TOULOUSE INP N7

APPER Martin
BOUDJEMA Mehdi
COMBAL Nicolas
FURRIEL Enzo
PERSONNAZ Alban
TOUCHAIS Baptiste

RAPPORT DE SYNTHÈSE

Pinte

Projet Long - Conception et Programmation Orientée objet

Table des matières

Introduction

Pour notre projet long, nous devons créer une application en langage java qui utilise SWING pour l'interface graphique et qui applique le modèle MVC. L'application que nous avons décidé de créer se nomme Pinte et est une application de dessin numérique qui s'adresse à tous les créateurs d'images numériques.

L'application propose plusieurs fonctionnalités qui permettent un usage avec une interface graphique . L'utilisateur peut ainsi utiliser sa souris pour dessiner des formes de différentes couleurs, remplir des zones de couleurs et a la possibilité d'effacer ces dessins sur une toile.

L'application se démarque des applications de dessins numériques par la possibilité de dessiner des formes au format SVG à partir d'un terminal, ainsi que par la possibilité d'utiliser des outils avancés.

L'objectif du projet est d'acquérir des compétences dans la programmation orientée objet et pouvoir utiliser correctement le modèle de conception MVC avec SWING. Pour réaliser ce projet, nous avons réalisé une conception détaillée de l'application en identifiant les différentes classes nécessaires, les interactions entre elles et les fonctionnalités à implémenter.

Conception

L'objectif de l'application est de pouvoir créer des dessins numériques en utilisant des outils à travers une interface graphique et avoir la possibilité de taper des commandes pour obtenir des formes sur une toile au format SVG.

Les epics identifiés à partir de cet objectif sont :

- En tant qu'utilisateur adepte des interfaces graphiques,
  Je souhaite pouvoir dessiner sur une toile,
  Afin de réaliser des dessins numériques et des images personnalisées au format PNG ou JPEG
- En tant qu'utilisateur adepte des interfaces en ligne de commande
  Je veux pouvoir taper des commandes
  Afin de réaliser des dessins numériques et des images personnalisées au format SVG

Les fonctionnalités les plus importantes de l'application qui ont été identifiées d'après les epics sont :

Utiliser différents outils pour dessiner sur une toile comme :

- Le pinceau
- Le seau
- La pipette
- La gomme
- L'outil forme

Utiliser un terminal des commandes qui permettent d'obtenir des formes sur la toile

- Cercle
- Rectangle
- Ovale
- Polygone

Le choix qui a été retenu est de créer d'un côté une fenêtre toile permettant d'avoir accès à une boîte à outil pour interagir avec une toile et enregistrer ce travail au format PNG ou JPEG et d'un autre côté une fenêtre qui permet d'avoir un terminal pour entrer des commandes qui forment un dessin sur la toile au format SVG. Nous avons fait ce choix afin de répartir au mieux les tâches auprès de tous les membres de l'équipe.

Les deux sous systèmes principaux qui découlent des épics sont :

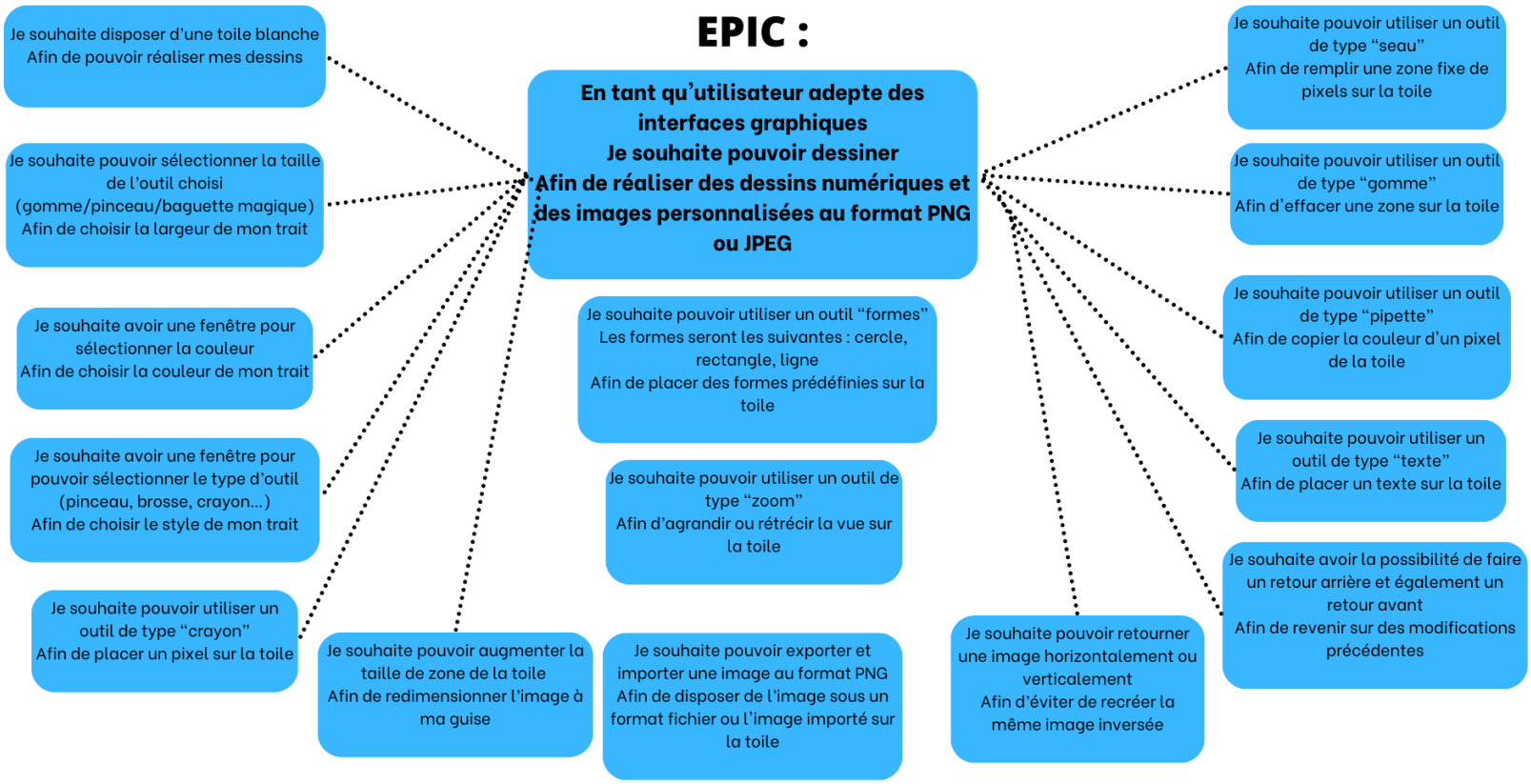
- **graphic** : fenêtre qui permet d'utiliser des outils pour interagir avec la toile et produire des images au format
- **terminalSVG** : fenêtre qui permet d'avoir le terminal pour rentrer les commandes et produire une image SVG

Conception Graphique

L'epic de cette partie est la suivante :

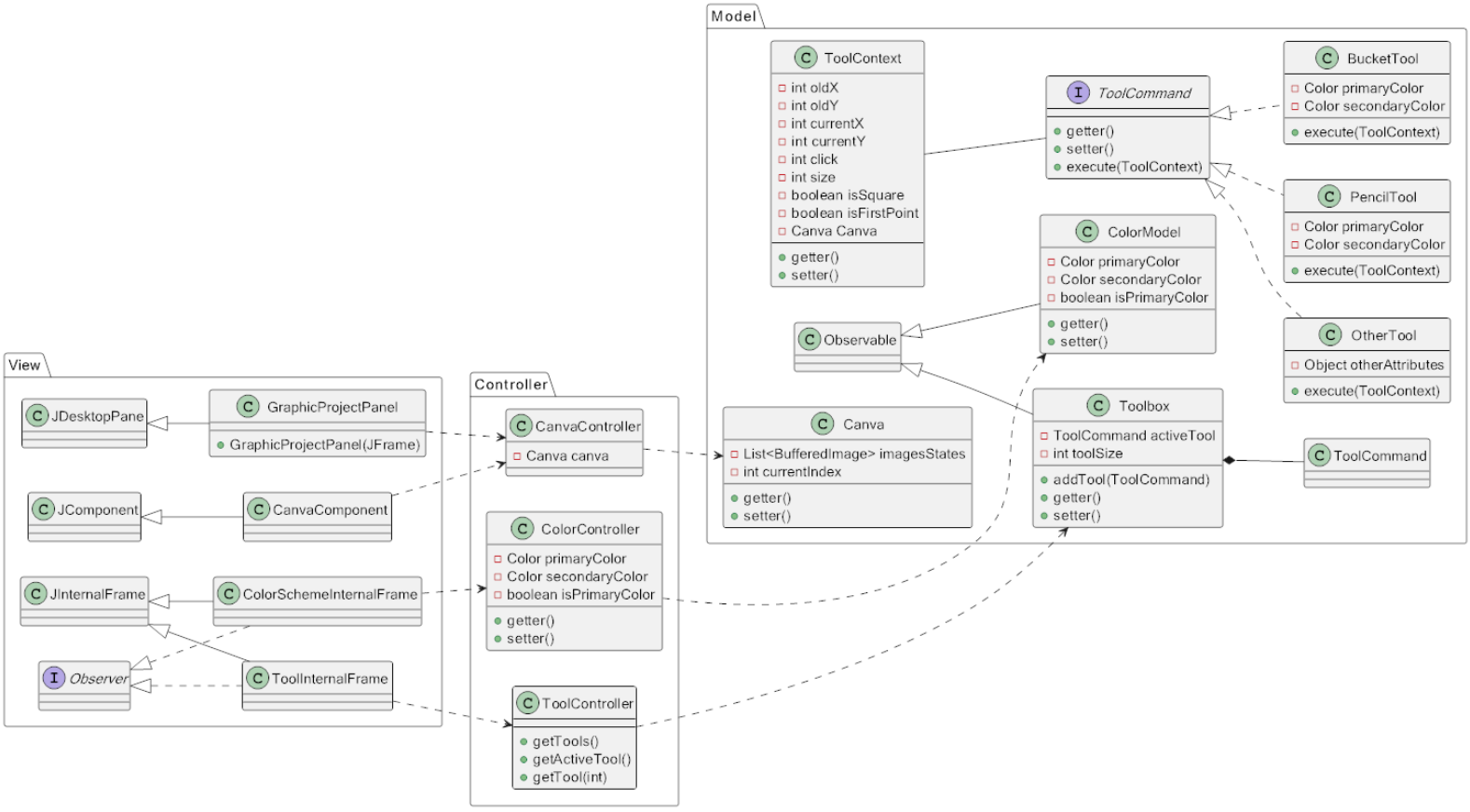
- En tant qu'utilisateur adepte des interfaces graphiques
  Je souhaite pouvoir dessiner
  Afin de réaliser des dessins numériques et des images personnalisées au format PNG ou JPEG

Voici un schéma qui représente les US identifiés d'après cette **Epic** :



Architecture : Diagramme de classe :

Voici le diagramme de classe correspondant à la **partie graphique** du projet :



Plusisieurs patrons de conceptions ont été mis en oeuvre car ils étaient adaptés à nos besoins de conception. Ainsi, pour la boîte à outils, le patron de conception **“Commande”** a été mis en place. Comme nous devons respecter le modèle **MVC** (Modèle Vue Controlleur), le patron de conception **Observe**r a été mis en place au niveau des vues et du modèle.

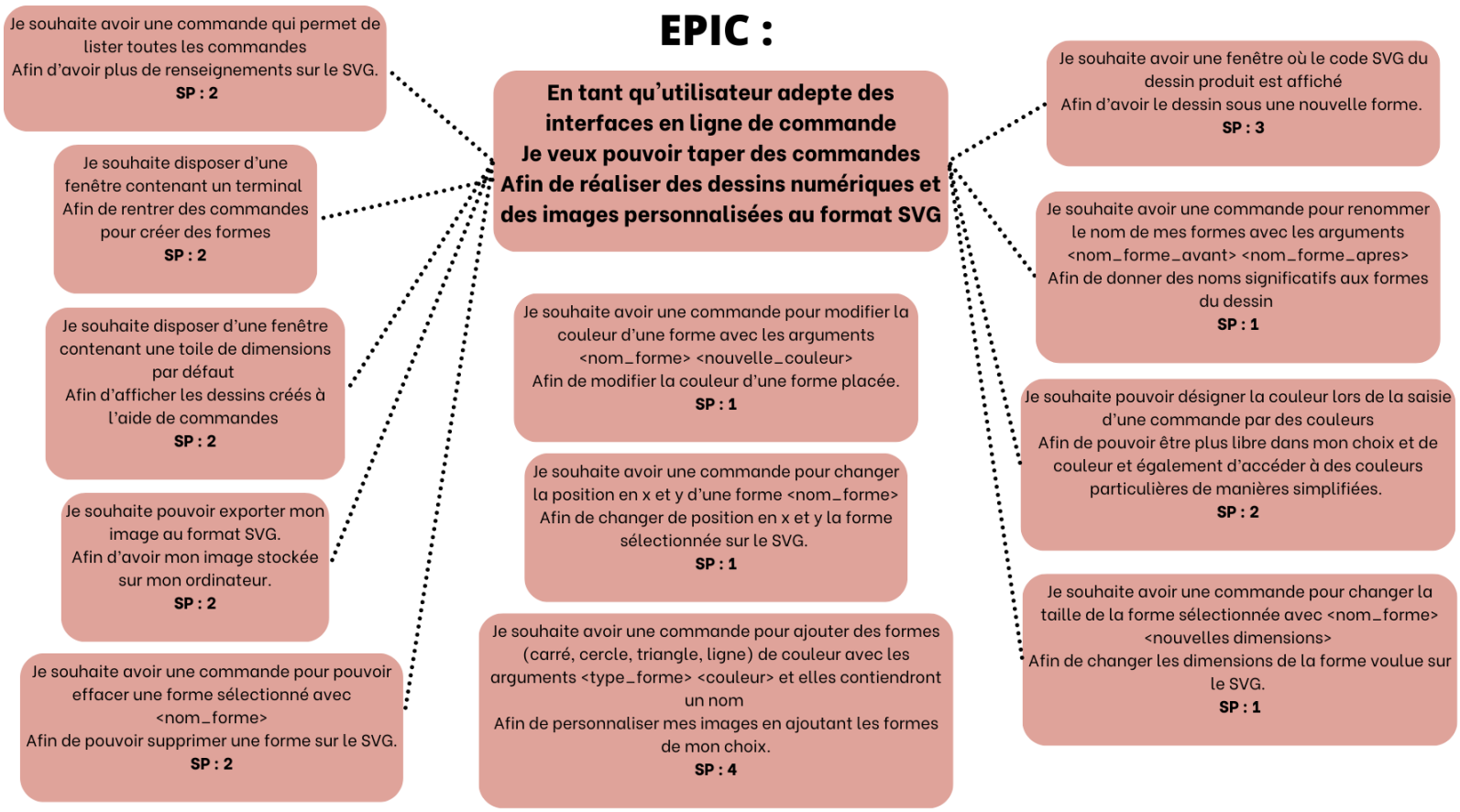
Remarque : Par soucis de clarté, tous les outils et toutes les classe ne sont pas représentés sur ce diagramme

Conception SVG

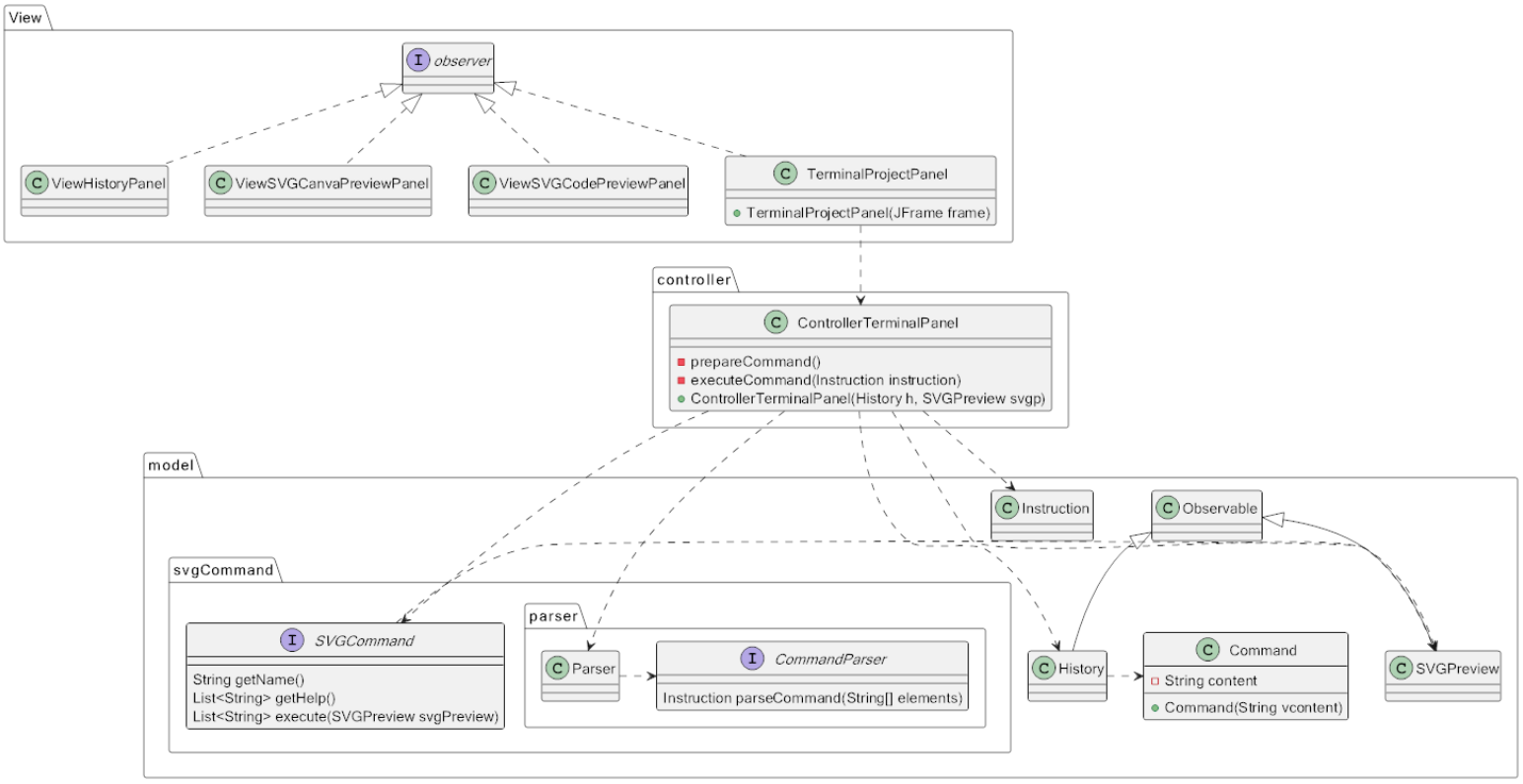
L'epic de cette partie est la suivante :

- En tant qu'utilisateur adepte des interfaces en ligne de commande
  Je veux pouvoir taper des commandes
  Afin de réaliser des dessins numériques et des images personnalisées au format SVG

Afin de réaliser cette épic, nous avons divisé en différentes User Stories (US) qui ont plus ou moins d'importance pour l'utilisateur. Voici les User stories principales que l'on a identifié :



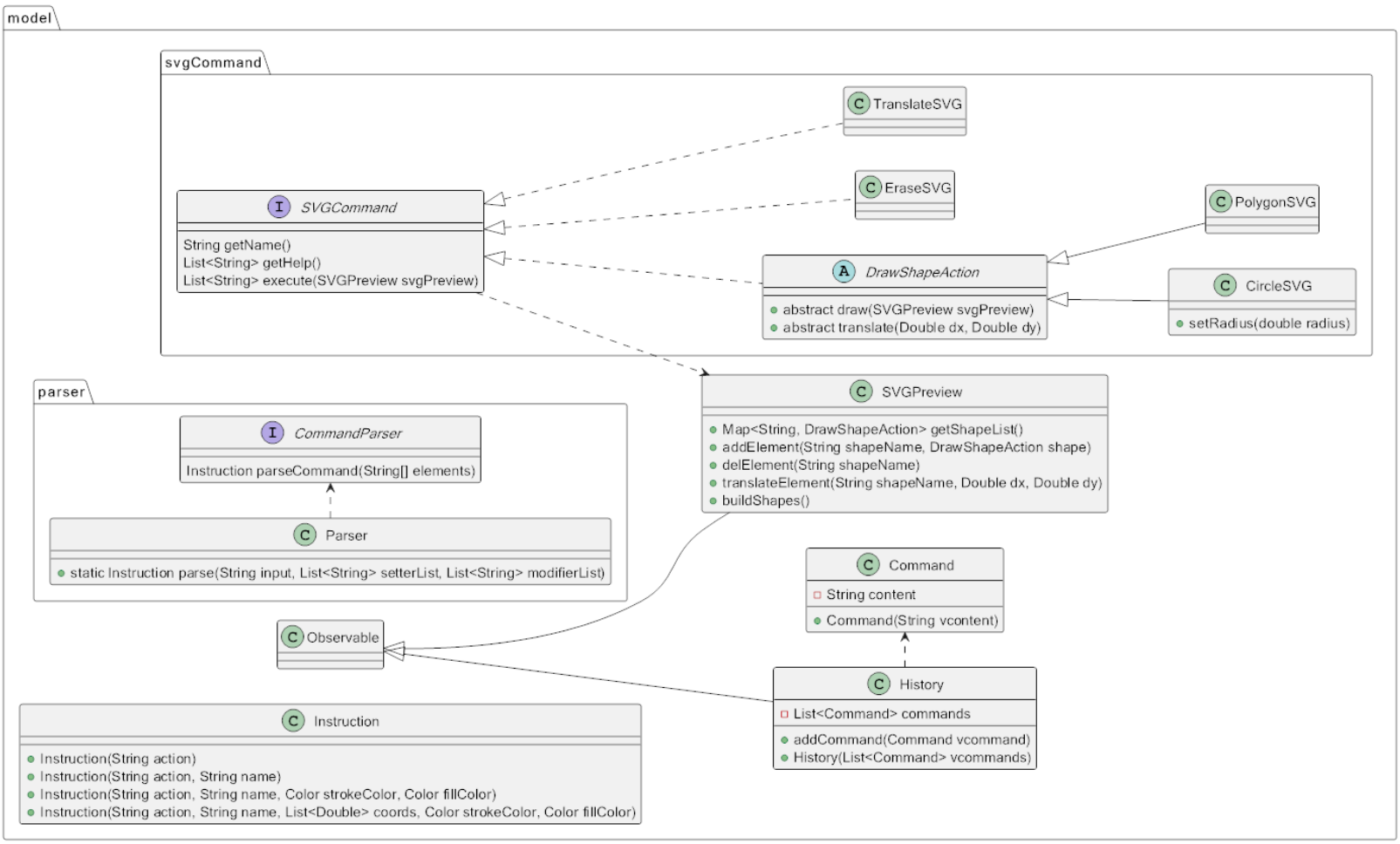
Architecture : Diagramme de classe



Voici le diagramme de classe global et simplifié de la partie **SVG** :

De la même manière que pour le package Graphic, on retrouve plusieurs patrons de conceptions comme le patron “Commande” car toutes les commandes rentrés peuvent au final implémenté ce patron ce qui sera beaucoup plus facile par la suite. Une méthode exécuter est présente dans toutes les commandes. Il existe 4 vues qui corresponden

Voici le diagramme de classe du **package model** qui a été implémenté pour la partie avec le terminal et les images au format SVG :



On peut mieux voir la partie **SVGCommande** qui sont des classes qui implémentent l'interface SVGCommande. On retrouve parmis ces commandes des commandes qui modifient les formes déjà en place comme **TranslateSVG** ou **EraseSVG** et des commandes qui créent des

Remarque : Par soucis de clarté, la plupart des attributs et méthodes ont été enlevées des diagrammes de classes précédents. Seules les éléments les plus pertinents sont représentés. Pour plus de précision sur les attributs, méthodes et constructeur, vous pouvez regarder directement dans les packages correspondants

Implémentation

Pour réaliser cette application nous avons réalisé une fenêtre principale, composée d'un panneau permettant à l'utilisateur de choisir entre un projet matriciel au format jpg ou png et un projet vectoriel en svg via l'interface en ligne de commandes. Nous allons donc par soucis de clareté séparer les explications d'implémentation en deux parties : la partie matricielle et la partie vectorielle.

Dessin matriciel

Réalisation de l'application

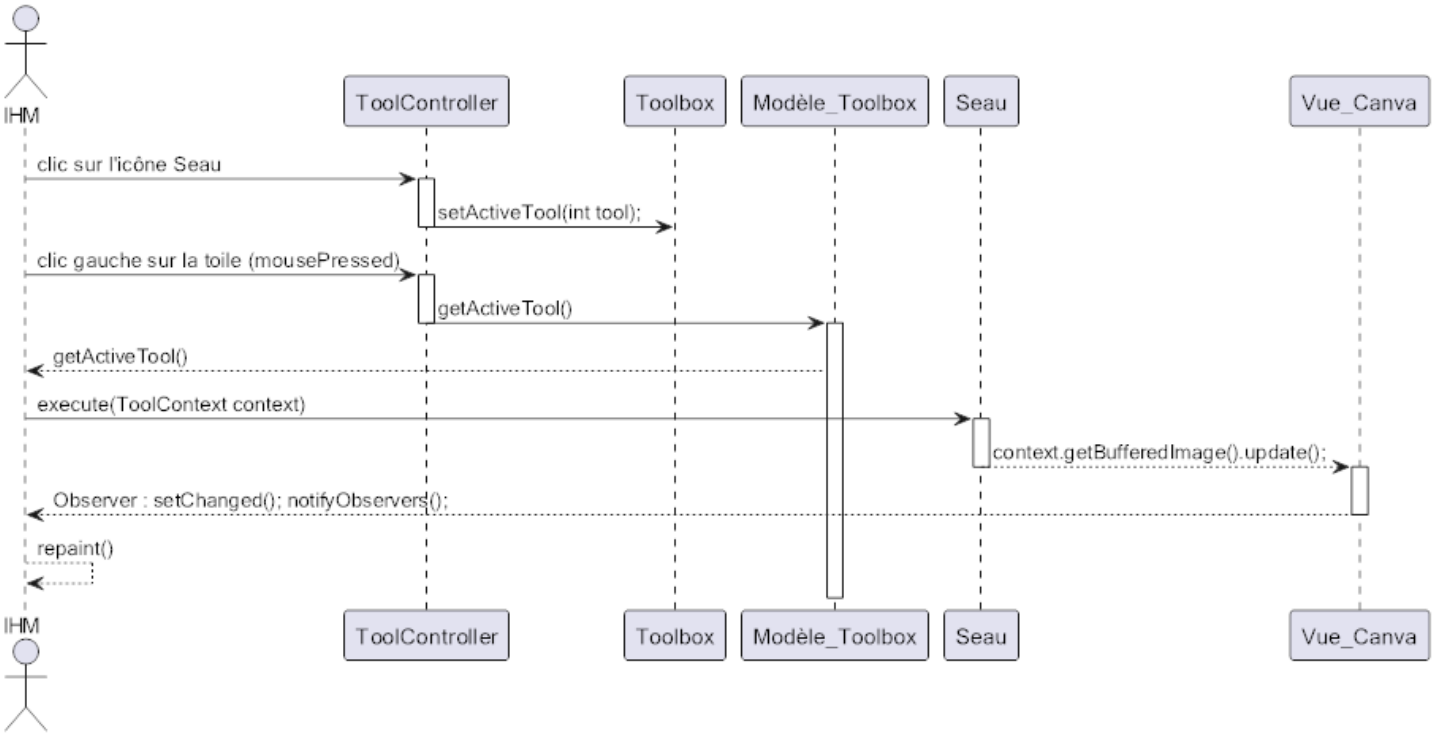
Nous avons pour la partie matricielle décidé dans un premier temps de partir sur un modèle de grille composée de JButtons. Nous avons donc initialisé un panneau avec un GridBagLayout, sur lequel nous avons appliqué des GridBagLayoutConstraints afin que les boutons soient de forme carré et bien disposés sur la toile. Nous avons développé un outil crayon, un outil gomme et un outil seau, qui fonctionnaient lorsque l'utilisateur cliquait et glissait sur la toile, ou juste cliquait pour le seau. Cependant, nous nous sommes rapidement aperçus que lors de la création d'une grande toile, la création était très longue, et glisser rapidement sur la toile faisait des pointillés plutôt que de faire une ligne distincte. De plus, l'outil seau mettait un temps très long pour remplir la toile et faisait parfois crasher l'application. Nous avons donc réfléchi à la question et nous sommes mis d'accord sur une refactoring de l'application.

Nous nous sommes donc rabattus vers une solution bien plus optimisée et adaptée pour répondre à notre problématique. En utilisant une BufferedImage et des Graphics et Graphics2D de la librairie java.awt, nous avons pu par ce refactoring réaliser plus facilement une toile de plus grande dimension et plus détaillée.

Après avoir mis en place cette nouvelle toile, nous avons dû adapter les outils que nous avions créés. C'est à ce moment que nous avons décidé d'utiliser le patron Command pour implémenter ceux-ci. En effet, nous avions prévu de réaliser de nombreux outils s'utilisant avec un clic sur la toile et nous avons donc pensé que ce patron serait adapté à l'application. Nous avons donc créé une interface ToolCommand, qui contenait des méthodes pour récupérer les attributs de l'outil (nom, image) et une méthode execute qui s'exécutera lorsque l'utilisateur cliquera avec un outil sélectionné.

Diagramme de séquence

Pour montrer les différentes intéractions entre le classe et l'implémentation du modèle Observer, voici le diagramme de séquence des intéractions lors de la sélection et l'utilisation de l'outil seau :



Difficultés rencontrées et solutions apportées

Au niveau des difficultés rencontrées nous en avons effectivement eu plusieurs. Tout d'abord, comme dis plus tôt la première difficulté rencontrée a été la taille de la toile et son optimisation. Pour palier ce problème nous avons réalisé un refactoring pour utiliser une BufferedImage et des Graphics afin de fluidifier grandement le tracé, la génération de la toile et tous les outils en général.

La deuxième difficulté rencontrée était déjà présente dans la première version de l'application. En effet, un glissement rapide avec l'outil crayon ou gomme provoquait toujours une ligne en pointillé plutôt qu'une ligne comme l'utilisateur le souhaitait. Pour régler ce problème, nous avons pris la décision d'enregistrer les coordonnées ou l'utilisateur clique, et de tracer une ligne de ce point au point courant de la souris. De ce fait une ligne est tracée au lieu de deux points plus ou moins éloignés. Après le tracé, on affecte aux anciennes coordonnées du point cliqué le point courant et on re trace une ligne de ce point au point courant.

La troisième difficulté rencontrée a été de régler la taille du crayon. En effet, l'objet Graphics2D n'offrant pas la possibilité de dessiner des lignes ou quoi que ce soit en réglant la taille, il n'était initialement pas possible d'avoir un trait avec une largeur plus grande qu'un pixel. Pour palier cette difficulté nous avons utilisé l'agorithme de Bresenham. Cet algorithme permet de calculer les coordonées de tous les points séparant deux points, et ce avec un mecanisme d'erreur pour que les points soient les plus précis possibles. Avec cet algorithme, à chaque calcul de point nous dessinons un rectangle ou un rond rempli de la taille souhaitée par l'utilisateur, ce qui permet de créer un trait de la taille souhaitée.

La quatrième et plus grande difficulté dont nous avons dû faire face a été la gestion de la possibilité d'annuler et rétablir (Ctl + Z / Ctl + Y). En effet, étant donné que nous appliquions les modifications à une BufferedImage à chaque fois que l'utilisateur faisait une action, nous ne pouvions pas enregistrer les modifications et ne pouvions donc pas revenir en arrière ou en avant. La solution trouvée a été d'utiliser une liste de BufferedImage plutôt qu'une seule BufferedImage. On enregistre chaque image a la fin d'un clic et nous disposons d'un index courant qui indique quelle image afficher.

Dessin vectoriel

Réalisation de l'application

Nous avons décidé de nous baser sur le TP de SWING réalisé avec Mme Hurault pour la partie interface graphique. En effet tout comme dans le TP nous avions besoin d'un champ de saisie et d'un historique pour les messages saisies dans le champ de texte sauf que dans notre cas il a servi pour l'historique des commandes tapées par l'utilisateur. Ensuite nous avons implanté un JPanel pour l'affichage du code SVG généré par les commandes ainsi qu'un canva affichant les formes elles aussi tapées dans le champ de saisie.

Pour la création des formes nous avons utilisé la bibliothèque Batik 1.16 qui contenait plusieurs classes utiles pour la création d'éléments SVG, comme la class SVGGraphic2D qui offrait un large choix de méthodes afin de créer des formes au format svg. Par la suite nous pouvions créer des formes SVG et obtenir le code pour les afficher sur le canva.

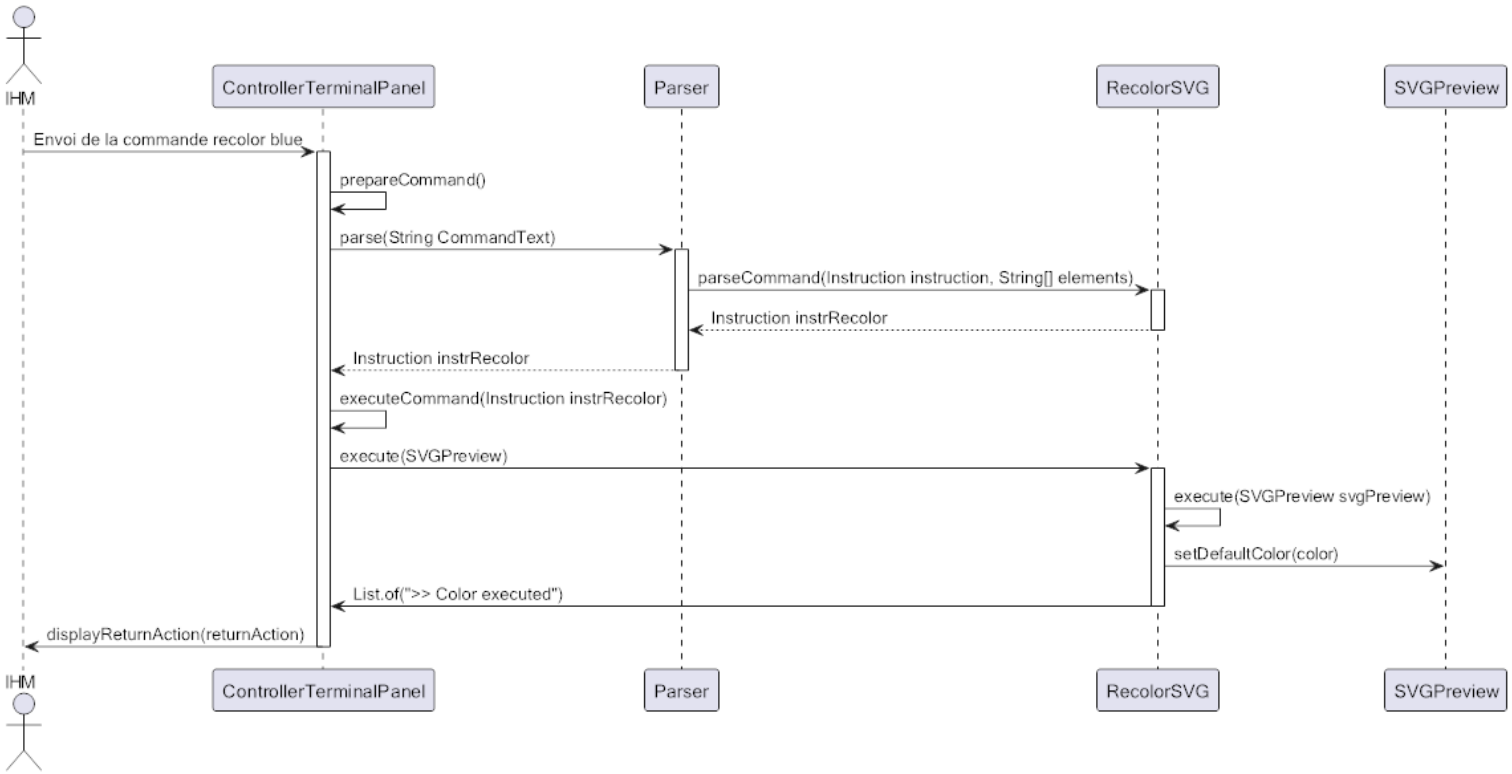
Nous avons hésité sur la librairie qui nous permmetrait d'afficher l'image SVG sur le canva entre Batik et JfreeSVG. Batik offre des solutions intéressantes pour l'affichage de code SVG que ne proposait pas Jfreesvg, qui lui ne proposait que des classes et méthode pour obtenir du code SVG sans permettre d'affichage par la suite via SWING.

Diagramme de séquence

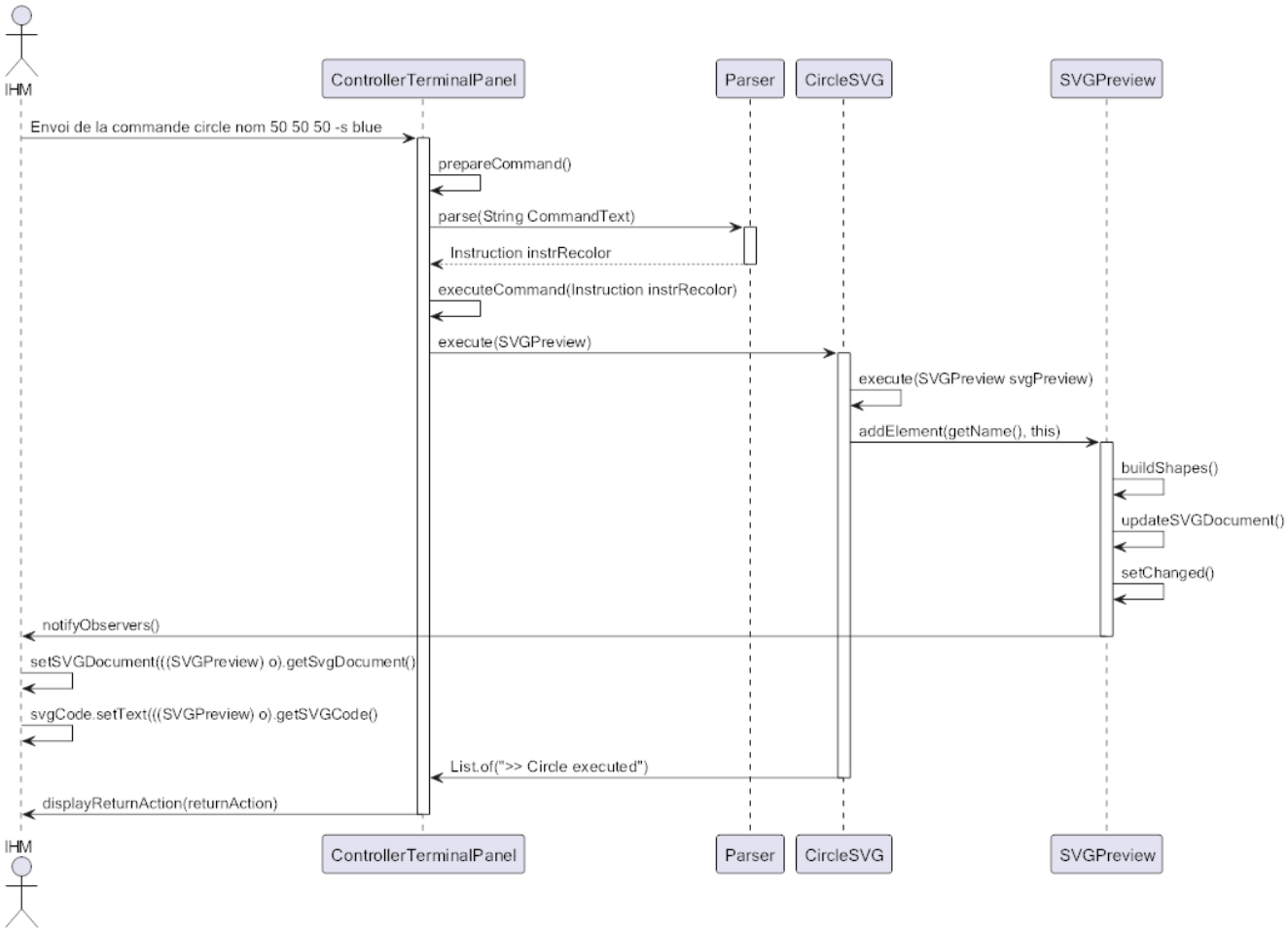
Pour montrer les différentes intéractions entre le classe et l'implémentation du modèle Observer, nous allons montrer deux diagrammes de séquences :

- Un diagramme de création de forme
- Un diagramme de modification de forme

Voici le diagramme de modification de la couleur d'une forme :



Ensuite, voici le diagramme de séquence qui montrent les intéractions entre les classes pour la création d'un cercle. On remarquera que les intéractions sont assez similaires.



Difficultés rencontrées et solutions apportées

Nous avons donc rencontré plusieurs problèmes, le premier étant l'affichage de des formes SVG à partir du code, ce problème fut résolu grâce à la bibliothèque Batik qui permettait donc un affichage du code SVG avec SWING. Ensuite nous avons voulu implanter tout ce qui était modification d'une forme existante cependant nous n'avions pas d'éléments permettant de revenir sur une forme particulière et son stockage. C'est là que la solution d'une Map c'est mise en place ainsi nous avons pu conserver les formes en les identifiant via un nom donné que nous stockions lors de sa création puis nous associons chaque nom au code SVG d'une forme. Cette solution nous a permis de mettre en place la suppression, le changement de nom, la translation…. Donc tout ce qui était lié à la modification d'une forme.

Nous avons eu des problèmes au niveau du parsing des commandes entré par l'utilisateur, car il a fallu différencier les commandes de création d'une forme avec toutes les commandes de modification d'une forme existante. Nous avons donc créé deux

types de parser différents et nous avons utilisé une Map encore une fois pour stocker les résultats du parsing pour pouvoir les réutiliser par la suite. Cependant le plus gros problème était d'exécuter les commandes de modifications d’une forme. Pour cela nous avons utilisé l’introspection afin d'exécuter les méthodes de la class dont le nom avait été donné et parser. Cette solution a permis d’avoir un code générale et d'utiliser peu importe la commande passée.

Documentation technique

Notre application comporte 3 packages principaux. Le package core, dans le lequel est contenu le Main, et la fenêtre de l'application FramePinte, le package graphic, comportant les classes et packages composant l'application côté dessin matriciel, et le package terminalSVG qui contient les classes et packages de l'application côté dessin vectoriel.

Dessin matriciel

Dans le package graphic, on retrouve 4 packages : 3 correspondent aux différentes parties d'un MVC : un package model, un package view et un package controller, et 1 contient les exceptions soulevées par l'application.

Dans le package model, on retouve la classe ToolContext, qui contient les différentes informations du contexte de l'application, telles que les coordonnées du point cliqué, la taille de l'outil etc. On retrouve aussi deux énumérations, CropTypes, qui énumère les façons dont on peut réaliser le crop, et ShapeTypes, qui sont les types de forme faisables avec l'outil ShapeTool. On retrouve aussi 3 packages, canva, color et tools, contenant respectivement les modèles relatifs à la toile, la couleur et aux outils.

Le package canva contient une classe Canva qui est le modèle de la toile. On y retrouve la liste de BufferedImage, l'index courant ainsi que les Graphics2D associés à la BufferedImage actuelle. Le model est un observable.

Le package color quant à lui contient la classe ColorModel, qui est aussi un observable et qui contient les couleurs primaires et secondaires, et si la couleur active est la primaire ou la secondaire.

Enfin le package tools contient la classe Toolbox, l'interface ToolComand ainsi que toutes ses réalisations. La classe Toolbox est un observable qui contient la liste des outils ainsi que des boutons associés à ceux-ci, l'outil actif, et les paramètres relatifs aux outil : taille, forme, type de forme et type de remplissage. Pour l'interface ToolCommand et ses réalisation, elles continennent la méthode execute qui prend en paramètre un ToolContext et qui sert à réaliser l'action associée à l'outil. La méthode execute varie selon les outils car ceux-ci ont tous une méthode de réalisation différente.

Dans le package view, on retrouve 5 classes : CanvaComponent, ColorSchemeInternalFrame, GraphicProjectPanel, ResizeDialog et ToolInternalFrame.

La classe CanvaComponent est la vue de la toile. Elle contient le contexte de l'application, ainsi qu'une variable canvaController qui permet d'effectuer les modification sur la toile lorsque l'utilisateur utilise un outil. Elle observe le model de la toile afin d'afficher à l'utilisateur les modifications lorsqu'il y en a.

La classe ColorInternalScheme est la vue créant la fenêtre interne de choix de couleur. Elle contient les boutons pour choisir la couleur primaire ou secondaire, un

champ pour le code hexadécimal de la couleur et un JColorChooser. Elle observe le model ColorModel afin de changer la couleur de ses boutons et le code héxadécimal lorsque la couleur est changée autrement que par le JColorChooser.

La classe GraphicProjectPanel est la vue principale du mode de dessin graphique. Elle contient les différents éléments de l'application graphique, soit la toile, la boîte à outils et la palette de couleurs. Elle initialise aussi les modèles et les controlleurs utilisés par les controlleurs et les vues, ainsi que le menu et les différents options de menu. C'est aussi dans cette classe que les entrées clavier sont définies et associées à leurs actions.

La classe ResizeDialog est la vue de la fenêtre affichée à l'utilisateur lorsqu'il souhaite redimensionner ou recadrer l'image. Elle contient deux champs pour la largeur et la hauteur souhaitée pour l'image, un système de choix de position lors du recadrage de l'image et un bouton pour valider. Lors du recadrage, lorsque l'utilisateur choisit une position, les champs horizontalAlign et verticalAlign sont mis à jour avec des valeurs CropTypes correspondant à la position choisie.

La classe ToolInternalFrame est la vue créant la fenêtre interne de choix d'outil. Elle contient une variable toolController permettant d'influer sur la toolbox en cas de changement, ainsi que l'outil actif. Elle contient aussi tous les boutons relatifs aux propriétés des outils.

Le package controller contient les 3 controlleurs de l'application : CanvaController, ColorController et ToolController.

La classe CanvaController permet d'effectuer des modification relatives à la toile, et modifier effectivement la toile une fois fait, ce qui va permettre au modèle de notifier la vue et d'afficher les modifications.

La classe ColorController permet de récupérer et définir les propriétés du ColorModel.

La classe ToolController permet de récupérer et définir les propriétés de la Toolbox.

Enfin, le package exceptions contient les exceptions soulevées par l'application lors d'erreurs durant l'exécution : BadFormatException, ClipBoardVoidException et NullImageException.

L'exception BadFormatException étend RuntimeException.

L'exception ClipBoardVoidException étend RuntimeException.

Finalement, l'exception NullImageException étend RuntimeException.

Dessin vectoriel

Dans le package terminalSVG, on retrouve les 3 packages qui correspondent au modèle MVC : Modèle, Vue et Controller

Dans le package Controller, on retrouve simplement la classe ControllerTerminalPanel qui correspond au controller de la partie SVG de l'application.

Dans le package controller on peut trouver plusieurs choses :

- Le package parser qui contient les classes liés au parsing de la commande
- Le package SVGCommand qui contient toutes les classes liés aux différentes
- commandes qui sont possibles d'effectuer sur le canva SVG
- La classe SVGPreview qui correspond au canva SVG
- La classe history

Tests

Afin de détecter certaines erreurs éventuelles du projet, valider le comportement attendu du code, faciliter la maintenance nous avons décidé de réaliser des tests unitaires à l'aide de la bibliothèque JUnit 4.

Au niveau de la partie textuelle, quelques tests ont été réalisés sur le comportement de certaines formes dans l'objectif de vérifier le bon fonctionnement de leurs méthodes. Ensuite, nous avons testé à l'aide d'une unique classe de test l'interaction entre les différentes classes pour la création et la modification des formes au format SVG à travers les commandes utilisées dans le champ de saisie dans la classe TestModifier.

21 tests JUnit ont été créés pour tester la partie textuelle de l'application

```
crosskicker
@Test public void testRayon() {
    assertEquals( message: "Le rayon n'est pas bon !",  expected: 50.0 , c1.getRadius(), EPSILON);
    assertEquals( message: "Le rayon n'est pas bon !",  expected: 78.0 , c2.getRadius(), EPSILON);
}
```

Au niveau de la partie graphique, nous avons commencé par créer deux classes qui nous aideront lors de la rédaction des tests unitaires :

- LoadImage qui possède une méthode statique qui permet de charger une image en la retournant à partir d'un String représentant le chemin d'accès.

- ImageComparator qui possède une méthode statique qui permet de dire si deux images sont identiques en retournant un booléen et en prenant en paramètres les deux images à comparer

Nous avons ensuite testé ces deux classes afin d'être certain de pouvoir les utiliser. Nous avons par la suite rédigé les tests de l'application en initialisant une toile et l'outil souhaité, en commençant par le test du CanvaController puis les différents tests pour chaque outil.
Certaines exceptions personnalisées ont été créées dans le projet et sont également testées telles que NullImageException qui est soulevé si une image est nulle, ClipboardVoidException soulevé lorsque le presse papier est vide et BadFormatException soulevé lorsque l'image est enregistré dans un format qui n'est pas pris en charge par l'application.
52 tests JUnit ont été créés pour tester la partie graphique de l'application.

```
IcePingus +1
@Test
public void testHighlighterPoint() {
    BufferedImage expectedImage = LoadImage.loadImage( path: "src/Test/graphic/ImageTest/highlighterTool/TestPoint.png");
    this.toolbox.getActiveTool().execute( oldX: 40,  oldY: 40,  currentX: 40,  currentY: 40, this.canva.getBufferedImage(), this.canva.getG2(), MouseEver
    assertEquals( expected: true, ImageComparator.areImagesSimilar(this.canva.getBufferedImage(), expectedImage));
}
```

Résultats

Dessin matriciel

Au niveau des réalisations; nous avoins pu implémenter la majorité des fonctionnalités prévues. Les fonctionnalités prévues et réalisées sont au niveau matriciel :

- Outil crayon
- Outil gomme
- Outil seau
- Outil pipette
- Outil surligneur
- Outil texte
- Outil forme
- Sauvegarder l'image
- Importer une image
- Annuler
- Refaire
- Coller
- Redimensionner
- Recadrer
- Flip horizontal/vertical
- Réinitialiser l'image
- Passer en noir et blanc

Les fonctionnalités prévues mais non réalisées au niveau matriciel ont été :

- Outil sélection
- Plusieurs toiles
- Calques

Dessin vectoriel

Au niveau des réalisations; nous avoins pu implémenter la majorité des fonctionnalités prévues. Les fonctionnalités prévues et réalisées sont au niveau matriciel :

- Créer un Cercle
- Créer un rectangle
- Créer un ovale
- Créer un Polygone
- Modifier la couleur d'une forme
- Modifier l'emplacement d'une forme
- Modifier le nom d'une forme
- Effacer une forme
- Afficher l'aide sur les commandes
- Sauvegarder l'image au format SVG

- Réinitialiser l’image

Les fonctionnalités prévues mais non réalisées au niveau matriciel ont été :

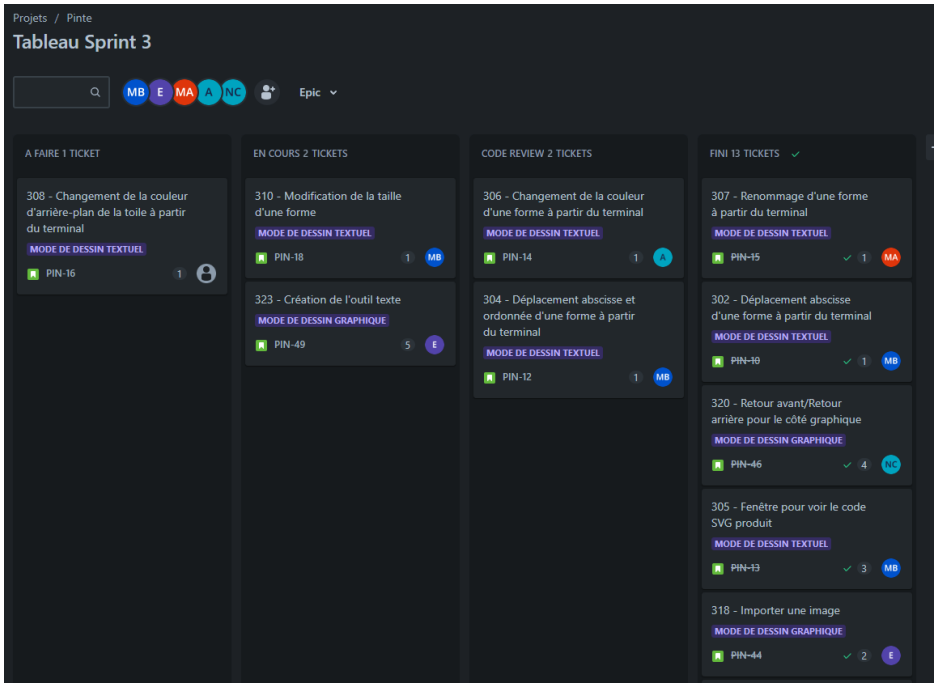
- Outil sélection
- Redimensionner la toile
- Changer la couleur de l’arrière plan de la toile

Gestion de projet

Lors de ce projet, nous avons mis en place la méthode agile SCRUM pour superviser et gérer notre projet.

Dans un premier temps,nous avons préparé l'organisation de l'équipe avec notamment la désignation des rôles.Nous avions donc un Scrum Master, Enzo Furriel, qui était responsable de s'assurer que l'équipe suit les principes et les processus SCRUM. Le Product Owner, soit Baptiste Touchais, quant à lui, était responsable de la vision du produit et de la gestion du backlog.

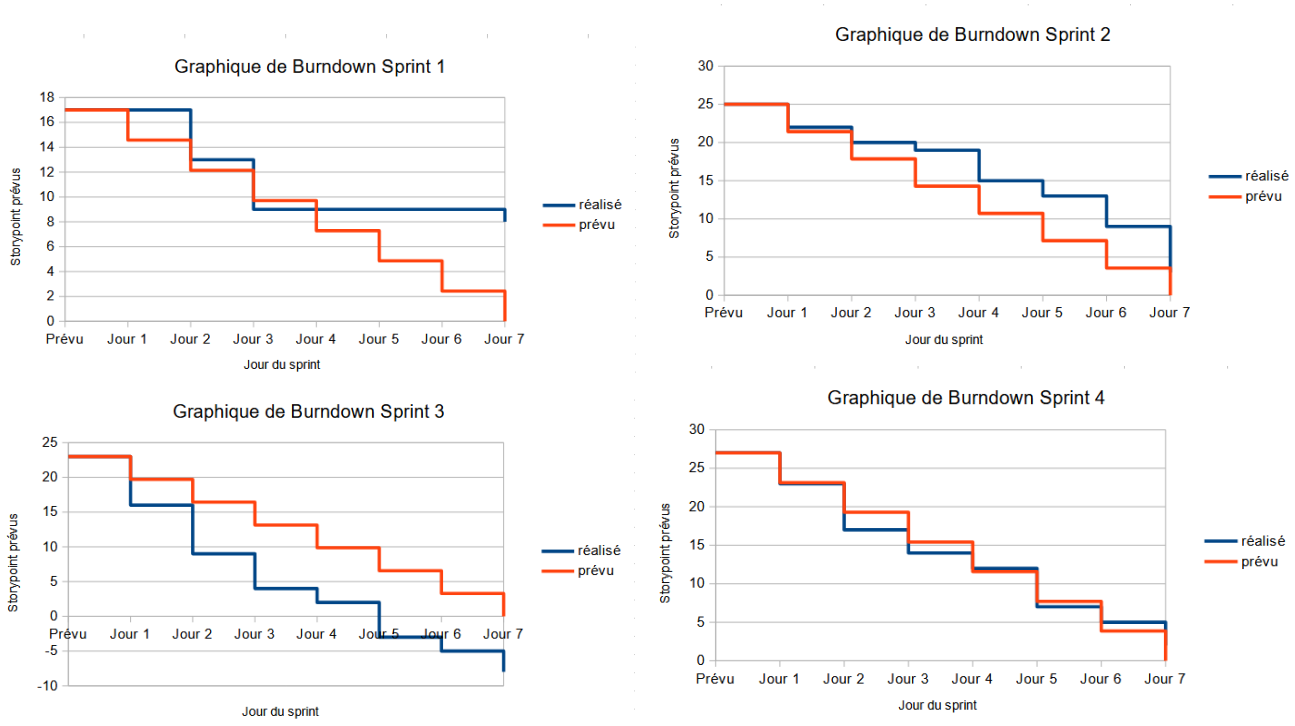
En ce qui concerne l'organisation de la production, nous avons utilisé JIRA, un outil de gestion. JIRA nous permet de créer des tâches, de les attribuer à des membres de l'équipe, de suivre leur avancement et de gérer le backlog du projet. Une des fonctionnalités essentielles de JIRA est la gestion des Epics. Chaque US est géré par une seule personne.



En parallèle, nous avons également utilisé Git. Cela nous a permis de travailler de manière collaborative, de suivre les modifications du code source et de gérer efficacement les différentes versions de notre logiciel. Nous avons organisé notre développement à l'aide de différentes branches. Nous avons la branche Master, qui représente la version stable de notre logiciel. Ensuite, nous avons la branche Dev, qui regroupe le travail sur les fonctionnalités du sprint en cours. Enfin, nous avons les branches User Story, les branches dédiées pour travailler sur une US spécifique. Cette approche nous permet de travailler en parallèle sur différentes fonctionnalités tout en maintenant une version stable du logiciel.

Ensuite, nous avons mis en place des sprint reviews, où nous revenons sur les fonctionnalités développées lors du sprint. C'est l'occasion de se concerter et de s'assurer que nous suivons la bonne direction.

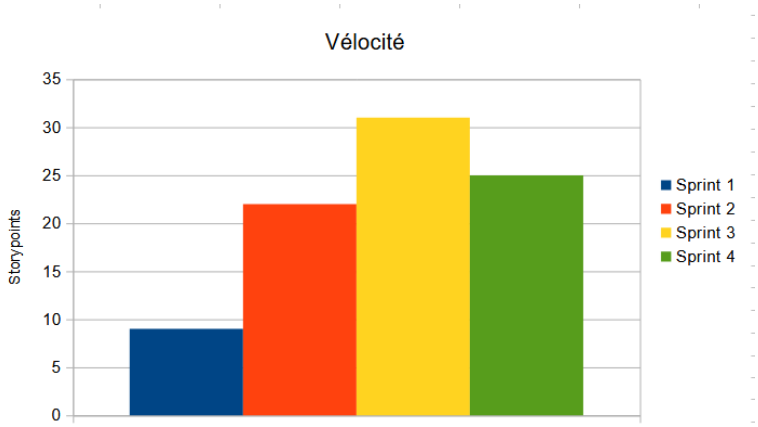
Un outil essentiel pour suivre l'avancement du projet lors de nos reviews est le graphique Burndown. Cela nous permet de visualiser la quantité de travail restante par rapport au temps disponible. Il nous aide à estimer si nous sommes sur la bonne voie pour atteindre nos objectifs et à ajuster notre planification si nécessaire.



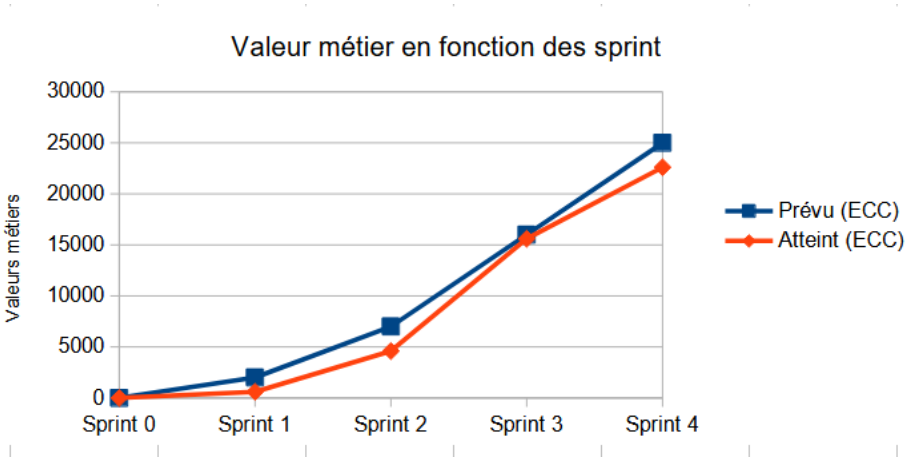
Enfin, nous allons voir via une revue globale du projet son avancement dans l'ensemble. Cette revue nous permet de prendre du recul, d'évaluer la performance globale du projet et d'apporter les ajustements nécessaires.

Pour cela nous avons 2 vues différentes:

- le graphique de vélocité, un outil visuel qui nous permet de suivre la valeur métier générée par le projet au fil du temps. Contrairement au graphique Burndown qui se concentre sur la quantité de travail restante, ce graphique Burnup met l'accent sur la valeur délivrée.



- Un graphique retraçant la valeur métier au fil des sprints.



Enfin, nous avons également réalisé des Daily. De notre côté, nous n'avions pas prévu concrètement d'en réaliser au cours du projet, mais nous nous sommes surpris à le faire naturellement tout au long de la production. Nous avons échangé assez régulièrement et naturellement sur l'avancement de nos tâches respectives et sur l'avancement global des sprints afin d'orienter au mieux le projet.

Conclusion

Pour conclure, la majorité des objectifs que nous nous sommes fixés ont été réalisés. Le projet a été une expérience enrichissante tant au niveau humain qu'au niveau technique. Ce projet a été l'occasion de travailler en équipe et de mettre en place les méthodes agiles SCRUM qui étaient nouvelles pour certains d'entre nous. Ces méthodes nous ont permis de travailler plus efficacement en prenant en compte les forces et faiblesses de chacun pour avancer ensemble dans la bonne direction. L'outil Jira ainsi que les réunions à chaque séance ont fortement contribué à créer une bonne organisation et une bonne cohésion d'équipe. Au niveau technique, ce projet nous a permis de mettre en oeuvre nos compétences en java ainsi qu'en conception pour réaliser l'application. En effet, nous avons pu mettre en place de nombreux patrons de conceptions tel que le patron commande ou encore observer ainsi que l'utilisation de SWING pour réaliser les interfaces graphiques. Nous sommes fiers du résultat obtenu et nous nous sommes surpris a utiliser notre application sur Linux a la place de certains logiciels de dessins ressemblant à Paint.