UNIVERSITÉ BLAISE PASCAL

Polytech Clermont-Ferrand

Projet de conception et programmation orientées objet
4 ème année

Département de Mathématique et Informatique
Filière : Génie Mathématique et Modélisation.

Le logiciel CAOStar
Manuel développeur

Clavel Jean
Mahamadou Abdoul Jalil
Maisonneuve Justin
Mortabit Zouhair
Pose Céline

Année universitaire 2016-2017 .

Table des matières

I. Analyse et Conception du logiciel

i. Présentation du logiciel

Notre logiciel, "CAOStar", a pour but de permettre aux utilisateurs de créer un véhicule grâce à des formes géométriques de taille et de couleur variées. Ces dernières peuvent en effet être choisies par le client afin de lui accorder plus de liberté quant à la conception de son véhicule. De plus, une fois celui-ci terminé, il pourra le faire se déplacer sur un axe horizontal ou bien sur un axe vertical, et aura même la possibilité de changer d'axe de déplacement à tout moment,ce qui rend notre logiciel plus agréable d'utilisation. D'autre part, soucieux d'apporter un maximum de fonctionnalités et de répondre au mieux aux attentes de nos clients, nous avons incorporé à notre logiciel la possibilité de moduler la vitesse de déplacement.
Dans la partie suivante, nous vous présenterons l'analyse UML de ce logiciel.

ii. Diagramme de cas d'utilisation

Notre logiciel permet à l'utilisateur de créer un véhicule à l'aide de formes géométriques et de le déplacer dans la direction souhaitée. Pour ce faire, deux options s'offrent à lui : **Composer** et **Déplacer**. Après avoir effectué ces opérations, il peut choisir de **Quitter** le logiciel. Ces opérations sont résumées dans le diagramme de cas d'utilisation ci-dessous :

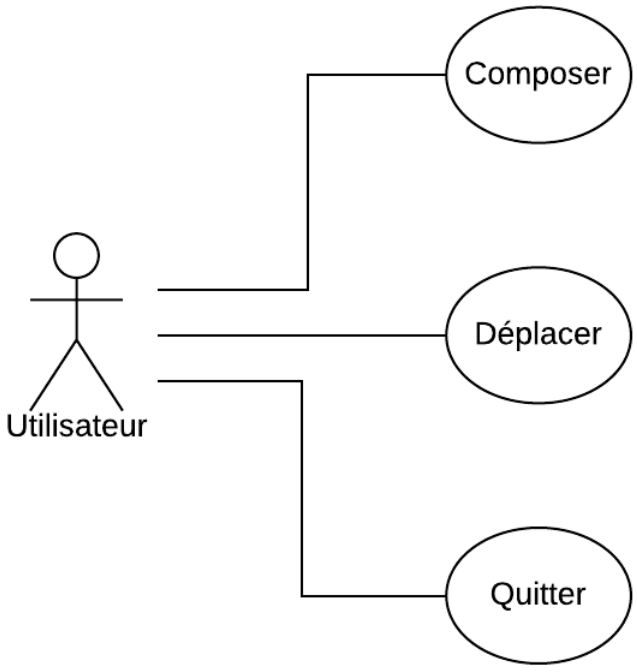

FIGURE 1 – Diagramme de cas d'utilisation

Scénario de Composer :

Précondition : L'utilisateur doit posséder le logiciel.

PostCondition : Un véhicule s'est créé. Il est ensuite centré dans la zone de composition.

Scénario de Déplacer :

Précondition : L'utilisateur doit avoir créé un véhicule et l'avoir validé.

PostCondition : Le véhicule se déplace suivant la direction choisie jusqu'à ce que l'utilisateur l'arrête.

iii. Analyse de haut niveau

Dans cette partie, nous présenterons, d'un point de vue global, les deux fonctionnalités les plus importantes de notre logiciel, à savoir, la composition et le déplacement du véhicule. Pour cela, nous allons voir le diagramme de séquence du système global du logiciel puis les diagrammes de séquence système des deux fonctionnalités (composition et déplacement). Dans ces diagrammes, nous ne représentons que l'intéraction de l'utilisateur avec le logiciel et non pas ce qu'il se passe à l'intérieur (on parlera de boîte noire).

Diagramme de séquence du système

Dans ce paragraphe, notre logiciel est vu de l'extérieur (nous n'allons pas préjuger de la façon dont il sera réalisé) et les fonctionnalités vues précédemment se déroulent selon un ordre précis.
En effet, l'"utilisateur génère dans un premier temps l'événement Composer qui conduit à la création du véhicule. Le système répond par le centrage du véhicule dans la fenêtre de composition. Ensuite l'utilisateur déplace le véhicule, et cela, autant de fois que souhaitées, puis il le stoppe.
Enfin, l'arrêt de notre logiciel se traduit par la fermeture de ce dernier en cliquant sur le bouton quitter.
Le fonctionnement plus précis de ces étapes sera détaillé dans les prochaines parties.
Le diagramme suivant est une illustration de ces propos :

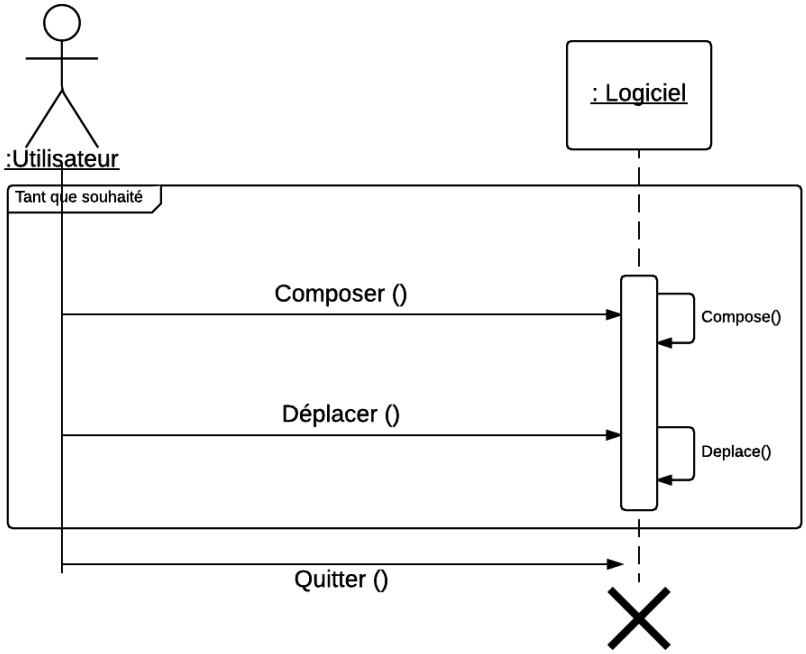

FIGURE 2 – Diagramme de séquence du système

Diagramme de séquence de Composer

De manière générale, pour composer son véhicule l'utilisateur doit tout d'abord appuyer sur le bouton **Composer**. Suite à cela, il peut choisir et saisir la forme géométrique qu'il souhaite pour le constituer, la déplacer jusqu'à la zone de composition où il la relache à l'endroit voulu. Cette opération est à réitérer jusqu'à l'obtention du véhicule désiré. Enfin, la composition se termine par la validation du véhicule, effectuée en appuyant sur le bouton **Valider**, et qui se manifeste par son centrage dans la zone de composition.
Toutefois, ce scénario est celui d'un cas sans difficultés. En effet, si l'utilisateur commet un placement erroné, il a la possibilité de revenir en arrière en utilisant le bouton **Cancel** qui entraînera la suppression de la dernière pièce placée dans la zone de composition. De plus, s'il n'est pas satisfait de sa création dans sa globalité, il peut la supprimer entièrement grâce au bouton **CancelAll**.
Ces deux variantes constituent des choix de notre part et sont destinés à rendre l'utilisation de notre logiciel plus confortable puisqu'elles évitent à l'utilisateur de devoir recommencer tout son véhicule pour une simple erreur de manipulation.
Les fonctions utilisées dans ce diagramme ne portent pas les mêmes noms que dans le code. Pour la plupart ce sont de simples gestionnaires d'événements. Nous detaillerons cela dans les diagrammes de séquences approfondis.
Nous pouvons observer le fonctionnement décrit sur le diagramme ci-dessous :

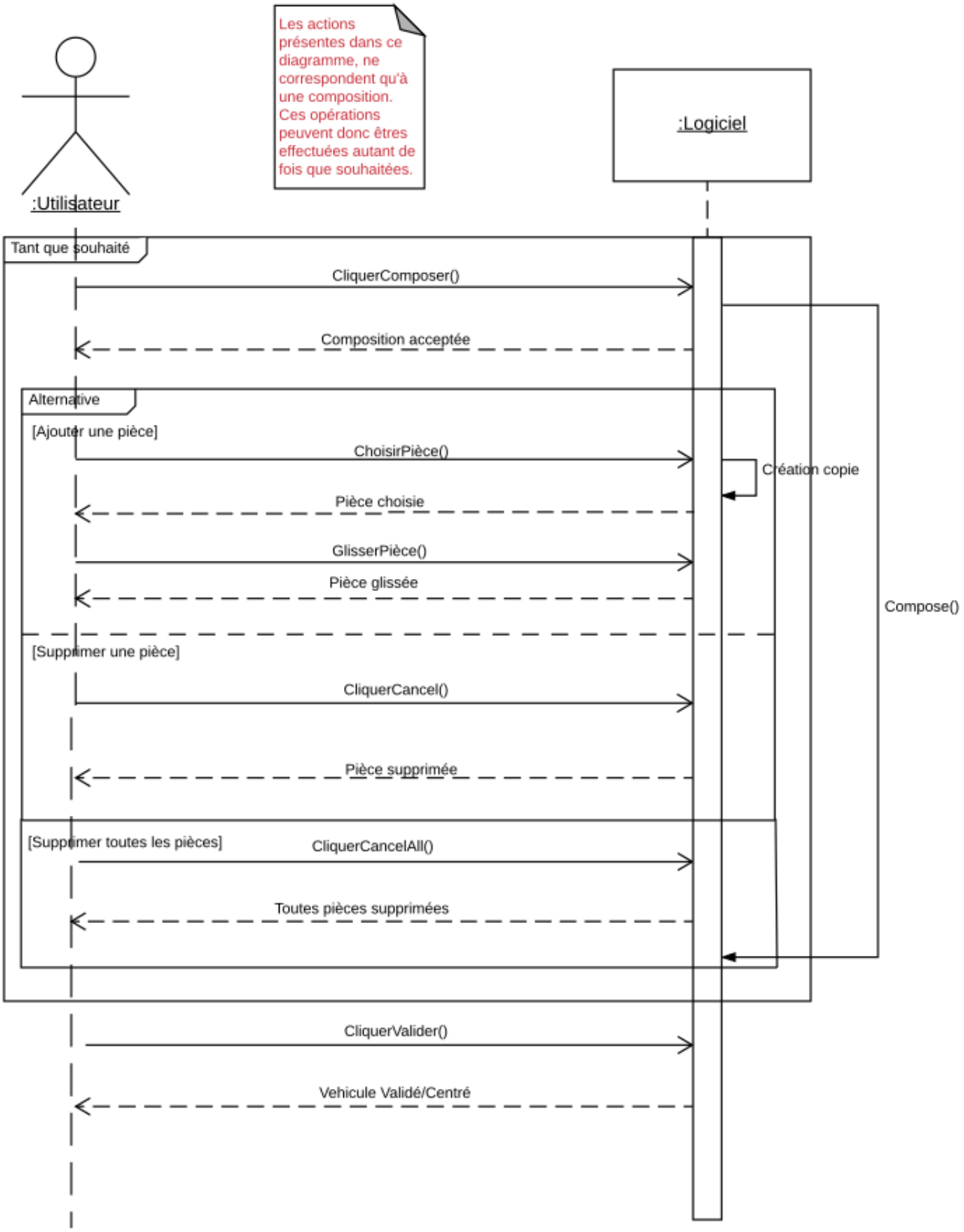

FIGURE 3 – Diagramme de séquence du système : **Composer**

Diagramme de séquence de Deplacer

Après la création du véhicule, comme vu précedemment, l'utilisateur à la possibilité de le déplacer selon une direction. Nous avons défini deux diagrammes pour le déplacement : le premier étant le diagramme de séquence système de **Deplacer**, et le deuxième son diagramme détaillé.

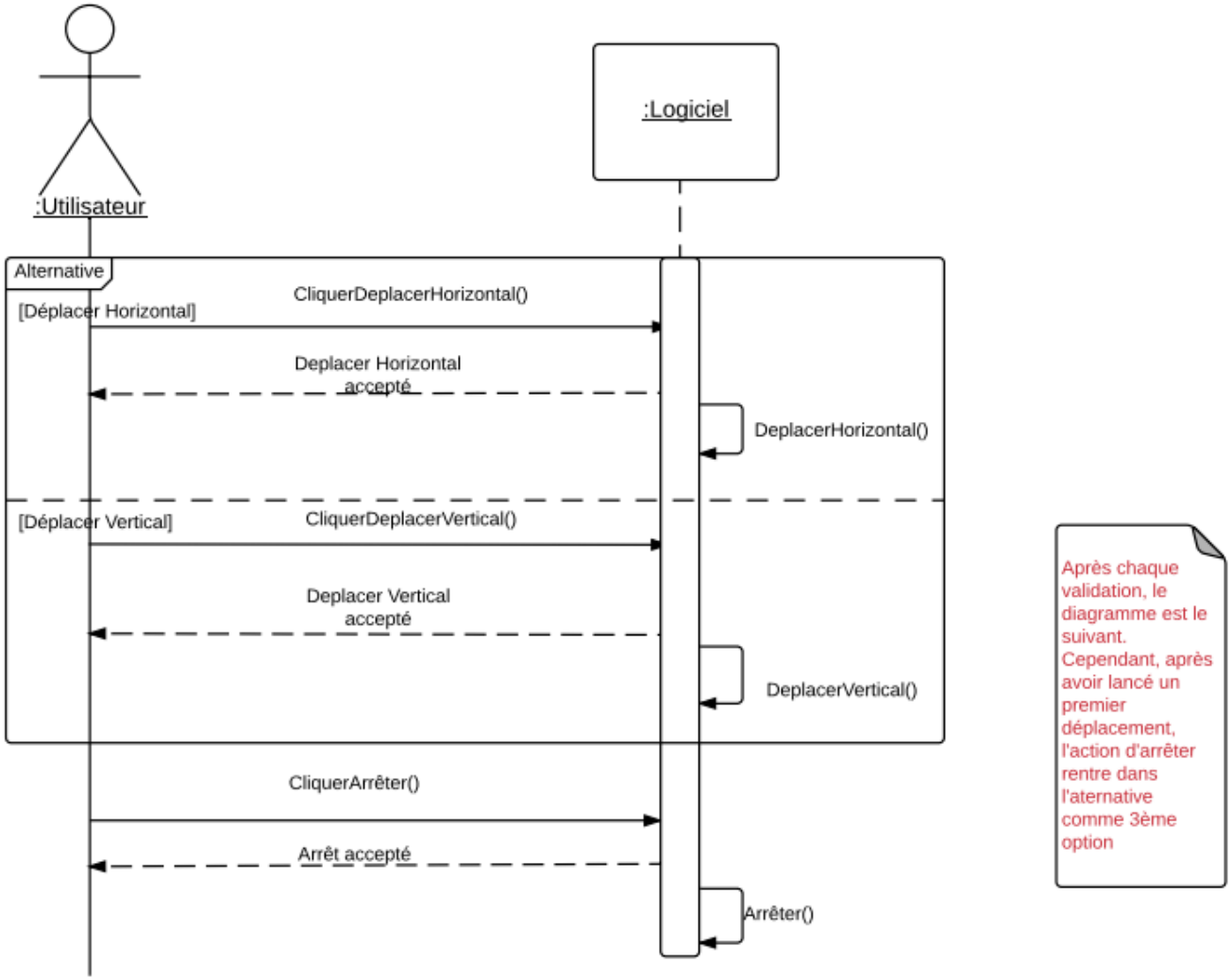

FIGURE 4 – Diagramme de séquence du système :**Deplacer**

Le diagramme de séquence système du déplacement décrit de manière générale le déplacement du véhicule. Ce dernier repose sur deux fonctions globales à savoir : **DeplacerHorizontal()** et **Deplacer-Vertical()**. Comme leur nom l'indique, elles permettent de déplacer le véhicule soit horizontalement, soit verticalement. Il faut noter que ces deux fonctions n'existent pas réellement, elles nous permettent juste de généraliser le déplacement.

Le choix de la direction de déplacement nécessite la validation de la composition. Une fois la création du véhicule validée, les boutons **DéplcacerHorizontal** et **DéplacerVertical** déviennent accessibles. Dès que l'utilisateur choisit une direction alors la direction opposée et l'arrêt du véhicule rentrent dans l'alternative de la boucle du déplacement (cf figure 4).
Pour plus de détails voir le diagramme de séquence détaillé de Deplacer (cf figure 10).

Diagramme de classe général

Le diagramme de classe reste le diagramme le plus important (c'est le seul obligatoire) lors d'une analyse UML. Il permet de fournir une représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation. C'est dans celui-ci que l'on représente les différentes classes utlisées dans le logiciel, ainsi que leurs attributs et fonctions associées.

Voici dans un premier temps, le diagramme de classe réduit, c'est-à-dire sans le détail des fonctions et des attributs :

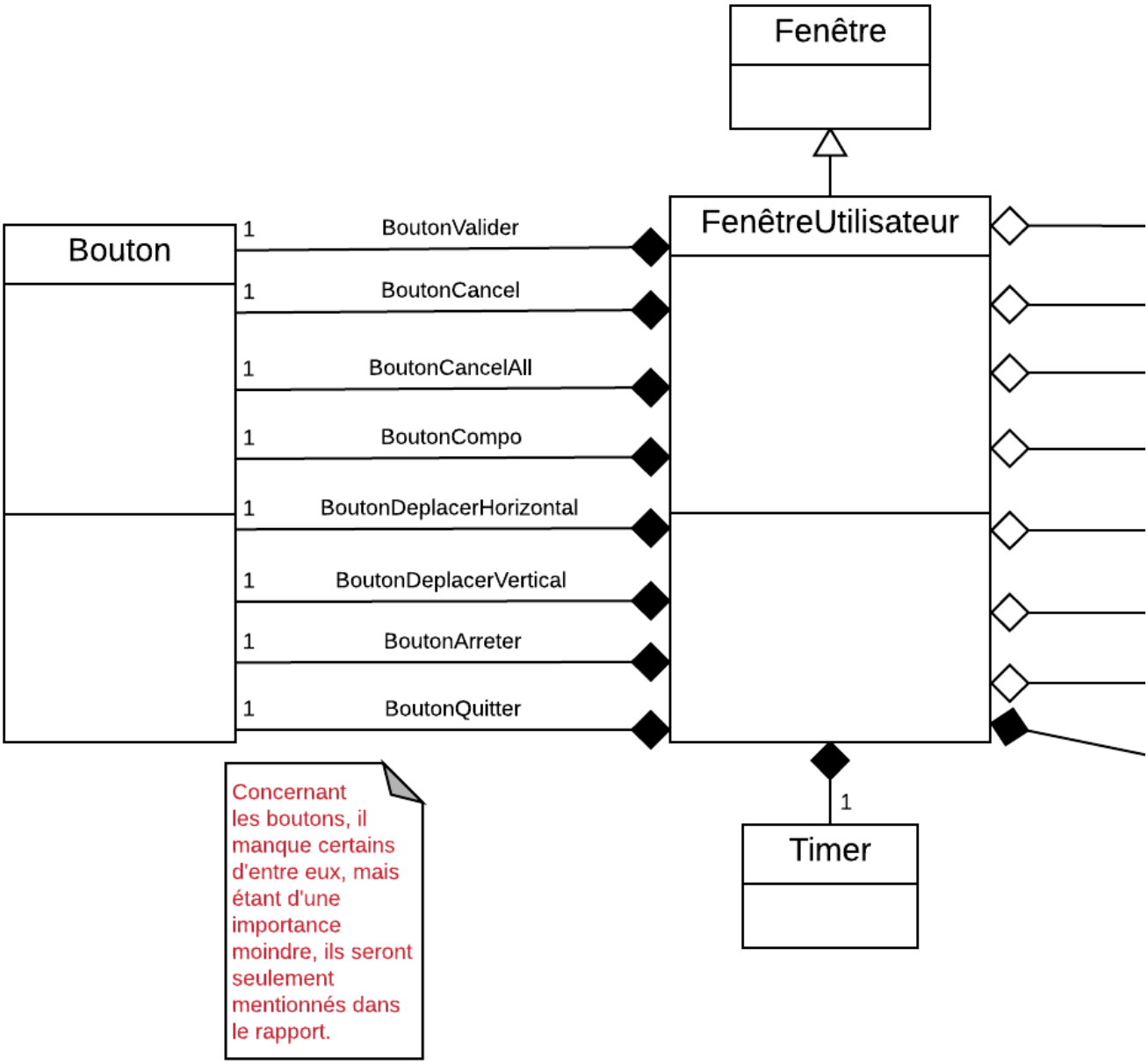

FIGURE 5 – Diagramme de classe réduit (1)

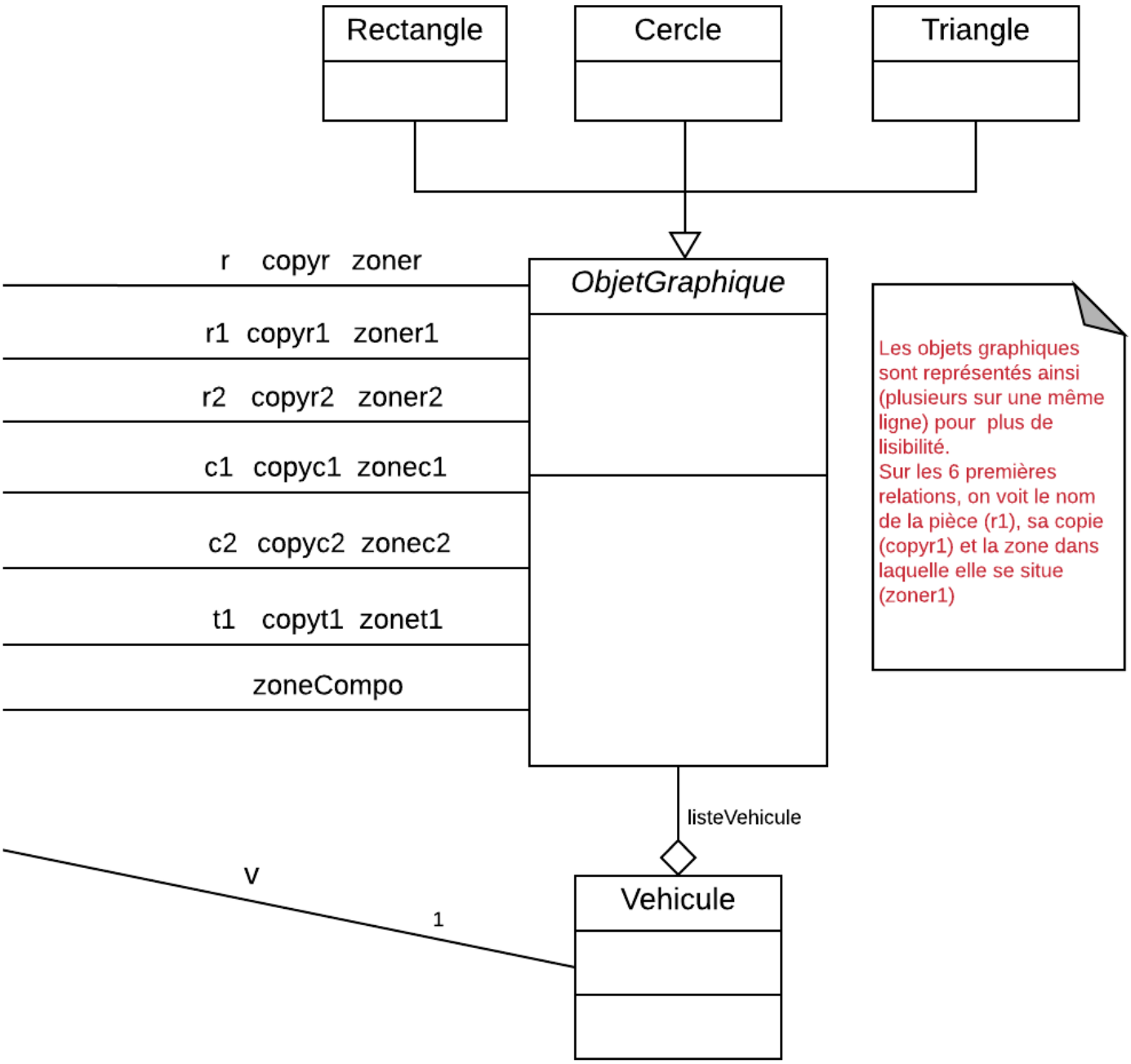

FIGURE 6 – Diagramme de classe réduit (2)

Notre logiciel possède principalement une **fenêtre utilisateur** (qui est la composante principale de celui-ci), qui se compose de plusieurs **boutons**, d'**objets graphiques** et enfin, d'un **véhicule** (noté v) étant lui-même composé d'objets graphiques.

Concernant les boutons, les principaux sont au nombre de 8 et gèrent la quasi totalité des fonctionnalités du logiciel à savoir, la composition, la suppression de pièces et du véhicule entièrement, la validation, le déplacement et son arrêt, et enfin la fermeture du logiciel. Cependant, comme nous pouvons le voir sur la petite note, ce ne sont pas les seuls boutons du logiciel, mais de part l'importance moindre des autres (grossissement, couleur et vitesse), nous avons décidé de ne pas les incorporer à

notre diagramme de classe. La fenêtre principale possède également un **timer** servant à gérer la partie déplacement du logiciel.

La deuxième classe indispensable à notre logiciel est la classe des **objets graphiques**. C'est grâce à elle que nous allons pouvoir composer et donc créer notre véhicule. Pour les objets graphiques nous nous sommes limités à 3 classes filles à savoir, le **rectangle** (servant également à instancier le **carré**), le **cercle** et enfin le **triangle**. Pour ce qui est des pièces dipsonibles à la composition d'un véhicule, la fenêtre utilisateur possède 3 rectangles (r,r1,r2), 2 cercles (c1,c2) et enfin un triangle (t1), ainsi qu'une copie de chacune d'entre elles (copyc1,copyt1,...). Cependant, les objets graphiques ne servent pas uniquement à la composition, ils servent également à définir les différentes zones du logiciel. La fenêtre utilisateur possède donc 1 rectangle supplémentaire pour définir la zone de composition (zoneCompo) et 6 autres rectangles pour définir les différentes zones de chaque pièce (zonec1, zonet1, ...).

Pour finir, la classe **vehicule** se définit simplement par plusieurs objets graphiques, stockés dans une liste appellée "listeVehicule".

Passons maintenant aux détails de chaque classe :

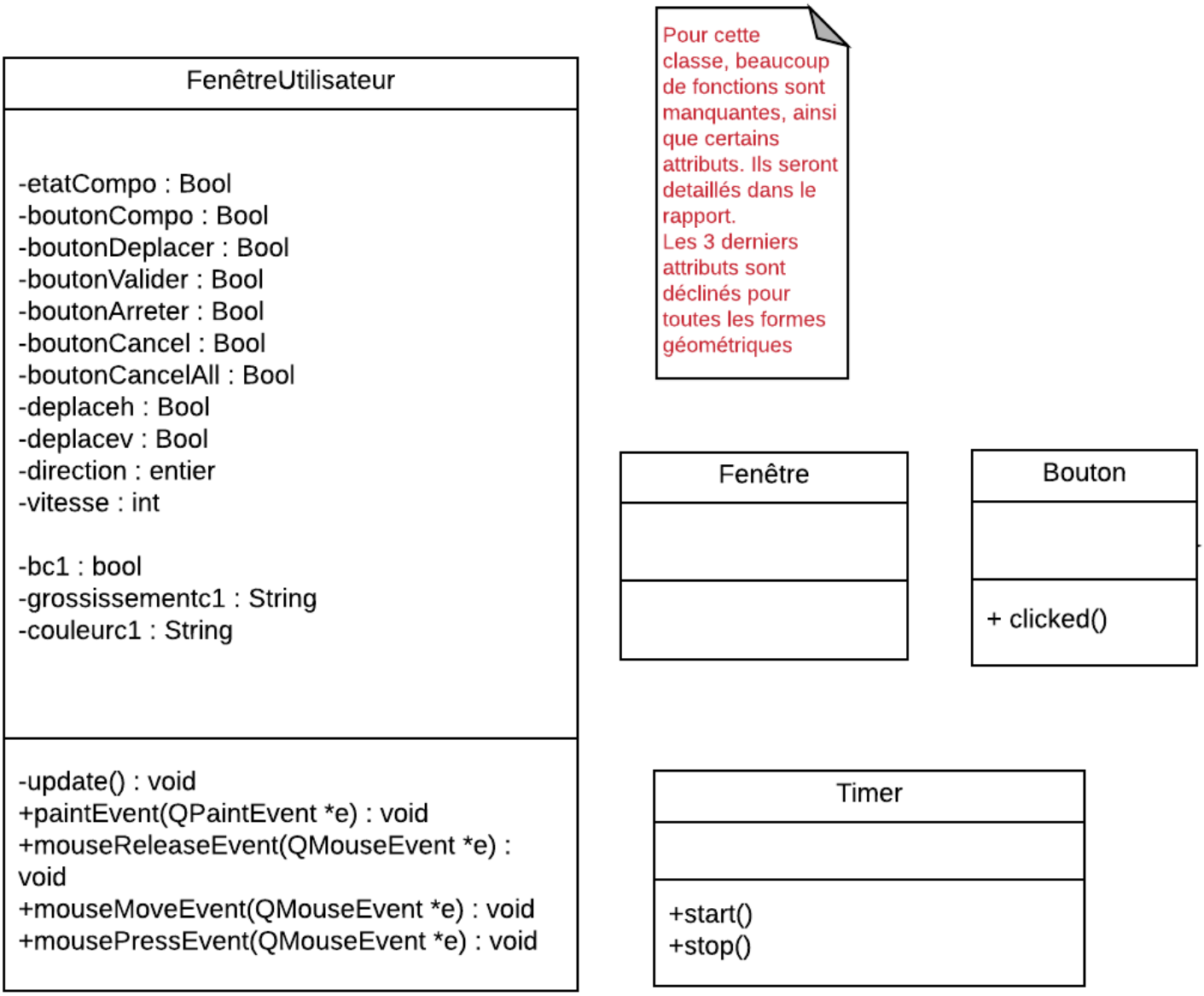

FIGURE 7 – Diagramme de classe détaillé (1)

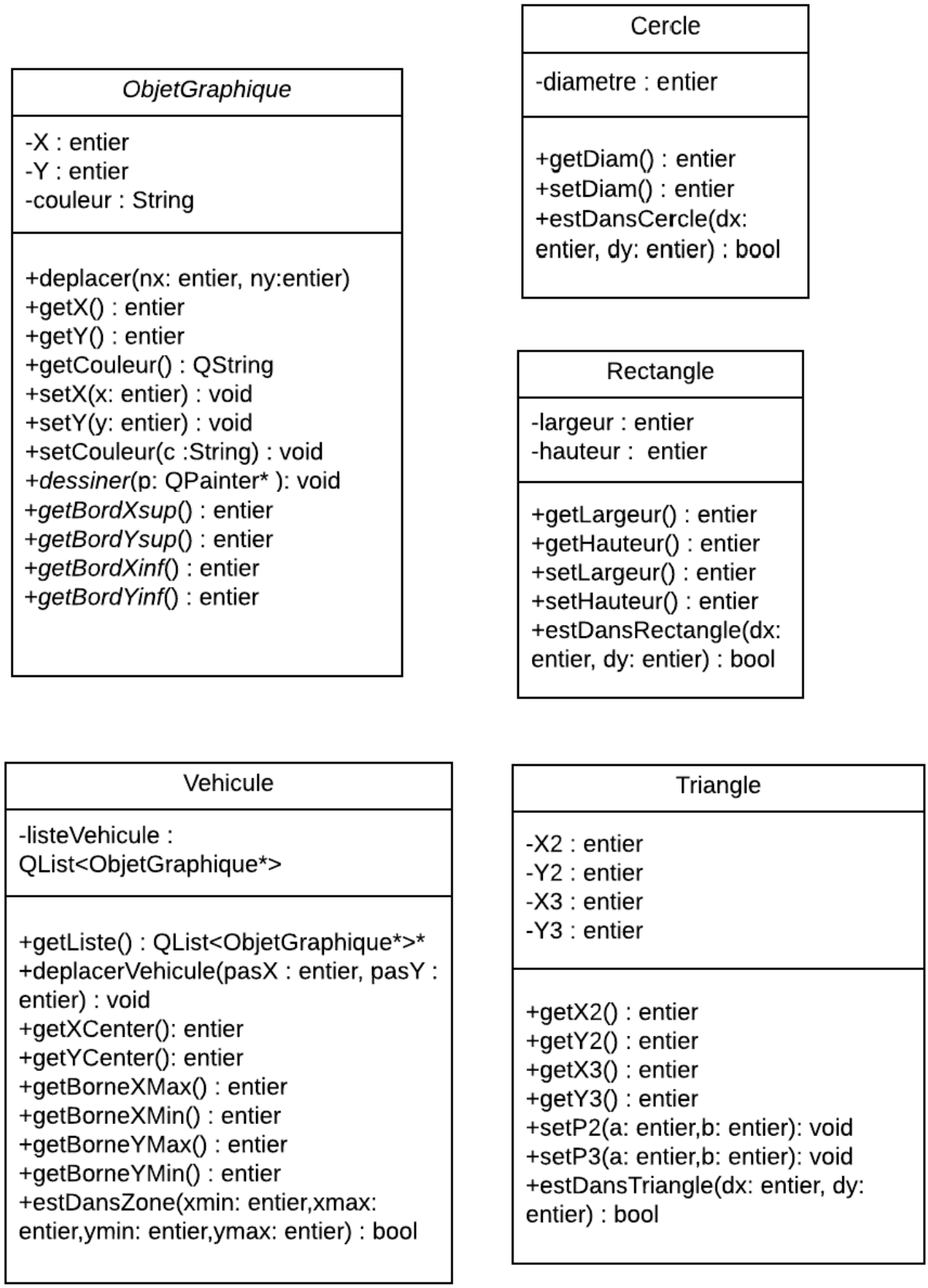

FIGURE 8 – Diagramme de classe détaillé (2)

Commençons par la **fenêtre utilisateur**. Celle-ci est la plus grande en terme d'attributs et de fonctions de part son importance majeure dans le logiciel (tout se passe dans cette fenêtre). Certaines fonctions et attributs ne sont pas répertoriés ici de part leur importance mineure dans le logiciel.
Ici l'attribut "etatCompo" gère le passage dans l'état de composition et donc la possibilité ou non à aller chercher et déplacer des pièces. Il faut savoir que nous ne possédons pas d'autres booléens gérant les états (comme etatValide ou etatDeplacer) car le passage dans ces états est géré par l'activabilité des différents boutons du logiciel (représenté par les booléens "boutonDeplacer" ou "boutonValider"). Par exemple, lors de la composition du véhicule, nous ne pouvons pas le faire déplacer, non pas parce que nous ne sommes pas dans un état de déplacement, mais tout simplement car les deux boutons de déplacement ne sont pas activables lors de la composition. Cette manipulation explique le fait que nous ne possédons pas de diagramme d'état. La classe possède également d'autre attributs gérant le déplacement et la vitesse du véhicule ou encore le grossissement et la couleur de chaque pièce.

Concernant ses fonctions, les plus importantes restent les suivantes : la fonction "update()" associée au **Timer** pour le déplacement, le "paintEvent()" permettant d'afficher et de dessiner les composantes du logiciel et enfin les 3 gestionnaires d'évenements "mousePressEvent()", "mouseMoveEvent()" et "mouseReleaseEvent()" permettant de gérer les actions de la souris, c'est-à-dire ce que le logiciel doit faire lorsque l'on clique, déplace ou relache le clic de souris.

La classe **Bouton**, étant une classe prédéfinie en Qt, est représentée ici avec la fonction "clicked()", fonction gérant l'activation du bouton correspondant (par exemple, pour le bouton valider, la fonction correspondante sera "on-boutonValider-clicked()"). La classe **Timer**, étant également prédéfinie dans Qt, possède deux fonctions, à savoir "start()" et "stop()" gérant l'activation et l'arrêt du timer.

Passons à la classe **ObjetGraphique**. Celle-ci est une classe abstraite de part la fonction "dessiner()" qui n'est pas définie de la même façon en fonction du type d'objet graphique (rectangle, cercle ou triangle). Chaque objet graphique est défini par un couple de coordonnées indiquant leur position d'origine, ainsi qu'une couleur. D'autres attributs leur sont donnés en fonction du type d'objet graphique, par exemple, pour le rectangle, on ajoute deux attributs pour la hauteur et la largeur de celui-ci (pour le cercle, on ajoute le diamêtre). Seul le triangle a une définition un peu différente des deux autres car celui-ci se définit non pas par un couple de coordonnées et des longueurs, mais par 3 couples de coordonnées indiquant chaque sommet du triangle. Dans chacune de ces classes il est possible de récuperer la valeur des différents attributs avec les getter et setter, et chacune possède une fonction permettant de savoir, en fonction d'un couple de coordonnées, si l'on se situe à l'intérieur de l'objet graphique (fonctions "estDansRectangle()", "estDansCercle()", ...). Pour chaque objet graphique, on peut obtenir les coordonnées de leurs extrémités (servant à calculer le centre d'un véhicule et à vérifier les conditions d'arrêt du déplacement) grâce aux fonctions "getBordXsup()", "getBordYinf(), ... Enfin, pour gérer leur déplacement, il y a la fonction "déplacer()" qui déplacera leurs coordonnées d'origine vers une nouvelle position.

Pour terminer sur les classes, la classe **Vehicule**, comme nous l'avons dit précédement, est composée d'une liste d'**ObjetGraphique**. Cette classe comprend la fonction "deplacerVehicule()" qui décalera d'un pas X et Y, tous les objets de la liste (en utilisant la fonction "deplacer()" de chaque objet graphique). Les fonctions "getXCenter()" et "getYCenter()" nous donnent les coordonnées du centre du véhicule. Ce centre est calculé en considérant notre véhicule comme un grand rectangle (chaque extrémité de ce rectangle entourant est obtenue grâce aux fonctions "getBorneXMax()", "getBorneYMin()", ...).

Dans cette partie, nous avons donc cité toutes les classes présentes dans notre logiciel, leurs fonctions et attributs ainsi que les relations qui les lient entre elles.

iv. Conception detaillée

Diagramme de séquence de Composer approfondis

La composition du véhicule consiste en une série d'opérations qui peuvent être réaliées à l'infini dans le but de créer un véhicule formé de pièces géométriques de taille et de couleur variées.
Pour pouvoir commencer à composer, il faut avant toute chose cliquer sur le bouton **Composer** (la fonction clicked() est une fonction prédéfinie par Qt sur des widgets). Cette action entraîne la création du véhicule, qui à ce stade correspond à une liste d'objets vide. Une seule possibilité s'offre alors à l'utilisateur, cliquer sur la forme géométrique de son choix, faisant appel à la fonction "mousePressEvent", puis déplacer la copie de celle-ci vers la zone de composition grâce au gestionnaire d'évènements "mouseMoveEvent". En effet, le clic de la souris sur une figure déclenche non pas une fonction de copie à proprement parlé mais la création au même emplacement d'une figure identique. L'utilisateur doit ensuite relacher la souris à l'endroit où il souhaite positionner sa pièce pour que la fonction "mouseReleaseEvent" soit activée. Toutefois, quelques conditions doivent être vérifiées puisque la forme ne peut être placée que dans la zone de composition. Dans le cas échéant, elle est automatiquement supprimée. Lorsque cette dernière remplie les conditions, elle est ajoutée à la liste d'objets du véhicule.
Une fois la première forme géométrique positionnée, trois options s'offrent à l'utilisateur. Il peut soit réitérer la manipulation précédente s'il veut ajouter une pièce à son véhicule, soit supprimer la dernière qu'il a disposé, soit supprimer toutes les pièces. Dans le cas de la suppression de la dernière figure, il suffit à l'utilisateur de cliquer sur le bouton **Cancel**, qui, lors de son activation, fera appel à la fonction prédéfinie sur les listes de QT, pop-back(), pour qu'elle soit supprimée de la liste. En revanche, s'il choisit le bouton **CancelAll**, la liste est vidée, ce qui se traduit par la suppression de toutes les pièces. Ces actions vont être répétées jusqu'à ce que le véhicule soit terminé, puis pour signifier la fin de la composition, l'utilisateur devra appuyer sur le bouton **Valider**. La création va alors être centrée grâce à une fonction qui va la déplacer de la distance qui la sépare du centre de la zone de compisition.

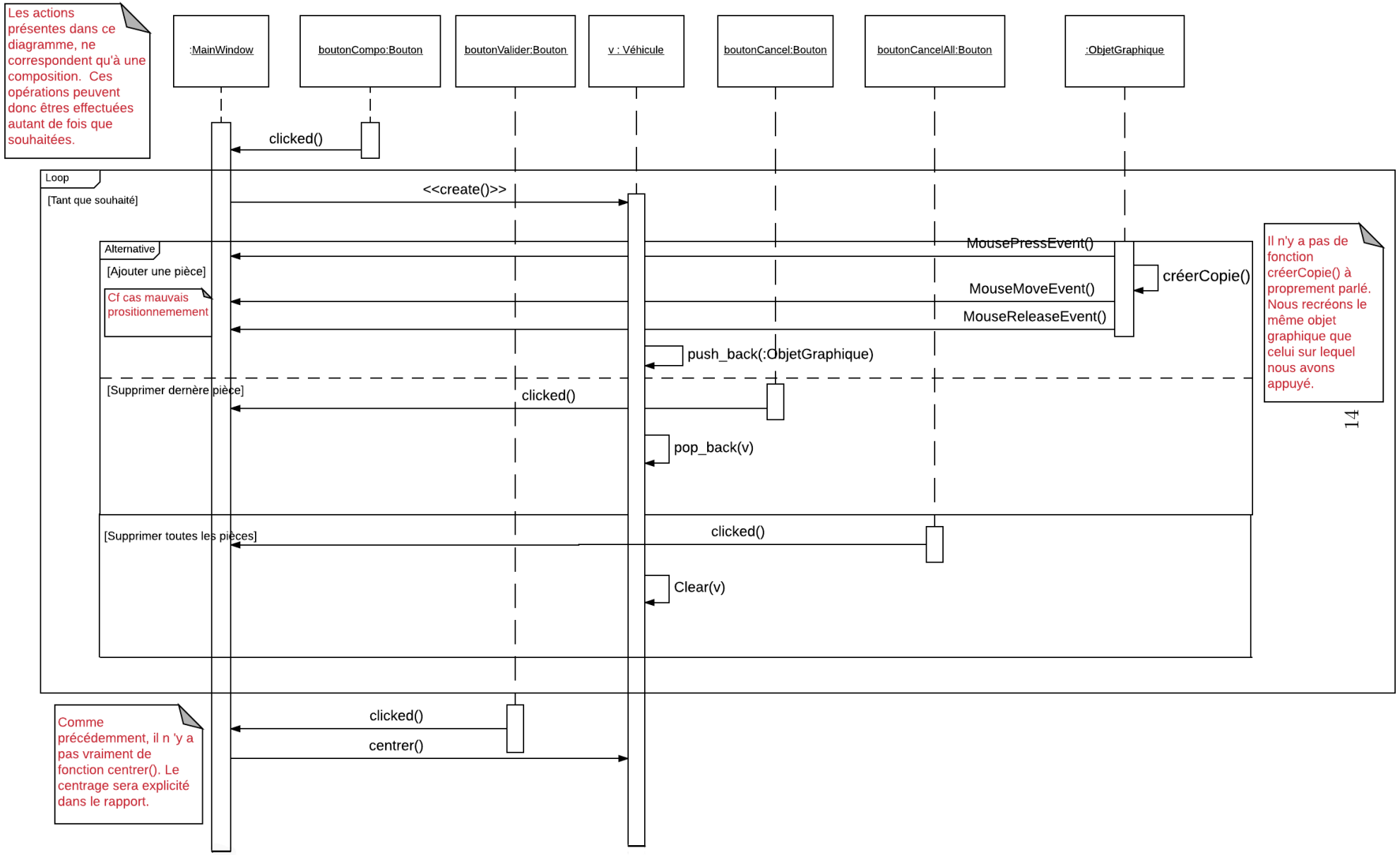
Les actions présentes dans ce diagramme, ne correspondent qu'à une composition. Ces opérations peuvent donc êtres effectuées autant de fois que souhaitées.
:MainWindow
boutonCompo:Bouton
boutonValider:Bouton
v : Véhicule
boutonCancel:Bouton
boutonCancelAll:Bouton
:ObjetGraphique
clicked()
Loop
[Tant que souhaité]
<<create()>>
Alternative
[Ajouter une pièce]
Cf cas mauvais positionnemement
MousePressEvent()
créerCopie()
MouseMoveEvent()
MouseReleaseEvent()
push_back(:ObjetGraphique)
[Supprimer dernère pièce]
clicked()
pop_back(v)
[Supprimer toutes les pièces]
clicked()
Clear(v)
Il n'y a pas de fonction créerCopie() à proprement parlé. Nous recréons le même objet graphique que celui sur lequel nous avons appuyé.
Comme précédemment, il n 'y a pas vraiment de fonction centrer(). Le centrage sera explicité dans le rapport.
clicked()
centrer()

Diagramme de séquence de Déplacer approfondis

Nous allons à présent détailler la fonction **Déplacer**.
Pour déplacer son véhicule, l'utilisateur va devoir tout d'abord choisir s'il souhaite le déplacer verticalement ou horizontalement. Dans les deux cas, la procédure est la même. En effet, il va devoir cliquer sur le bouton correspondant à son choix (**DeplacerVertical** ou **DeplacerHorizontal**) et en fonction de cela, l'entier direction prend soit la valeur 0, soit la valeur 1 afin de gérer la fonction de déplacement. Une fois le choix effectué, le timer est déclenché grâce à la fonction prédéfinie "start(5)", puis tant que l'utilisateur ne demande pas l'arrêt du véhicule, ce dernier va être mis en mouvement par la fonction "deplacerVehicule()", qui sera réactualiser toutes les 5ms (comme le paramètre 5 de la fonction "start()" l'indique) par la fonction "update()". La fonction permettant le déplacement prend deux entiers en paramètres. Ces derniers correspondent aux pas de déplacement horizontal et vertical, et vallent donc 0 sur l'axe qui n'est pas concerné et "vitesse" sur l'autre. Le paramètre "vitesse" est modifiable par l'utilisateur s'il veut augmenter ou réduire la vitesse de mouvement de son véhicule. De plus, lorsque l'une des extrémités du véhicule rencontre un bord de la zone de composition, ce paramètre devient de signe contraire afin de permettre au véhicule de réaliser des aller-retours dans cette zone sans en sortir.
Dès lors que le premier déplacement a été demandé, à la possibilté de choisir l'autre déplacement, qui prendra effet à l'endroit où se trouve le véhicule au moment de la requête, s'ajoute une nouvelle option qui est l'arrêt de celui-ci. Pour ce faire, l'utilisateur doit cliquer sur le bouton **Arrêter**, ce qui appellera la fonction prédéfinie "stop()" du timer. Cette fonction ne stoppe pas véritablement le véhicule, elle arrête le timer et de ce fait la réactualisation du déplacement d'où un arrêt du véhicule.

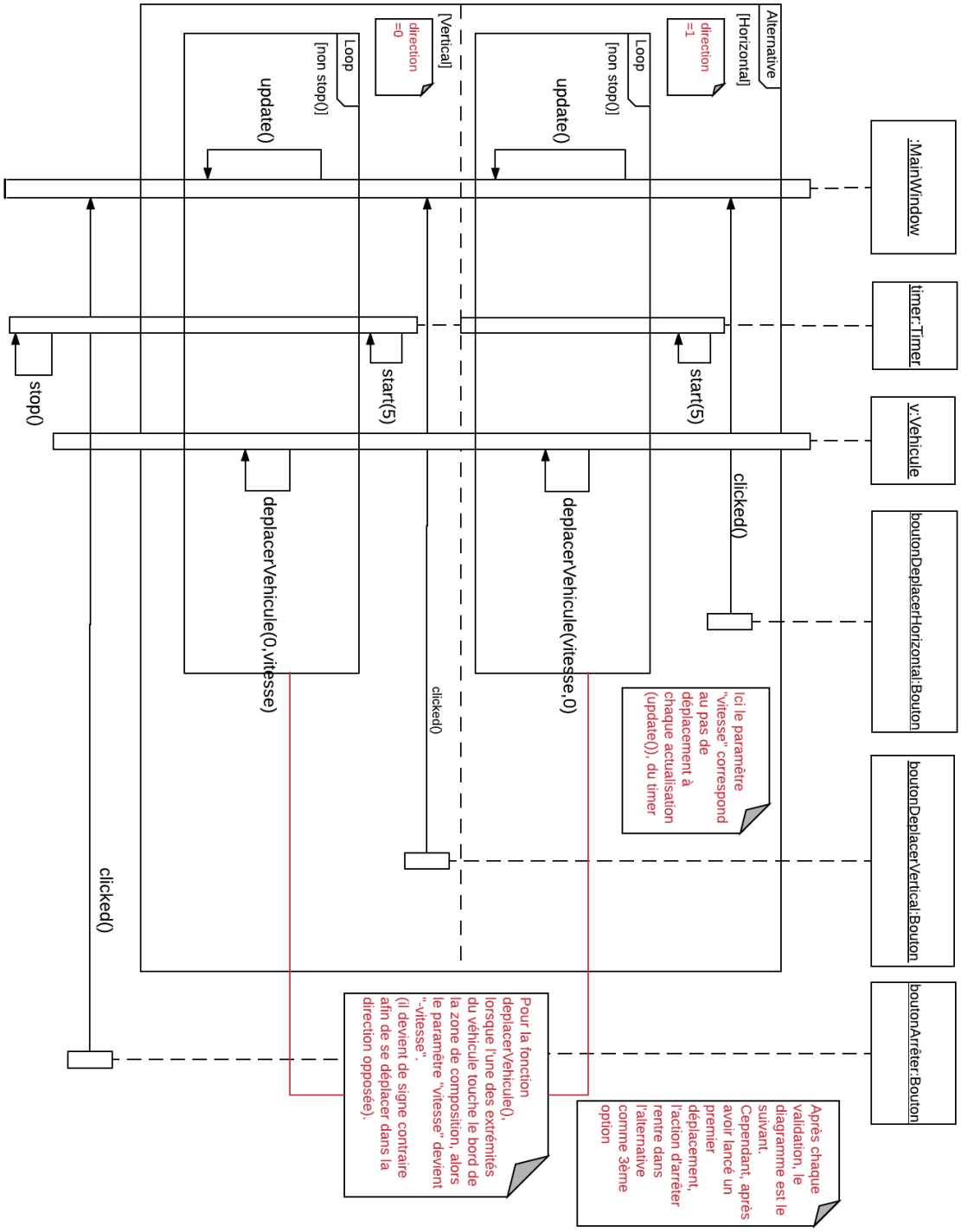

FIGURE 10 – Diagramme de séquence détaillé de Déplacer

II. Programmation du logiciel