SAE 3.01 – Développement d'application

Ollert

Enzo M. - Grégoire H. - Mathéo P. - Stéphane L.

Fonctionnalités réalisées

- Page
    - Déplacer une liste de tâche
    - Créer une liste de tâche
    - Changer d'affichage (Tableau et Tableur)
    - Générer le Gantt
- Liste de tâches
    - Supprimer la liste
    - Modifier le titre
    - Créer une tâche
    - Ouvrir une tâche
- Tâche
    - Déplacer une tâche
    - Ouvrir tâche pour modification
    - Administrer les dates (début et fin)
    - Administrer les étiquettes (tags)
    - Gérer les membres
    - Supprimer une tâche
    - Modifier le titre
    - Modifier description
    - Administrer les dépendances
    - Afficher sous-tâches

Itérations

Itération 1 : Structure de base

- Liste : Mathéo + Enzo
- Tache : Stéphane + Enzo
- Page : Grégoire + Enzo

Pour cette première itération, nous avons réalisé la mise en place de la structure des données du projet. Il était aussi nécessaire de faire l'implémentation des méthodes de base pour la manipulation des données. Cela permettait l'ajout, la suppression et la modification des données.

Itération 2 : Structure visuelle et données

- Sauvegarde : Grégoire
    - Sérialisation
    - Chargement
- Tâche : Enzo & Stéphane
    - Assigner un utilisateur
    - Assigner une étiquette
    - Assigner une priorité
    - Ajout d'une sous-tâche
- MVC – Fabrique : Mathéo
    - Afficher une page et ses listes de tâches sous la forme de colonnes (Tableau)

Pour cette deuxième itération, nous avons mis en place d'une classe de sauvegarde pour conserver la page après avoir quitté l'application. L'ajout des attributs complexes de la tâche (étiquette, priorité et utilisateur) s'est aussi réalisé lors de cette itération.

La manière de représenter les tâches a également été revue pour passer à une classe abstraite avec deux implémentations (tâche et sous-tâche). Désormais, les données sont également des classes implémentant Parent et Enfant (sauf Page qui n'est pas un enfant) permettant alors de les parcourir.

Finalement, le MVC fait son apparition avec l'ajout des premières vues, principalement des vues permettant la gestion d'autres vues : les vues principales (interface VuePrincipale). Celle-ci sont créables en passant par une fabrique qui laissera le choix de développer de nouveaux types d'affichage aisément dans le futur.

Itération 3 : Affichage

- Diagramme de scène : Enzo
- Vue de la page : Enzo
- Vue de la liste : Grégoire et Mathéo
- Vue de la tâche : Stéphane
- Introduction vue de la tâche étendue (pour modification) : Stéphane
- Enlever les données en attribut des vues : Grégoire
- Commencer l'ajout des interactions (drag and drop de tâches et de liste) : Mathéo

Dans cette itération, le développement visuel a débuté. Nous mettons en place des diagrammes de scène pour avoir une idée de comment gérer et interpréter l'affichage. La vue en tableau est donc intégrée en totalité, du moins pour le design basique, les tâches n'ont, par exemple, pas toutes les données qui sont affichées pour le moment. L'affichage complet d'une tâche est introduit sans avoir de visuel propre. De plus, nous avons également nettoyé le code, notamment le retrait des données en attribut dans les vues. En remplacement, des méthodes pour récupérer la localisation des tâches et listes de tâches sont implémentées permettant ainsi grâce à une symétrie avec les données de pouvoir retrouver la bonne tâche par rapport à une "VueTache".

Le "drag and drop" est débuté pour les tâches mais n'est pas encore fonctionnel. En revanche, il fonctionne pour les listes et permet donc de les déplacer horizontalement entre elles.

Itération 4 : Affichage

- Vues de la page Gantt : Stéphane
- Vues de la tâche étendue (pour modification) : Enzo
- Finaliser vue de la liste et de la tâche : Mathéo
- Ajout des interactions (drag and drop de tâches) : Mathéo
- Design de toute l'application : Grégoire

Lors de cette itération, nous avons intégré l'affichage en Gantt des tâches. Ce dernier est actualisé lors de la modification de tâche dans les autres affichages. De plus, nous avons repris la tâche en affichage complet et l'avons réellement implémentée avec la plupart des fonctionnalités. Pour la vue de la page en tableau, celle-ci se dote du drag and drop de tâche (pas des sous-tâches) ainsi que la complétion des tâches pour afficher le contenu en entier.

L'application obtient un tout nouveau design bien plus agréable et intuitif.

Itération 5 : Affichage

- Vues Tableur : Grégoire
- Vues de la page Gantt : Stéphane
- Restructuration des vues (association des contrôleurs aux vues dans la fabrique) : Stéphane
- Optimisation du code : Stéphane
- Vues de la tâche étendue (pour modification) : Enzo
- Affichage des sous-tâches dans la tâche : Mathéo
- Ajout des interactions (drag and drop de tâches) : Mathéo

Pour cette itération, la vue en tableur est réalisée et a demandé moins de travail que celle en tableau étant donné que le plus dur est de mettre en place le système d'implémentation avec la fabrique et les "VuePrincipales". Ici, il a suffi de reprendre le pattern déjà mis en place avec un nouveau design.

La Fabrique est, par ailleurs, maintenant vraiment utile suite au déplacement de la création des contrôleurs dans celui-ci.

Finalement, la vue en tableau se dote de l'affichage des sous-tâches à l'infini ainsi que d'une refonte de la manière du fonctionnement du drag and drop. Cela permet une intégration plus fluide et optimisée en ne mettant à jour les données qu'une fois (lors de la fin du drag and drop). La prévisualisation n'est donc plus qu'un élément visuel et n'a pas d'incidence sur les données.

Itération 6 : Affichage

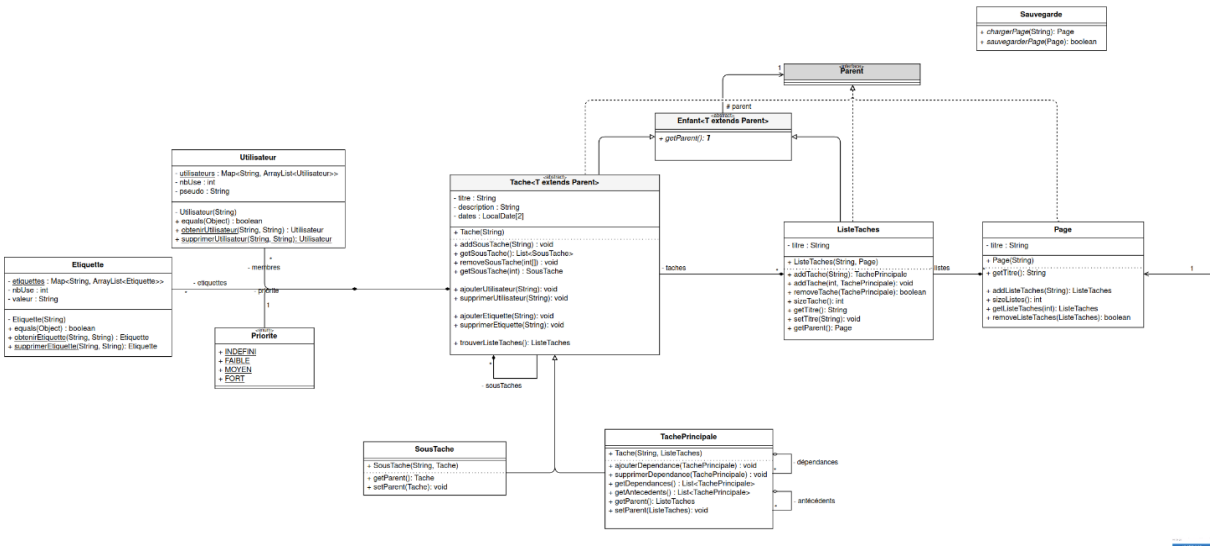
- Vues de la page Gantt : Stéphane
- Optimisation du code : Stéphane, Grégoire et Mathéo
- Préparer soutenance : Stéphane et Mathéo
- Vues de la tâche étendue (pour modification) : Enzo
- Terminer le système de sauvegarde : Grégoire
- Ajout des interactions (drag and drop de tâches) : Mathéo

L'affichage en Gantt est désormais fini et respecte les dépendances. Elles sont, par ailleurs, représentées par une flèche. Le code est revu entièrement pour intégrer les méthodes de recherche de données correspondantes à la vue rendant le code plus lisible, propre et surtout plus facilement extensible à de nouvelles fonctionnalités.

Le drag and drop se termine, intégrant alors le déplacement de sous-tâche. De plus, le déplacement d'une liste de tâche est corrigé (ne fonctionnant plus depuis une itération).

Nous préparons également la soutenance ayant lieu après l'itération.

Diagramme de classe final

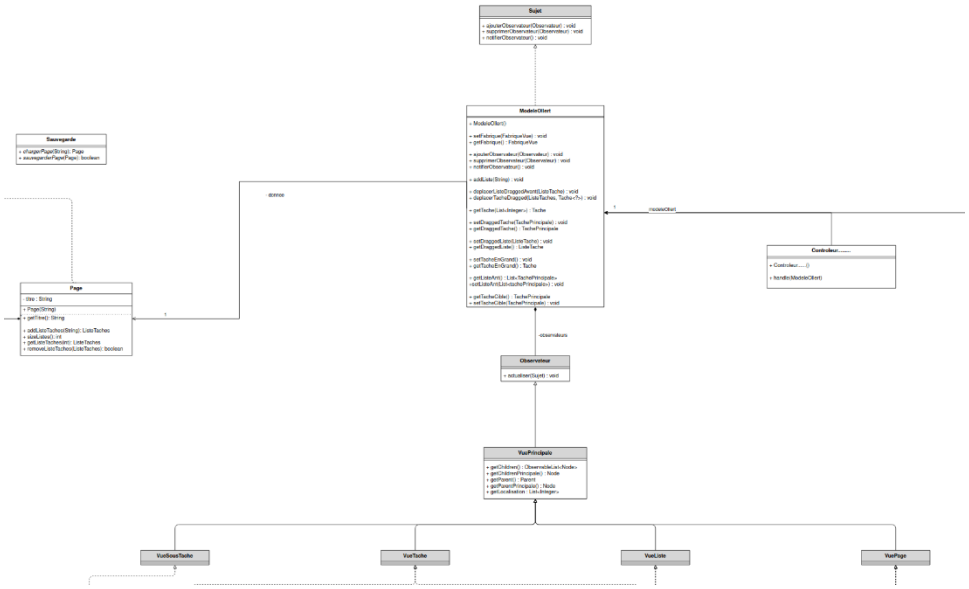


Package Ollert

Différences :

Une tâche peut avoir des sous-tâches mais une sous-tâche ne peut pas avoir de dépendances avec la nouvelle conception imaginée lors de l'itération 2.

De plus, le parcours d'un enfant vers un parent et inversement est rendu beaucoup plus simple avec l'utilisation de types génériques. En effet, le getParent() renvoit la bonne classe et ne demande pas de faire un cast à chaque fois qu'on utilise la méthode (très pratique, notamment pour les boucles !).

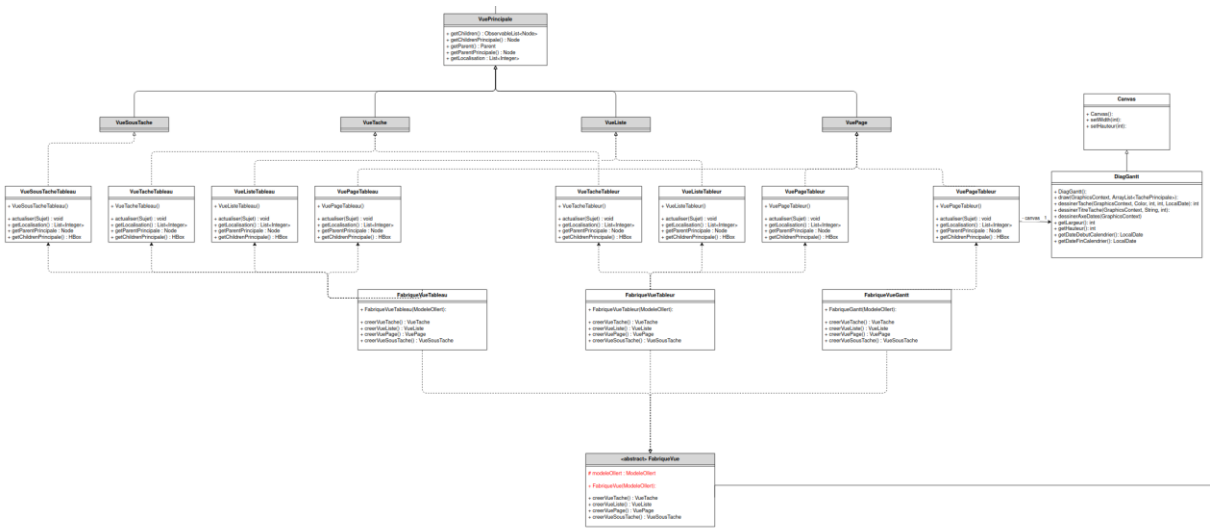


Modèle MVC

Différence :

Durant la conception, nous avions prévu d'utiliser un MVC. Donc un patron avec un modèle, des vues, et des contrôleurs. Cette conception ayant fait ses preuves dans le domaine, nous n'avons pas dû modifier le MVC de la conception.

Malheureusement, nous n'avons pas eu le temps mais nous avions pensé à un Dispatcher pour actualiser uniquement les observateurs dont les données ont changé et non pas recharger toutes les vues à chaque changement/modification.



Patron Fabrique et stratégie

Différences :

L'utilisation du patron Fabrique n'a pas était remis en cause. Ainsi, les seules modifications dans le patron ont été essentiellement des ajouts et de l'optimisation de code. A partir de la 5ème itération, nous avons décidé de faire passer la FabriqueVue qui était une interface en une classe abstraite qui possède le modèle en attribut pour éviter de passer le modèle à chaque appel de méthode.

Patrons de conception et d’architecture

MVC

Partie centrale du projet, le MVC nous permet d’articuler les différents éléments visuels de Ollert. Nous avons pris la liberté de modifier légèrement le principe de base qui est d’avoir une vue par visuel, en créant des vues principales, c’est-à-dire que, par exemples, les tâches, ou encore les listes de taches possèderont eux-mêmes d’autres vues. Cela nous permet de parcourir plus rapidement les vues et surtout de pouvoir garder une symétrie avec les données. Mis à part cela, nous conservons les contrôleurs classiques et essayons d’éviter les duplications inutiles entre les différents affichages notamment avec le contrôleur d’affichage d’une tâche en grand qui est unique peu importe si la tâche cliquée est une VueTacheTableur, une VueTacheTableau ou encore une VueSousTacheTableau.

Fabrique abstraite et stratégie

Afin de gérer plus facilement la création de vues, nous avons implémenté le patron Fabrique abstraite au projet. D’une part pour avoir un Main plus simple mais surtout pour initialiser les contrôleurs et les vues internes des vues principales en son sein. Le patron Fabrique est utilisé en complément du patron stratégie, en effet le modèle possède un attribut fabrique lui disant comment gérer son contenu. Grâce à cela, le changement d’affichage demande un simple changement de fabrique et une actualisation du contenu. De plus, si dans le futur un nouveau type d’affichage venait à être développé, il suffirait de reprendre le pattern imposé par le fabrique et l’affichage fonctionnerait directement.

Multiton

Les classes Etiquette et Utilisateur sont des multitons. Ainsi, lors de l'utilisation d'un de ces éléments, si une tâche utilise déjà un élément du même nom, celui-ci ne sera pas dupliqué, mais l'instance de l'objet déjà créée sera retournée. Il s'agirait plus précisément d'un multi-multiton étant donné que les utilisateurs et les étiquettes de pages différentes sont des instances distinctes. Cependant, il est à noter que l'utilisation de cette fonctionnalité n'est pas présente actuellement, car la gestion de plusieurs pages n'est pas prise en charge.

Fonctionnalités personnelles

Enzo MOUGIN : Gestion des dates par DatePicker

La fonctionnalité dont je suis le plus fier est la gestion des dates. Je n'étais pas familier avec le DatePicker avant cette SAE, c'était donc une opportunité de le découvrir. J'ai rapidement saisi son potentiel, notamment la capacité de désactiver certaines cases. Partant de cette fonctionnalité, ainsi que de la possibilité de choisir des dates dans un calendrier, j'ai commencé à coder les deux vues associées aux dates (début et fin).

Il était maintenant nécessaire de rendre les bonnes cases non sélectionnables. Pour ce faire, il fallait prendre en compte toutes les formes de dépendances qu'une tâche peut avoir. Une tâche principale (qui n'est pas une sous-tâche) peut dépendre d'autres tâches (antécédents), avoir des tâches qui dépendent d'elle (dépendances) et des sous-tâches.

Exemple de la VueDateFin :

Pour les dépendances, on récupère la date minimale de début des tâches (boucle for) puis on désactive toutes les cases après :

```
for (TachePrincipale tp1 : ((TachePrincipale) tache).getDependances()) {
    if ((tp1.getDateDebut() != null) && (tp1.getDateDebut().isBefore(dateMin))) {
        dateMin = tp1.getDateDebut();
    }
}
if (item.isAfter(dateMin)) {
    setDisable(true);
    setStyle("-fx-background-color: #ffc0cb;");
}
```

Pour les antécédents, on récupère la date maximale de fin des tâches (boucle for) puis on vérifie si la tache possède une date de début et on désactive toutes les cases avant la date de début de la tâche et elle-même puis toutes les cases avant la date maximale des de fin des tâches :

```
for (TachePrincipale tp2 : ((TachePrincipale) tache).getAntecedents()) {
    if ((tp2.getDateFin() != null) && (tp2.getDateFin().isAfter(dateMax))) {
        dateMax = tp2.getDateFin();
    }
}
if (tache.getDateDebut() != null) {
    if (item.isEqual(tache.getDateDebut()) || (item.isBefore(tache.getDateDebut()) || (item.isBefore(dateMax)))) {
        setDisable(true);
        setStyle("-fx-background-color: #ffc0cb;");
    }
}
```

Pour les sous-tâches, on récupère la date de début minimale et la date de début maximale et on désactive les dates entre les deux dates :

```
for (SousTache st : ((TachePrincipale) tache).getSousTaches()) {
    if (st.getDateDebut() != null && st.getDateDebut().isBefore(dateMin)) {
        dateMin = st.getDateDebut();
    }
    if (st.getDateFin() != null && st.getDateFin().isAfter(dateMax)) {
        dateMax = st.getDateFin();
    }
}
```

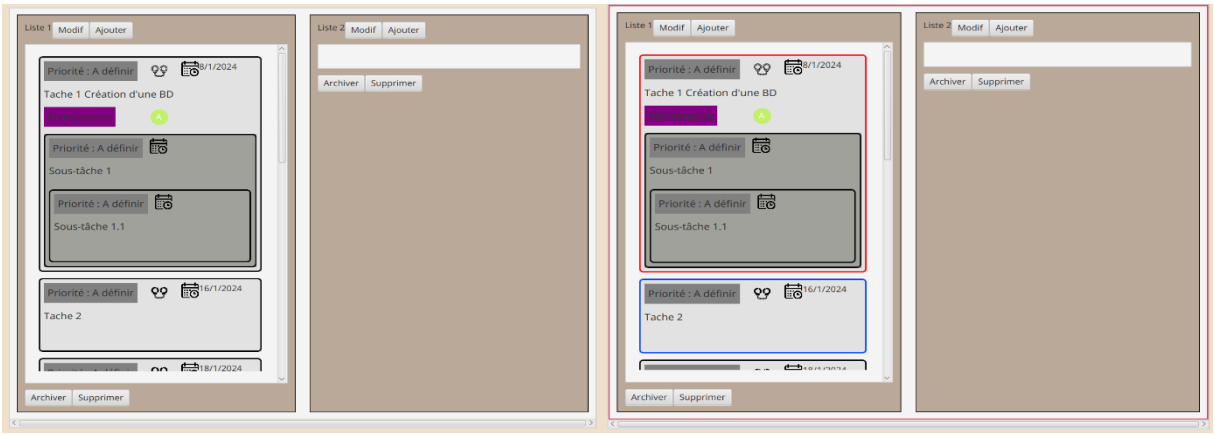
Si la tâche est une sous-tâche, on suit le même processus pour ses sous-tâches à elle. En revanche, on teste aussi les dates par rapport aux dates du parent. Les dates avant la date de début de la tâche mère et les dates après la date de fin sont désactivées :

```
for (TachePrincipale tp1 : ((TachePrincipale)tache).getDependances()){
    if ((tp1.getDateDebut() != null) && (tp1.getDateDebut().isBefore(dateMin))){
        dateMin = tp1.getDateDebut();
    }
}
if (item.isAfter(dateMin)){
    setDisable(true);
    setStyle("-fx-background-color: #ffc0cb;");
}
```

Grégoire HIRTZ : Gestion des dépendances

Pour la gestion des dépendances, j'avais la possibilité d'utiliser un système de menu, mais j'ai décidé de mettre en place un système visuel directement dans la vue. Ainsi, le fonctionnement de mon système de gestion des dépendances est le suivant.

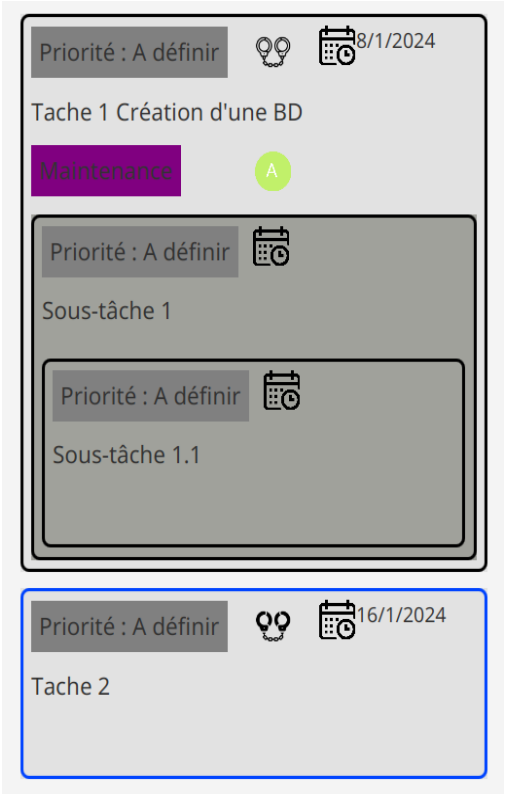
Tout d'abord, lorsque l'utilisateur clique sur le bouton de la tâche, le contour de la zone d'affichage des listes passe en rouge, le drag and drop est désactivé, tout comme l'ouverture de la tâche étendue. De plus, la tâche sélectionnée obtient un contour bleu.



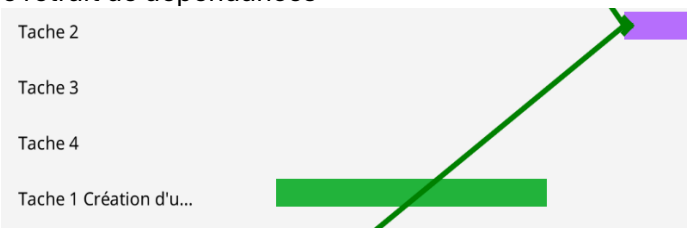
État normal Etat en mode édition

Dans le mode, les antécédents de la tâche sont affichés avec un contour rouge. Il suffit à l'utilisateur de cliquer avec sa souris sur une tâche. Si la tâche dispose d'une date et que les conditions d'ajout sont remplies, alors la tâche est ajoutée. De même, dans l'autre sens, la tâche peut être supprimée en effectuant la même opération.

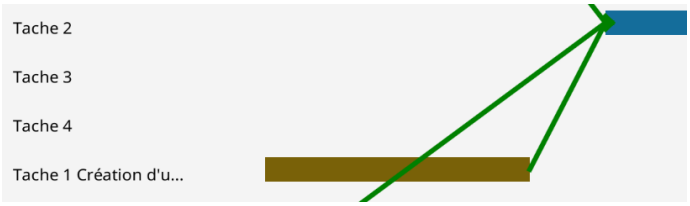
Exemple de retrait de dépendances



Etat après avoir cliqué sur ‘Tache 1’



Gantt avant modification



Gantt après avoir cliqué sur ‘Tache 1’

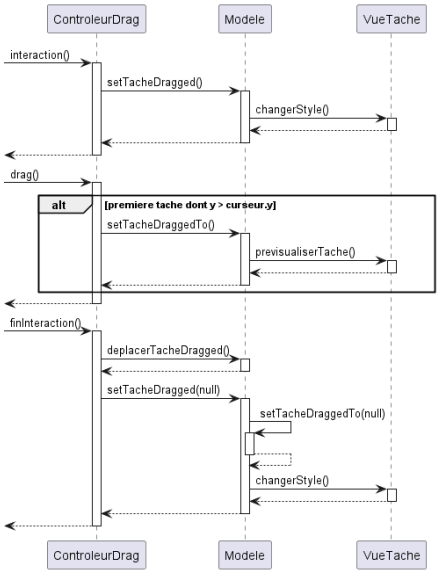
La 'Tâche 2', qui avait une date de fin ultérieure à celle de la 'Tâche 1', dépendait initialement de la 'Tâche 1'. Après un clic, l'encadrement rouge a disparu. Comme visible dans le diagramme de Gantt ci-dessus, la 'Tâche 2' n'a plus la 'Tâche 1' comme antécédent.

Pour conclure, l'utilisateur peut quitter le mode d'édition en recliquant sur le bouton du mode édition une deuxième fois, revenant ainsi au mode d'affichage normal.

Mathéo PEDRON : Drag and Drop

Débutée lors de l'itération 3, la fonctionnalité de drag and drop avait pour objectif de permettre le glisser-déposer des tâches entre différentes listes de tâches, tout en autorisant également le déplacement d'une tâche vers une autre tâche. De plus, la possibilité de déplacer les listes de tâches a également été implémentée.

La fonctionnalité de déplacement de liste a été rapidement réalisée lors de l'itération 3, suivie par le drag and drop de tâche dans l'itération 4, et celui des sous-tâches lors des itérations 5 et 6. Cependant, il est important de noter que le drag and drop de tâche sera entièrement refait lors de l'itération 5. Cette refonte a été entreprise pour faciliter l'extension vers d'autres systèmes, ainsi que pour adopter une méthode plus optimisée qui ne modifie pas les données à chaque déplacement du curseur, mais plutôt à chaque relâchement du drag and drop.

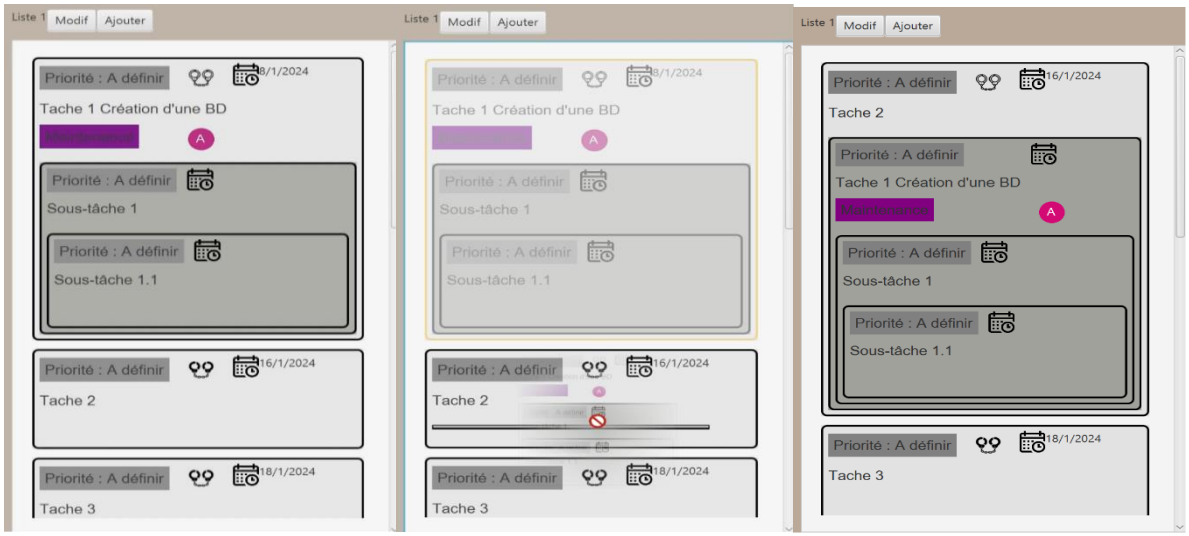


Comme le présente le diagramme de scène ci-contre, l'objectif était de diviser l'activation d'un drag and drop en trois étapes.

Premièrement, l'appui sur une tâche, ce qui la définit comme tâche déplacée et ainsi lui modifie son style.

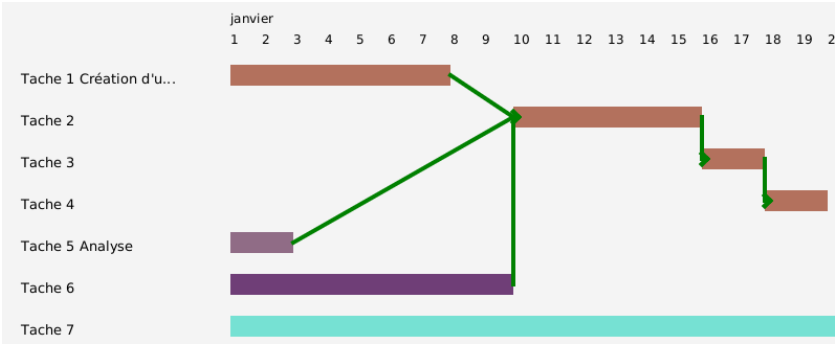
Deuxièmement, lors du déplacement de la tâche, sa position est calculée en temps réel. Ainsi, la première tâche dont la coordonnée Y du bord inférieur est supérieure à celle du curseur est définie comme le nouvel emplacement de la tâche déplacée. Pour ce faire, les données ne sont pas modifiées directement. La Vue Liste récupère la position de déplacement de la tâche et, au lieu d'afficher la Vue Tâche initialement prévue, elle la décale et affiche un séparateur indiquant la position de la tâche.

Troisièmement et enfin, lors du relâchement de la tâche, nous informons le modèle de déplacer la tâche à l'endroit où le séparateur se trouve. Ensuite, la tâche sélectionnée est remise dans son style normal. Lors du déplacement, il a été nécessaire de prendre en compte les différentes dates des tâches, empêchant une tâche de devenir une sous-tâche si elle possède des dates en dehors de celles de sa future tâche parente.

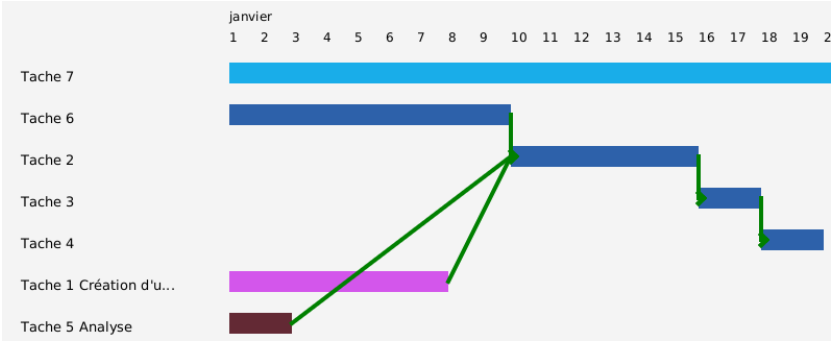


Stéphane LOPPINET : Gantt

La génération du diagramme de Gantt fut complexe, car il fallait faire attention à chaque cas possible. Initialement, j'avais pris la décision de faire le diagramme de Gantt le plus tôt possible (ASAP) pour que l'utilisateur puisse avoir un calendrier de ses tâches le plus optimisé. Pour cela, je calculais à chaque fois la durée des tâches. Néanmoins, il est en fait plus pertinent de montrer à l'utilisateur les espaces qu'il a laissés dans le calendrier au cas où il voudrait ajouter une tâche à cet endroit. Je me suis donc finalement établi l'échelle temporelle : une journée = 30 pixels en largeur. Sur le principe, il faut donc commencer par les tâches sans antécédents et suivre les dépendances. Aussi, je devais simplement dessiner les tâches sur le Canvas selon sa date de début et de fin. Néanmoins, je voulais aussi que la lecture du diagramme soit très simple pour l'utilisateur. Je voulais donc faire en sorte que les tâches dépendantes qui commencent le plus tôt et terminent le plus tard soient affichées ensemble pour éviter cela par exemple :



On voit ci-dessus que la tâche 6 est plus longue que la tâche 1 et qu'il serait plus logique que les deux tâches soient interchangées. De plus, la couleur unique avait pour but de montrer le chemin "critique" pour chaque groupe de tâches indépendantes les unes des autres. Ainsi, j'ajoute donc un Comparateur qui s'occupe de trier les tâches pour avoir les tâches sans antécédents qui commencent le plus tôt et qui terminent le plus tard. Cela donne donc :

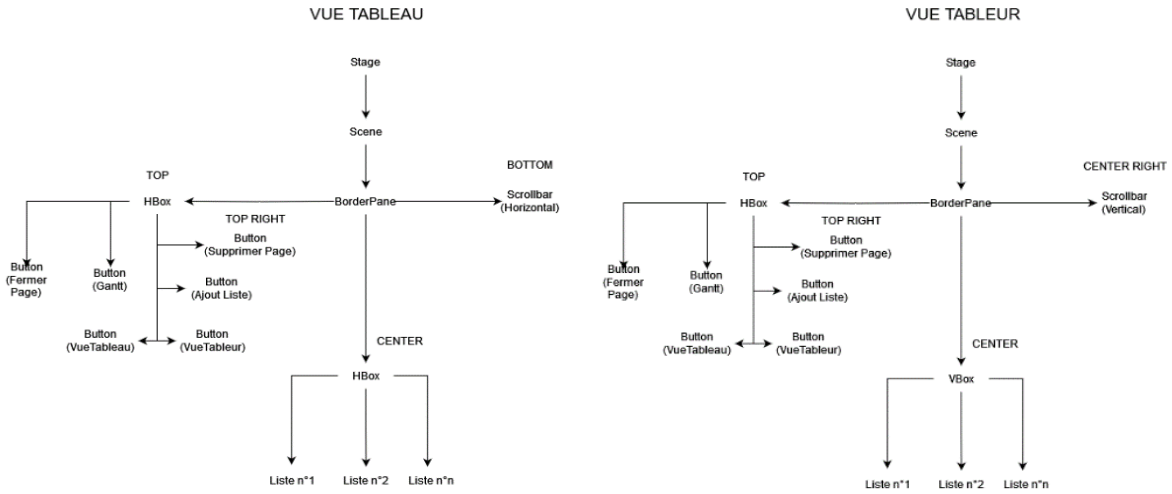


De plus, je souhaite ajouter des flèches pour indiquer les dépendances entre les tâches. Je stocke donc dans une HashMap la tâche et ses coordonnées (x, y), représentant le milieu de l'arête gauche de chaque tâche (la destination d'une flèche).

En conclusion, je suis très satisfait de mon diagramme de Gantt. D'une part, la dimension du Canvas est dynamique, ajustée en fonction du nombre de tâches et de la durée du calendrier, ce qui permet de défiler le long du diagramme grâce à un ScrollPane. De plus, mon code respecte le principe SOLID en suivant la responsabilité unique, car j'ai segmenté la création de mon diagramme (dessinerTache, dessinerFleche, dessinerAxe).

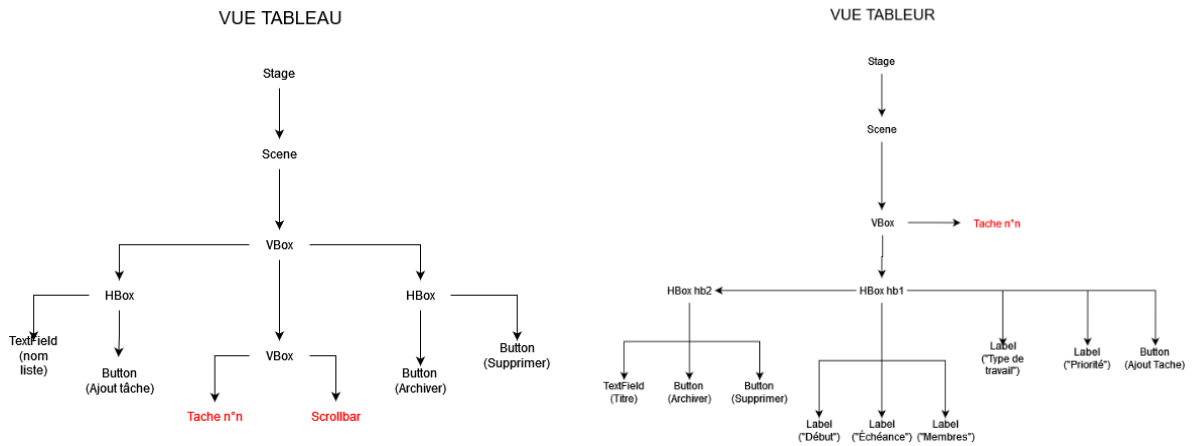
Graphes de scène

En premier lieu, il fallait définir le graphe de scène de la page entière. Effectivement, il s'agit de la première interface à laquelle l'utilisateur va être confronté, il était donc nécessaire de penser en premier lieu à ce graphe. Voici ce qui en est ressorti :



La différence principale, pour la page entière, est que la HBox devient une VBox.

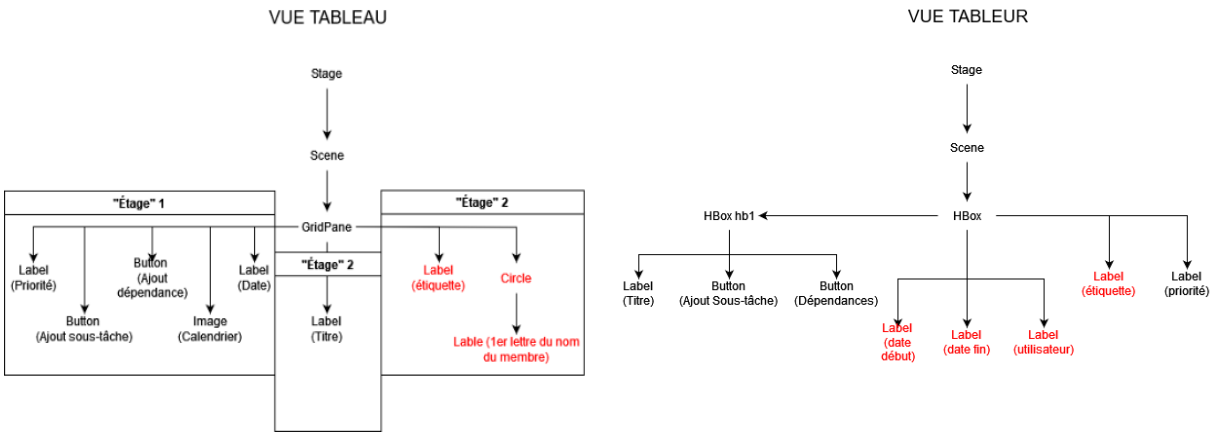
Pour la suite, nous devons décrire les vues listes présentent dans les HBox et VBox. Comme pour les pages, nous devons les représenter sous forme de Tableau et de Tableur et ainsi prévoir deux graphes de scène différents que voici :



Les éléments en rouge sont des éléments susceptibles de ne pas être affichés s'ils ne sont pas définis ou s'ils n'existent pas (ici, pas de tâche ou pas assez de tâches pour avoir la nécessité d'afficher une Scrollbar).

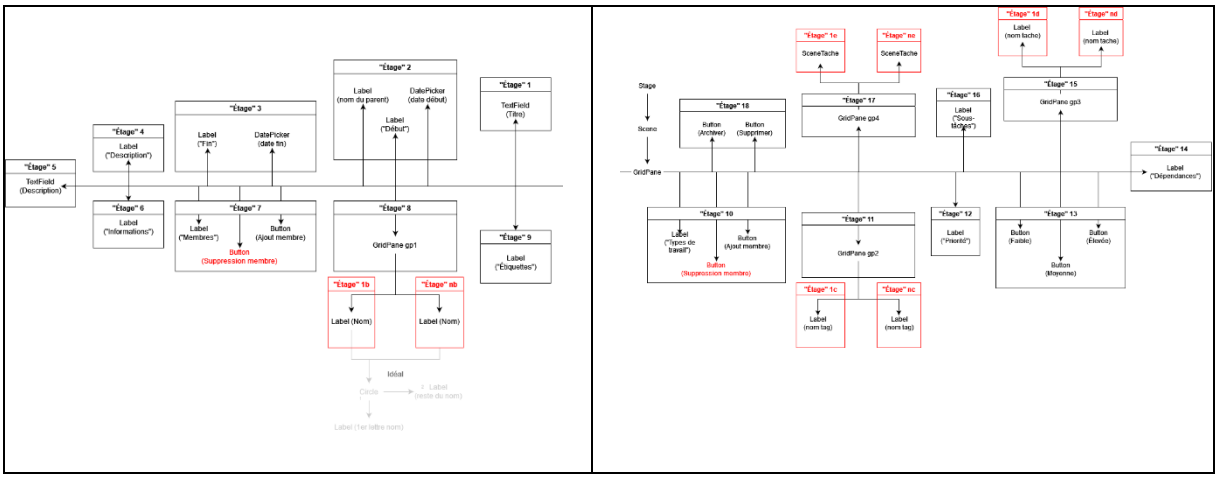
Cette fois-ci, les deux vues sont clairement distinctes l'une de l'autre. Nous avons donc une vue Tableau plus simpliste, comprenant une VBox avec un en-tête (HBox avec titre...), un corps (VBox avec toutes les tâches...) et un pied de page (HBox avec les boutons de suppression). La vue Tableur, quant à elle, est nettement plus complexe. À l'image d'un véritable tableur, il était nécessaire d'attribuer un nom aux colonnes. Nous avons donc envisagé d'utiliser une VBox pour regrouper la barre des "titres" et les tâches. La barre des titres est représentée sous forme d'une HBox contenant une HBox principale avec le titre de la liste et ses boutons, et le reste est constitué de labels pour les titres des colonnes.

Ensuite, nous avons donc à imaginer une façon de représenter nos tâches dans ces deux différentes listes. La forme Tableau étant plus esthétique et la Tableur plus brute pour mieux voir les données. Nous avons donc réalisé les deux graphes de scène suivants :



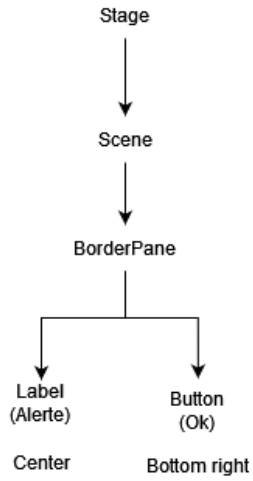
Encore une fois, les deux vues ne sont pas similaires du fait de l'approche que nous avons adoptée pour chacune d'entre elles. Ainsi, la vue Tableau possède un GridPane qui offre la possibilité de placer les éléments avec une plus grande facilité et précision. On remarque notamment que le titre doit être centré, avec les boutons positionnés au-dessus. La vue Tableur se compose d'une HBox principale et d'une sous-HBox représentant l'étage 1 du GridPane de l'autre vue. Pour le reste, il s'agit des labels correspondant aux données des titres des colonnes de la vue liste Tableur.

Après cela, nous avons abordé la dernière vue majeure, la vue interface tâche. Celle-ci s'ouvre lors d'un clic sur une tâche, permettant l'accès aux données non affichées (uniquement si on vient de la vue Tableau) et la possibilité de les modifier. C'est également ici que se trouvent les options de suppression d'une tâche. Voici le résultat de notre réflexion :

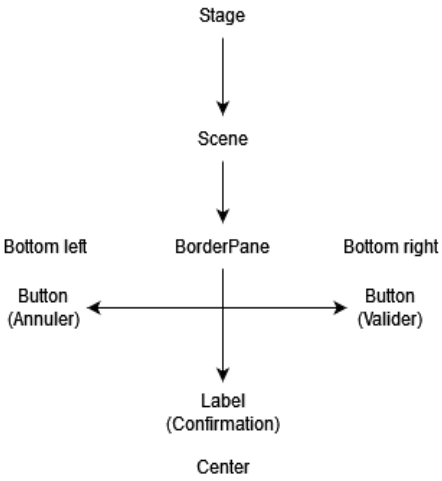


Nous avons donc opté pour la façon qui nous semblait la plus simple pour afficher autant d'informations, soit le GridPane. L'idée de ce graphe de scène est de se rapprocher le plus possible de l'idée de base du Balsamiq.

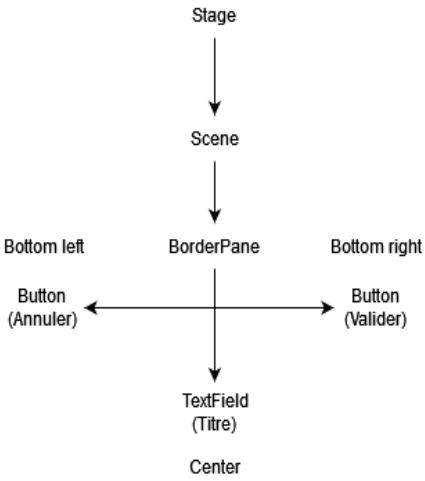
Enfin, nous avons quelques graphes de scène secondaires, notamment des pop-ups :



Graphe erreur



Valider suppression



Créer un élément

Changement de vue

Partons du principe que l'on veut passer d'un affichage Tableau à Gantt.

Dans le modèle, nous avons un attribut "fabrique" héritant de la classe FabriqueVue (FabriqueVue – Tableur, Gantt, Tableau), qui s'occupe de créer les vues (Page, ListeTaches, Tache, SousTache). Lorsque l'utilisateur veut changer le type d'affichage, il clique sur le bouton associé (dans notre exemple : le bouton Gantt), puis le contrôleur s'occupe de changer le fabrique (via un Setter) puis d'appeler la méthode creerVuePage pour avoir la nouvelle VuePage (VuePageGantt dans notre exemple) qui doit supprimer l'ancienne VuePage (VuePageTableau) et ajouter la nouvelle VuePage dans les observateurs. Enfin, il actualise les observateurs pour afficher la nouvelle VuePage, qui elle-même s'occupe de parcourir le modèle pour récupérer les listes et les tâches, afin de créer les Vues relatives (via le Fabrique toujours), puis de les afficher.

```
/**
 * Fabrique correspond au type d'affichage de la page
 */
3 usages
private FabriqueVue fabrique;
```