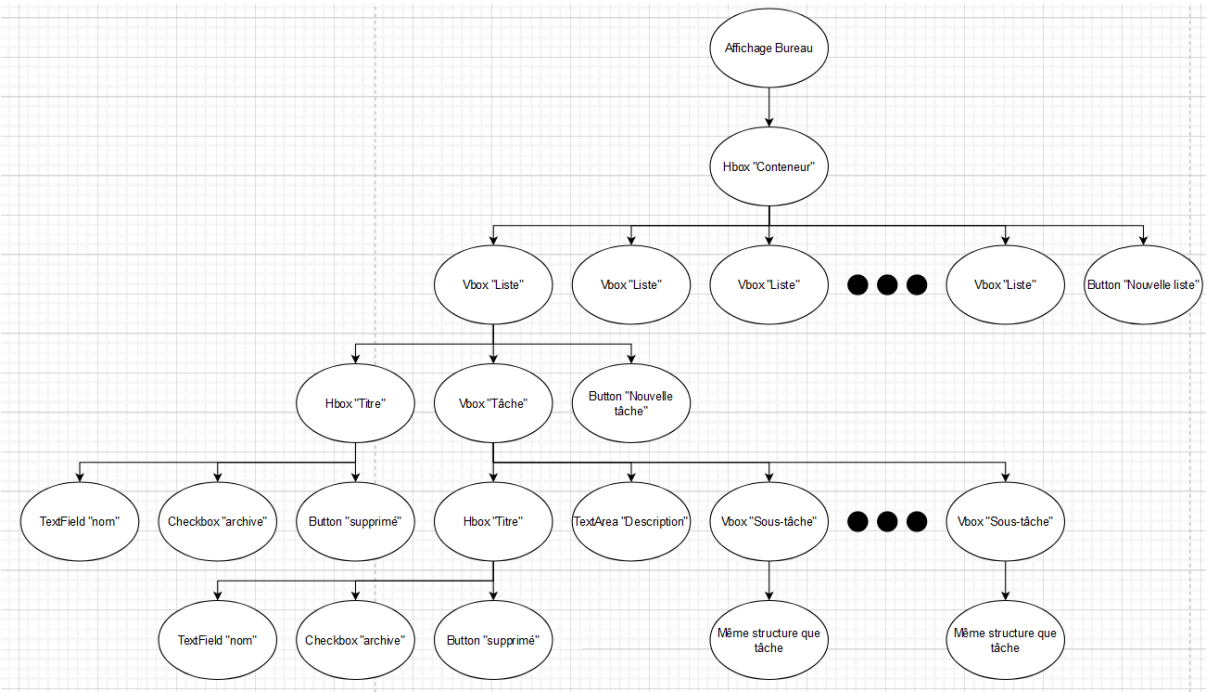
```
Fiera42 +2
public static DisplayFactory getDisplayFactory(Treulo model, String displayMode) {

    //Création de l'afficheur en fonction du mode d'affichage
    switch (displayMode) {
        case "Nouvelle tache":
            return new CreateTaskDisplayFactory(model);
        case "Nouvelle Liste":
            return new CreateTaskListDisplayFactory(model);
        case "Tableau" :
            return new DeskDisplayFactory(model);
        case "Liste" :
            return  new ListDisplayFactory(model);
        case "GANTT" :
            return new GanttDisplayFactory(model);
        case "Detaille Tache" :
            return new DetailTaskDisplayFactory(model);
        default :
            return new TextDisplayFactory(model);
    }
}
```

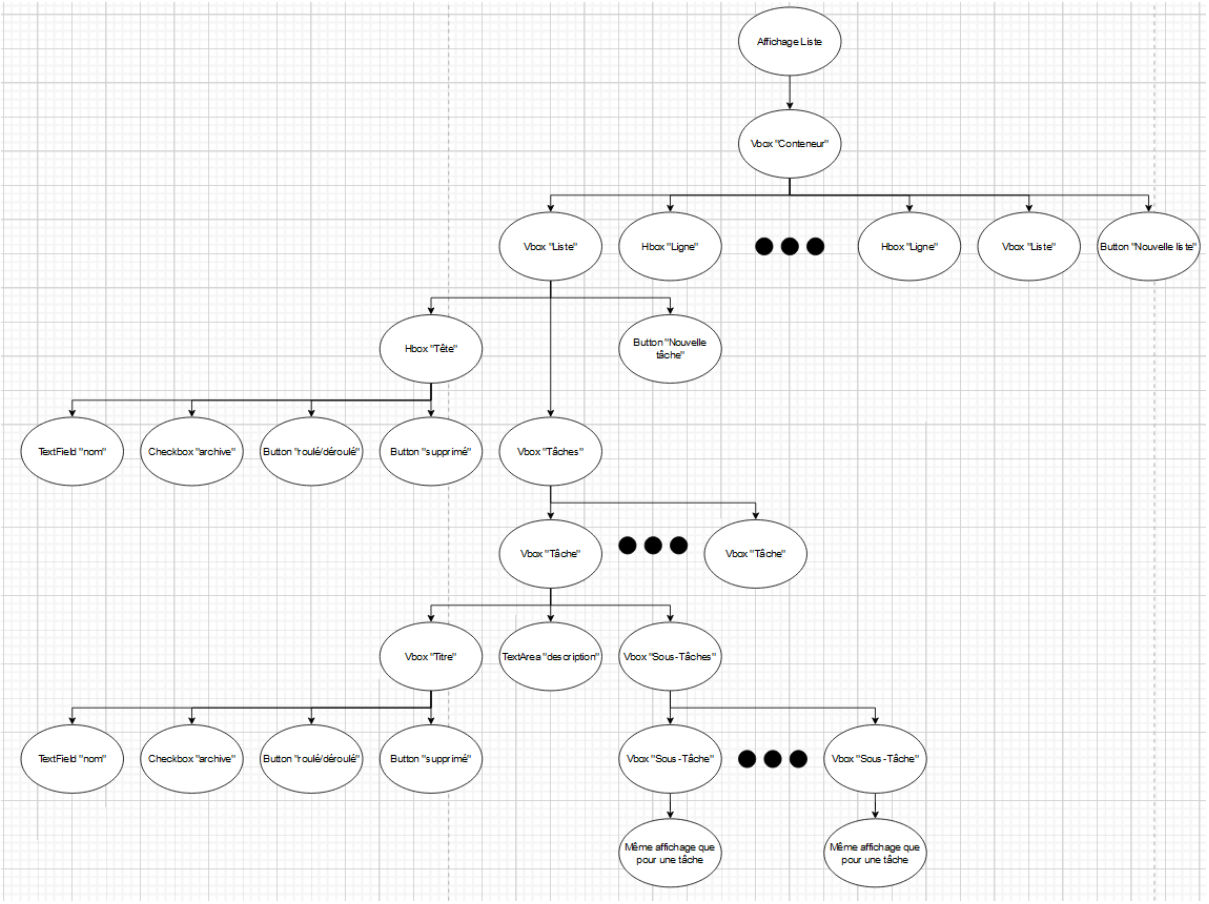
Graphe de scène global



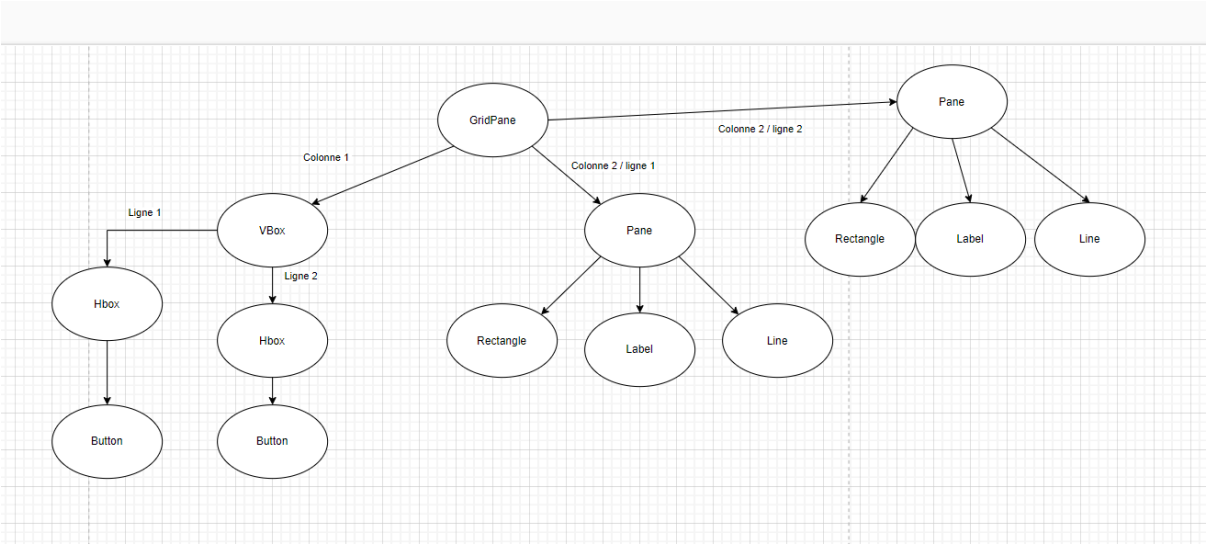
Graphe de scène vue Bureau



Graphe de scène vue Liste

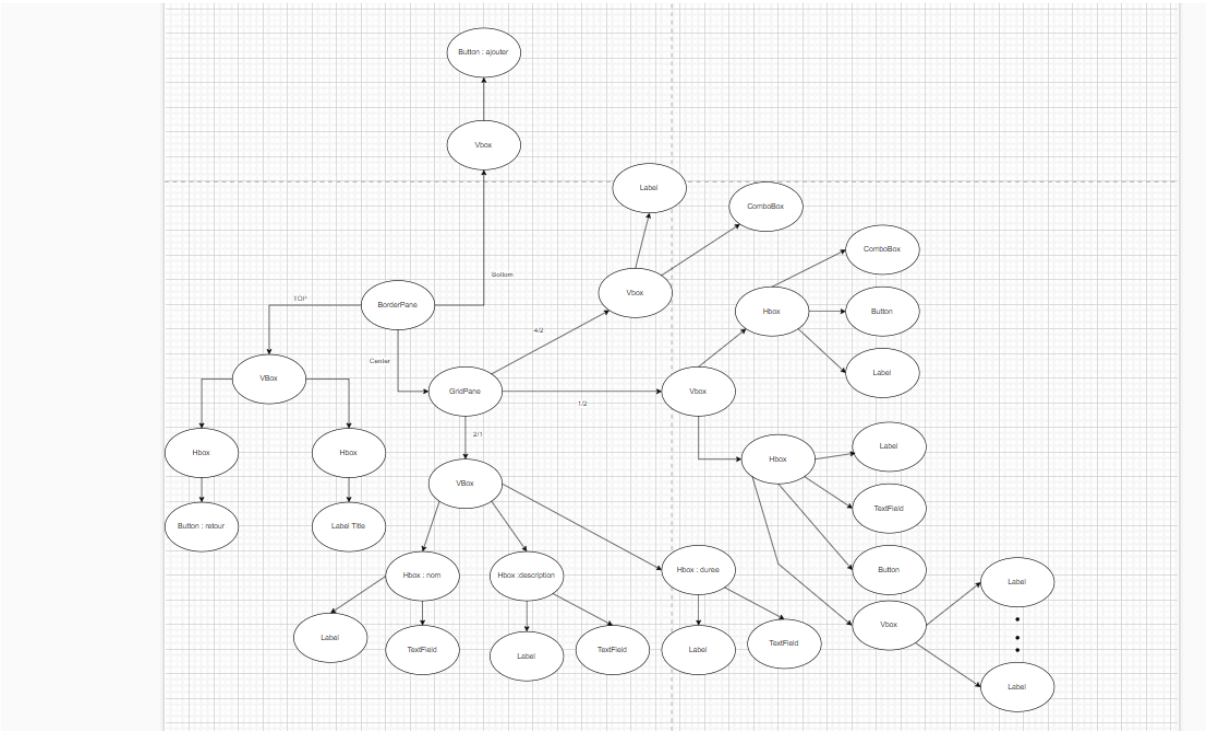


Graphe de scène vue Gantt

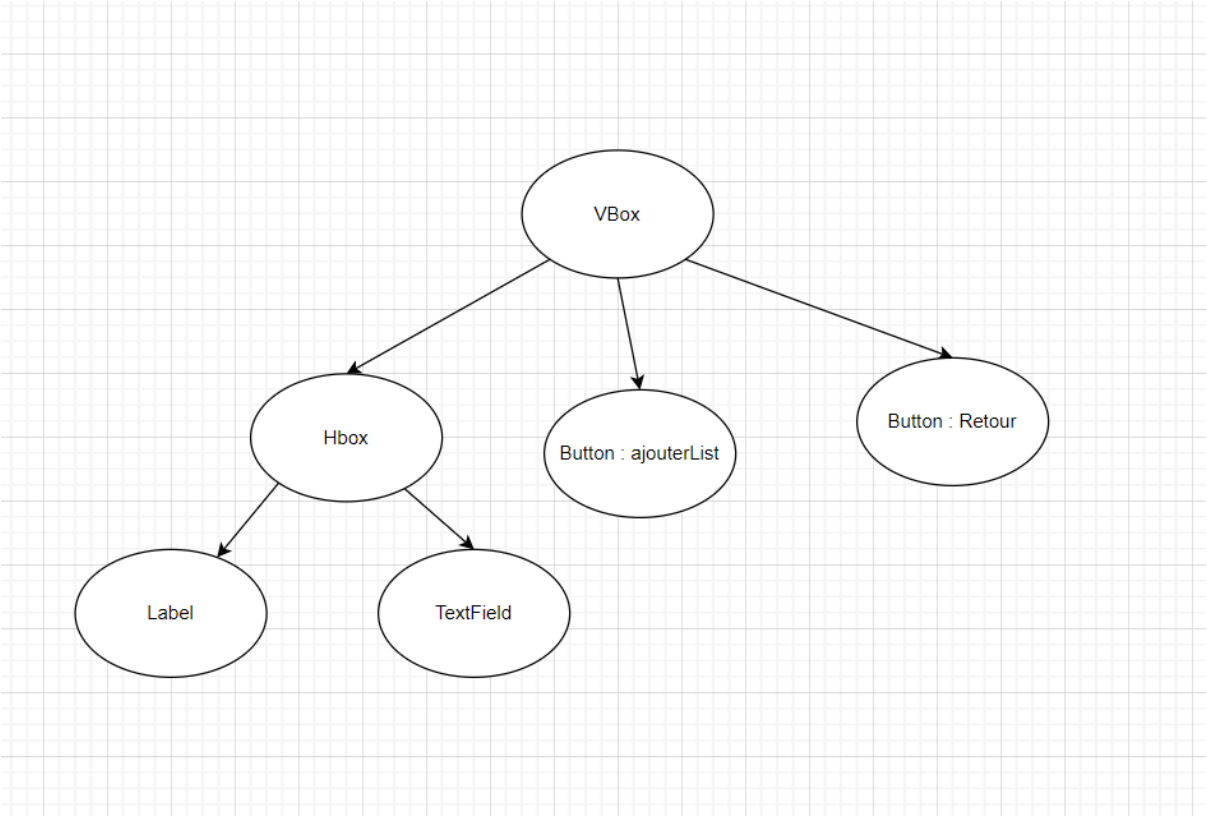


Il y a autant de Hbox et de Pane qu'il y a de tâche à chaque fois on va incrémenter la ligne de 1.

Graphe de scène vue Création de tâche



Graphe de scène vue Création de liste



Graphe de scène vue Tâche en détail