

ENAC

# Dossier de conception intermédiaire

Réalisation d'une interface de calibration pour IMU de mini-drones dans le cadre du projet PAPARAZZI

Alinoé ABRASSART, Florent Gervais, Christophe Naulais,  
Guillaume Saas  
23/05/2013

## Table des matières

I.	Présentation du domaine.....	3
1.	Description du contexte.....	3
2.	Modèle du domaine .....	3
3.	Présentation des Use cases .....	4
II.	Description de l'application par ses Use Cases .....	6
1.	Use case magnétométrique .....	6
a.	Stockage des mesures .....	6
b.	Affichage et filtrage des mesures .....	7
c.	Guidage utilisateur .....	8
d.	Arrêt temporaire et résultat.....	9
e.	Continuer ou quitter.....	10
2.	Use case accéléromètre.....	12
3.	Start-Up et arrêt .....	13
a.	Démarrage de l'application .....	13
b.	Démarrage de la calibration.....	14
III.	Choix techniques issus de cette description .....	19
1.	Choix de conception d'interface graphique.....	19
2.	Choix de conception algorithmique.....	20
IV.	Points à confirmer pour le dossier de conception final .....	21
1.	Dossier des besoins à compléter.....	21
2.	Prototypage d'interface à terminer .....	21
3.	Algorithmique de l'affichage à confirmer.....	21
V.	Annexes.....	23
1.	Use Cases .....	23
a.	Use case magnétométrique .....	23
b.	Use case accéléromètres .....	24
2.	Prototypage papier d'interface .....	24
a.	Prototypage accueil .....	25
b.	Prototypage interface accéléromètre .....	26
c.	Prototypage interface magnétomètre .....	29

Figure 1 : Diagramme du domaine .....	3
Figure 2 : Classes du modèle conceptuel .....	4
Figure 3 : Décomposition des use cases en opérations.....	5
Figure 4 : Diagramme séquence stockage magnétométrique .....	6
Figure 5 : Diagramme classe stockage magnétométrique .....	7
Figure 6 : Diagramme séquence filtrage et affichage.....	7
Figure 7 : Diagramme classe filtrage et affichage.....	8
Figure 8 : Diagramme séquence d'affichage des consignes idem précédent.....	9
Figure 9 : Arrêt temporaire de la calibration .....	9
Figure 10 : Diagramme classe pause dans la calibration magnétométrique.....	10
Figure 11 : Diagramme séquence reprise de la calibration magnétométrique.....	10
Figure 12 : Diagramme séquence arrêt calibration.....	11
Figure 13 : Diagramme classe arrêt calibration.....	11
Figure 14 : Diagramme séquence filtrage et affichage de la calibration des accéléromètres .....	12
Figure 15 : Diagramme classe filtrage et affichage de la calibration des accéléromètres .....	12
Figure 16 : Diagramme séquence start-up.....	13
Figure 17 : Diagramme classe start-up .....	14
Figure 18 : Création en cascade.....	15
Figure 19 : Création avec classe intermédiaire.....	15
Figure 20 : Diagramme séquence de démarrage de la calibration.....	16
Figure 21 : Diagramme classe de démarrage de la calibration.....	17
Figure 22 : Diagramme séquence d'arrêt .....	17
Figure 23 : Diagramme de conception intermédiaire .....	18

## I. Présentation du domaine

Cette partie joue le rôle de résumé par rapport au dossier des besoins, dans l'hypothèse où ce dernier a été lu il n'est pas nécessaire de relire la présente partie. Dans le cas contraire, sa lecture est conseillée afin de pouvoir comprendre la logique sous-jacente à la réalisation du modèle conceptuel de l'application.

### 1. Description du contexte

Le projet vise à développer une partie du Framework PAPARAZZI dédiée à la calibration de certains des capteurs de calibration des drones. En effet, le Framework PAPARAZZI vise à développer un autopilote pour véhicule aérien léger et sans pilote (microdrones). Pour cela, il s'appuie sur toute une gamme de capteurs et notamment une centrale inertuelle (IMU). Cette IMU a besoin pour fonctionner d'être d'abord calibrée. Cette étape comprend la calibration des gyromètres, des magnétomètres et des accéléromètres. La procédure actuellement utilisée consiste à enregistrer un certain nombre de manipulations faites sur le drone en aveugle (sans retour à l'utilisateur sur la pertinence des manipulations effectuées) dans un fichier de log. Un script est ensuite appelé sur ce fichier de log et génère les paramètres de calibration dont on peut seulement alors juger de la pertinence avant de les conserver en les copiant dans le fichier adéquat (AirFrame du drone) ou de les rejeter ce qui force à reprendre toute la manipulation de calibration.

La difficulté pour l'utilisateur est donc essentiellement de faire les manipulations, il aurait donc besoin d'un feedback pour le guider au cours de ces manipulations afin de pouvoir être sûr, dès la prise des mesures, que le fichier généré sera correct. D'autre part, le gain d'audience du Framework fait qu'un nombre croissant d'utilisateurs novices cherchent à construire leurs propres drones. Le risque est qu'un certain nombre soit aujourd'hui rebutés par cette manipulation peu intuitive. L'application devra donc aussi s'intéresser à ces utilisateurs.

Pour un plus grand nombre de détails, il est recommandé de se référer à la présentation faite dans le dossier des besoins.

### 2. Modèle du domaine

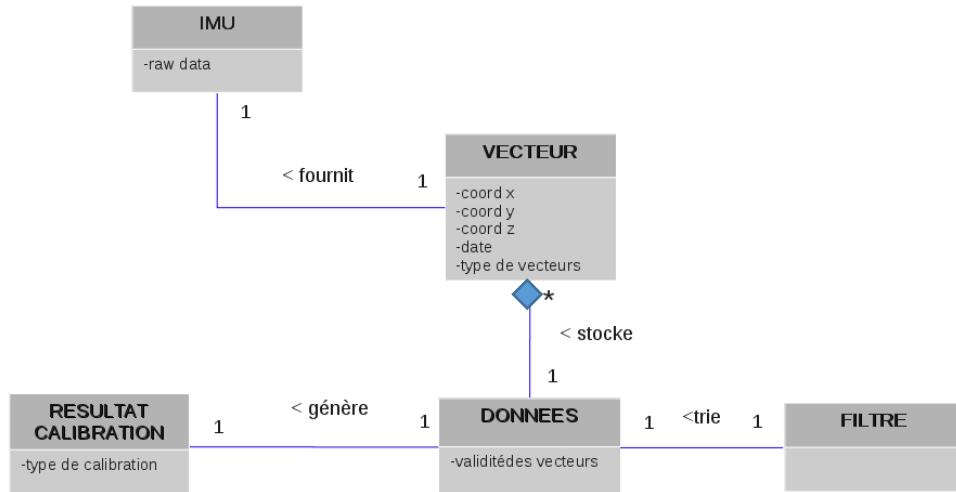


Figure 1 : Diagramme du domaine

<b>Classe</b>	<b>Fonction</b>	<b>Justification du format</b>
<b>IMU</b>	L'IMU fournit les données nécessaires à la calibration. C'est elle qui fait le lien entre l'architecture physique du système PAPARAZZI, c'est-à-dire le drone et l'ordinateur, et le logiciel que nous développons.	Une seule classe qui permet de faire l'abstraction de toute la partie non-logicielle, l'interface du système n'étant conceptuellement pas nécessaire. D'autre part il serait absurde de vouloir effectuer une calibration sans retour du système que l'on veut calibrer
<b>VECTEUR</b>	La classe Vecteur représente les données fournies par l'IMU.	Format au plus proche de la représentation réelle des vecteurs tridimensionnels qu'ils soient magnétique ou d'accélération.
<b>DONNEES</b>	Assure le stockage des vecteurs. Elle est l'expert d'information concernant les vecteurs.	Une classe générique pour le stockage. Comme on peut le voir dans les scénarios actuellement cette fonction est réalisée par le fichier de log. Cette classe n'est que l'abstraction de ce fichier auquel on ajoute des méthodes génériques de traitements de données. On pourra donc la traiter sous une forme quelconque.
<b>FILTRE</b>	La classe qui assure le filtrage des données afin de ne conserver que les vecteurs corrects	On aurait pu l'intercaler entre les vecteurs et données comme son nom l'indique mais le fait que DONNEES soit l'expert d'information concernant les vecteurs voudrait qu'on la place plutôt ici. De plus, il apparaît dans les scénarios que le traitement de filtrage se fait sur les données groupées.
<b>RESULTAT_DE_CALIBRATION</b>	Cette classe représente le résultat de la calibration. Elle contient les informations permettant à l'utilisateur de finaliser la calibration	Le choix le plus contestable. L'idée de conception est que les résultats ne doivent pas être reliés à DONNEES pour des raisons de cohésion. Pour cela, de la même façon que pour FILTRE, on crée une classe dédiée qui prend en charge ces opérations et la gestion du résultat. On gagne en plus en matière de couplage car ces résultats apparaissent de manière indépendante au DONNEES comme on pourrait s'y attendre en voyant les scénarios.

Figure 2 : Classes du modèle conceptuel

Le diagramme suivant et les classes sont issus des scénarios de travail que l'on peut trouver dans le dossier des besoins. Le nombre réduit de classes s'explique par une grande similarité des scénarios de ces derniers qui implique aussi un nombre limité de use cases.

### 3. Présentation des Use cases

En effet, les use cases sont au nombre de deux, un troisième pouvant voir le jour lorsque l'implémentation des deux premiers sera terminée. Les deux premiers uses case sont la calibration des magnétomètres et des accéléromètres, le troisième est celle des gyromètres. On pourra consulter les use cases en annexe. Pour chacun on distingue 4 phases décomposées en opérations génériques tant ils sont semblables. On reproduit ici cette décomposition dont la justification détaillée peut être trouvée dans le dossier des besoins.

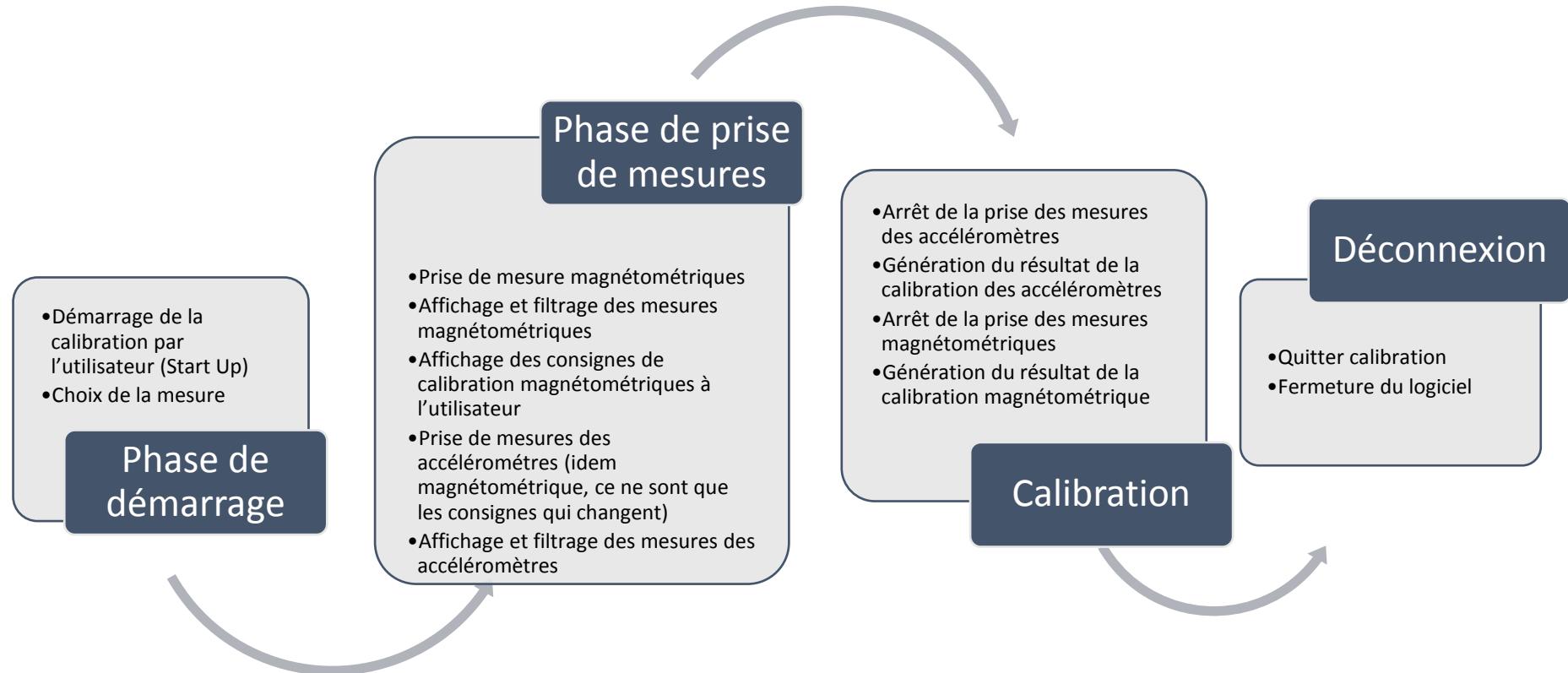


Figure 3 : Décomposition des use cases en opérations

## II. Description de l'application par ses Use Cases

Nous allons ici décrire la construction du diagramme de classe de l'application. Pour cela nous allons analyser use case par use case les différentes classes qu'ils impliquent. On a pu le voir dans le dossier des besoins, le nombre de use cases est pour l'instant limité à deux, la calibration magnétométrique et la calibration des accéléromètres. Pour pouvoir pleinement prendre conscience des implications de ces use cases, nous allons réutiliser la décomposition en opérations proposée dans le dossier des besoins. A chaque étape nous présenterons le diagramme séquence de l'opération puis le diagramme de classe associé. On aura pu le remarquer, les étapes sont extrêmement semblables pour ce qui est du démarrage et de l'arrêt. On ne traitera ces étapes qu'à la fin. Les noms des classes ont été anglicisés pour faciliter la maintenance future du système. Pour les deux premiers use case, on considère que le diagramme du domaine est disponible et fonctionnel, que la calibration a été choisie et que le logiciel a reçu une action de l'utilisateur lui indiquant de démarrer la prise de mesures.

### 1. Use case magnétométrique

#### a. Stockage des mesures

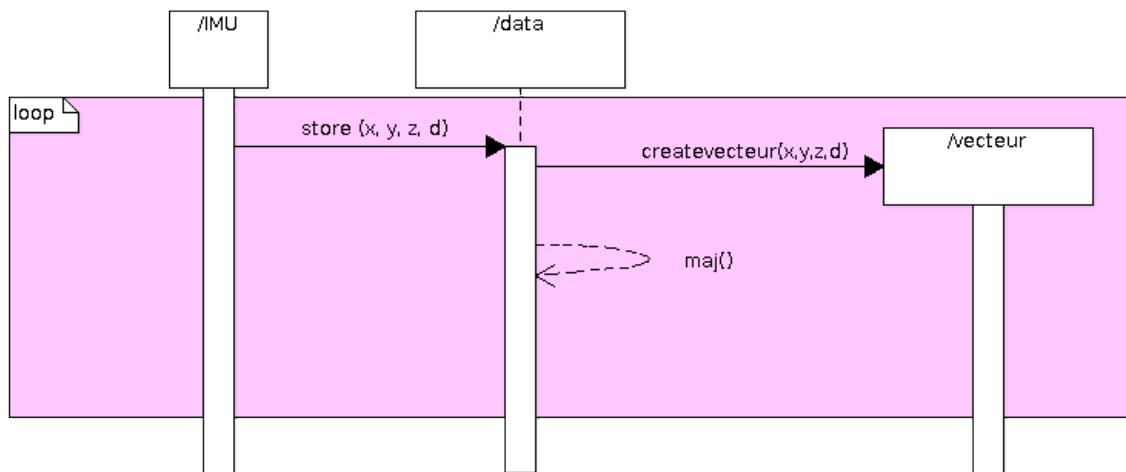


Figure 4 : Diagramme séquence stockage magnétométrique

L'opération d'ajout d'un vecteur dans DATA se fait donc logiquement suite à un appel d'IMU qui est l'expert en information sur les données des vecteurs (Pattern expert). En vertu du pattern de faible couplage (FaC dans la suite), la création des vecteurs est déportée dans DATA qui assume toutes les tâches de stockage permettant ainsi qu'IMU ne dépende plus du format de stockage. Le diagramme de classe se trouve donc un peu modifié pour prendre la forme suivante :

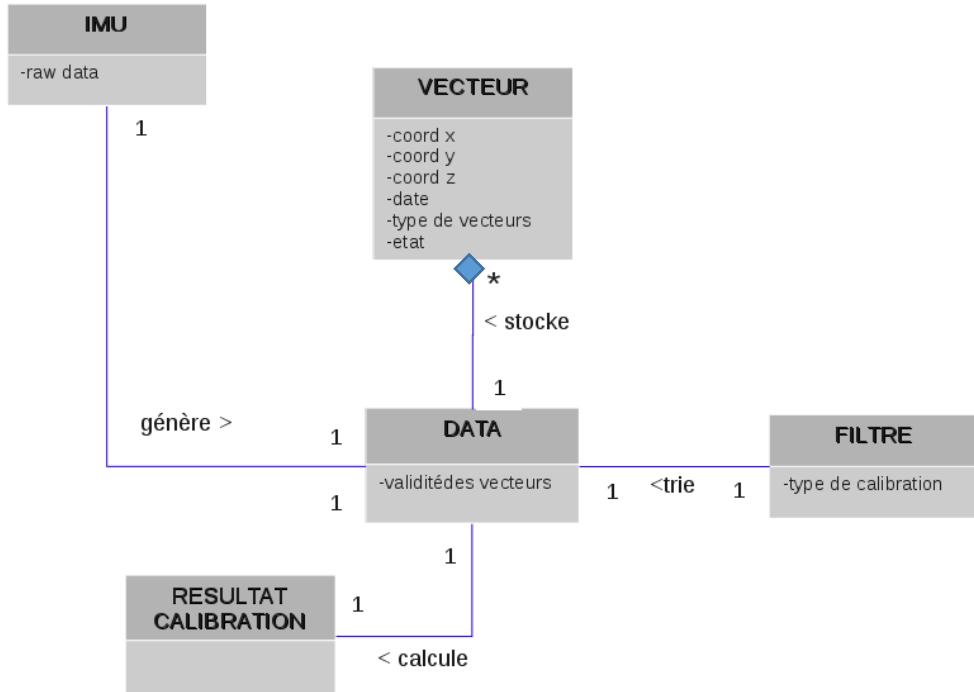


Figure 5 : Diagramme classe stockage magnétométrique

### b. Affichage et filtrage des mesures

A chaque ajout de vecteur, on veut que les données soient filtrées en accord avec les nouvelles informations que ce dernier apporte. Pour cela, on fait déclencher par DATA un filtrage sur les données qu'il contient.

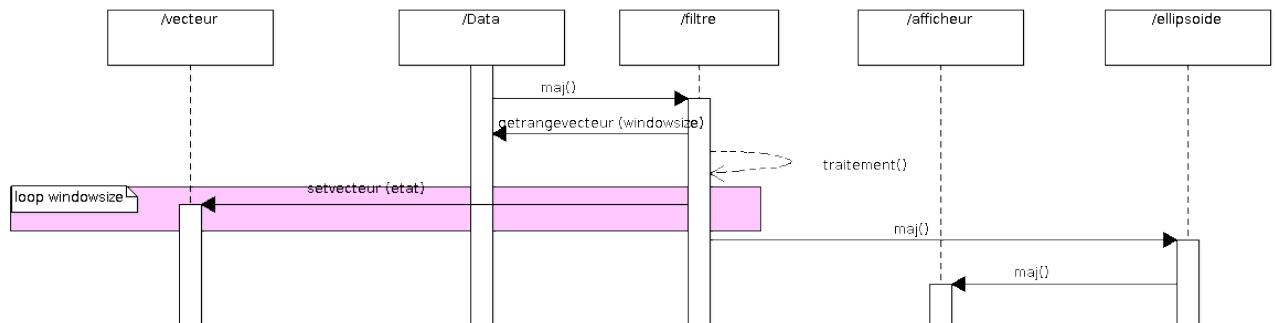


Figure 6 : Diagramme séquence filtrage et affichage

On le voit, la tâche de filtrage déclenchée par DATA se fait via une classe dédiée. Celle-ci présente l'intérêt de garder la forte cohésion (FCo) de DATA. Cette nouvelle classe se trouvera de plus indépendante de la structure utilisée pour le stockage tant que celle-ci implémentera une interface que l'on pourra nommer filtrable. FILTRE assure le filtrage et déclenche la mise à jour de l'affichage. Cet affichage est distinct de filtrage puisque réalisant des fonctionnalités différentes et théoriquement

indépendantes (FaC, FCo). On introduit la classe ELLIPSOIDE qui se chargera de générer le modèle de la vue dont le prototypage peut être consulté en Annexe 2. On voudra de plus que cet affichage se fasse à l'utilisateur. On créée donc un classe qui assure l'affichage sur demande de la classe précédente afin d'être en accord avec le pattern model-view-controler (MVC) dans lequel elle assurera le rôle de vue. Ce pattern étant introduit, on remarque alors qu'IMU tient le rôle de contrôle puisqu'il reçoit ses instructions depuis un élément extérieur à l'application qui est la centrale inertuelle du drone.

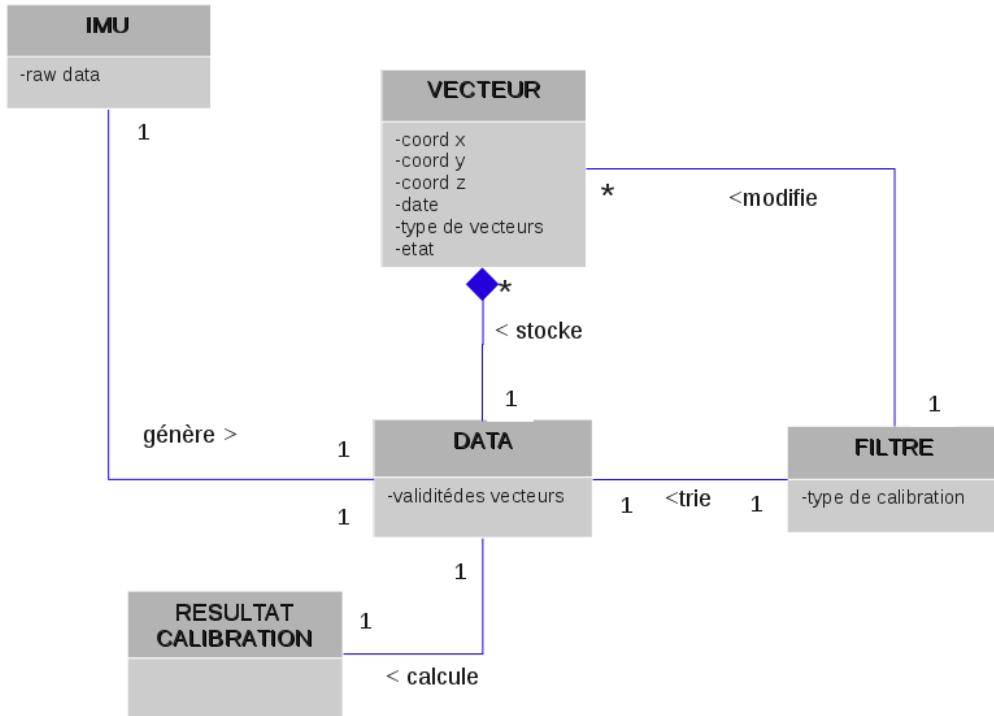


Figure 7 : Diagramme classe filtrage et affichage

### c. Guidage utilisateur

Il est aussi demandé d'afficher des consignes afin de guider l'utilisateur. La façon dont celles-ci sont gérées dépend du choix quant à la forme du retour de la prise des mesures. L'expert en information n'est donc pas DATA comme on aurait pu s'y attendre mais plutôt ELLIPSOIDE qui sera donc chargé de ce retour. On constate alors que les fonctions de retour et d'affichage visant à préparer la vue se retrouvent dans la même classe (FCo). On a alors deux options qui diffèrent par leur forme, l'un propose un retour immédiat tandis que l'autre un retour différé. Ces deux types de retour sont en prototypage papier afin d'être soumis à l'utilisateur avant de passer à un éventuel prototypage logiciel (voir IV.2 ci-dessous)

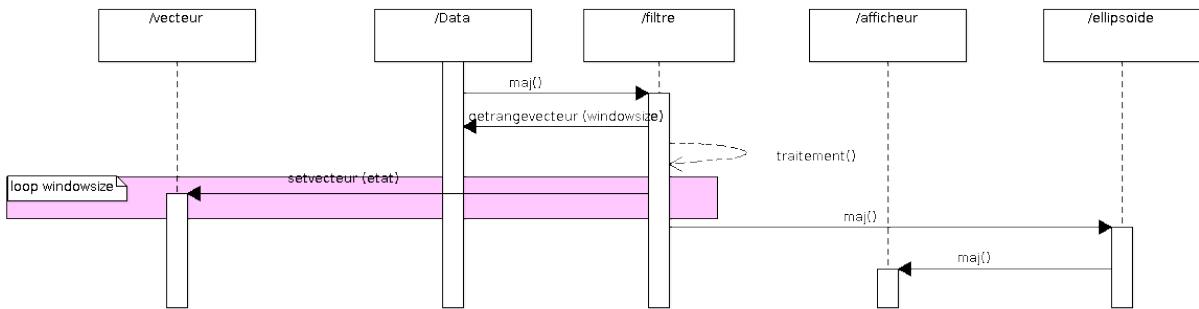


Figure 8 : Diagramme séquence d'affichage des consignes idem précédent

Le diagramme classe est donc identique au précédent.

#### d. Arrêt temporaire et résultat

L'utilisateur peut alors choisir d'arrêter les mesures pour générer un résultat de la calibration effectuée jusqu'alors ou faire une pause dans la calibration avant de poursuivre.

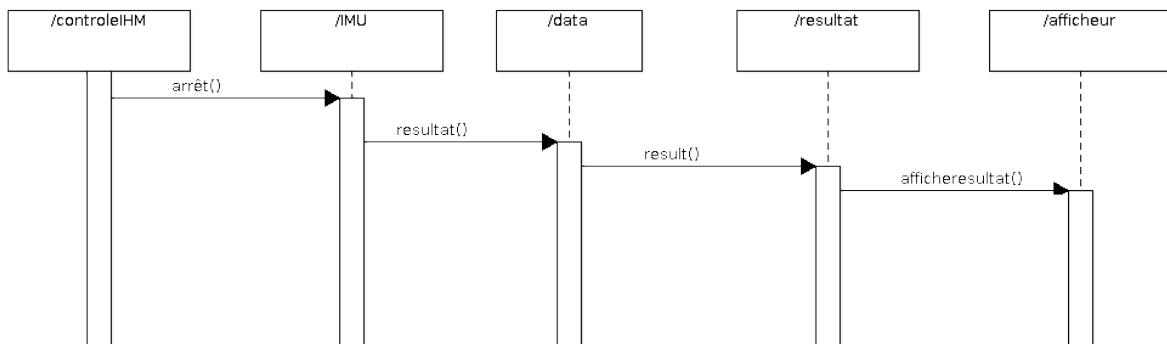


Figure 9 : Arrêt temporaire de la calibration

Cela impose donc que l'utilisateur puisse agir sur le logiciel via une autre interface que l'IMU afin de pouvoir signaler qu'il veut faire une pause. Via le pattern MVC, on constate que cela nécessiterait qu'IMU puisse détecter des inputs issu d'une interface homme machine(IHM) distincte de l'IMU via le clavier ou la souris. En raison des patterns de forte cohésion et de faible couplage, on préfère créer une nouvelle classe dédiée au recueil de telles interactions : CONTROLEUR\_IHM. La classe RESULTAT doit pouvoir demander l'affichage du résultat, on fait donc apparaître son lien avec AFFICHEUR. On en déduit le diagramme de classe suivant :

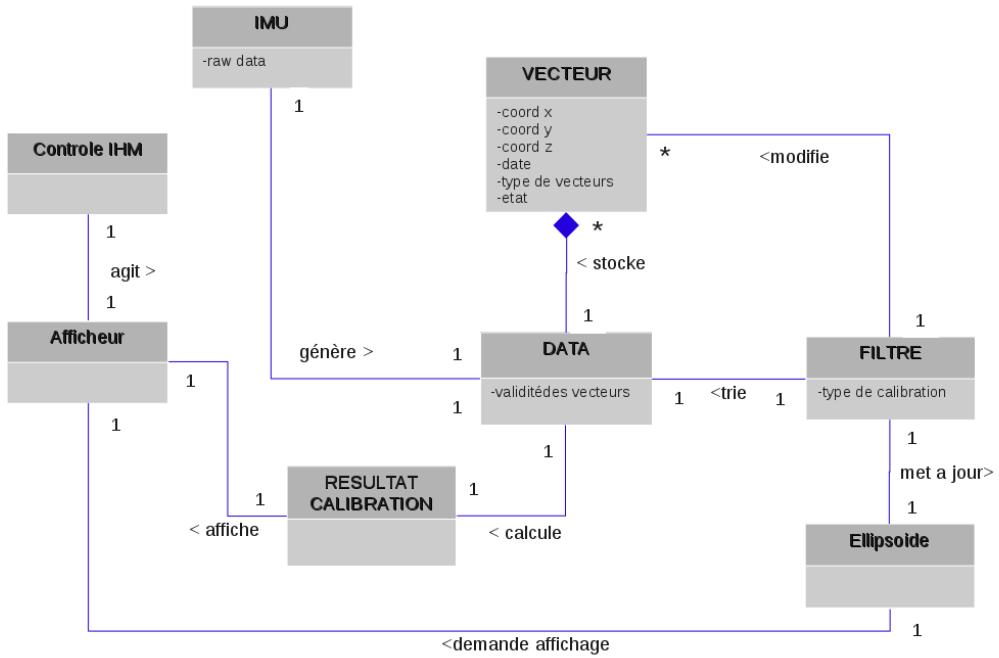


Figure 10 : Diagramme classe pause dans la calibration magnétométrique

#### e. Continuer ou quitter

Une fois le résultat intermédiaire généré, deux possibilités sont encore possibles pour l'utilisateur, d'une part reprendre la calibration et d'autres part, l'arrêter. Dans le premier cas, il suffit de relancer IMU :

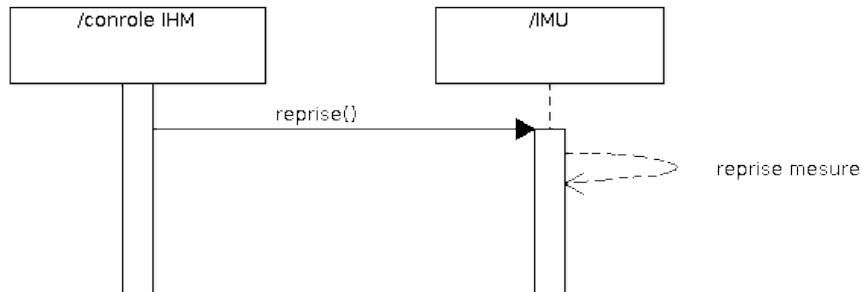


Figure 11 : Diagramme séquence reprise de la calibration magnétométrique

Le diagramme de classe reste inchangé, toutes les relations nécessaires à la reprise sont les mêmes que celles de l'arrêt, aucune classe n'ayant été détruite lors de l'arrêt temporaire.

Il peut aussi décider d'arrêter la calibration en se satisfaisant ainsi des résultats obtenus. Le diagramme séquence associé est alors le suivant :

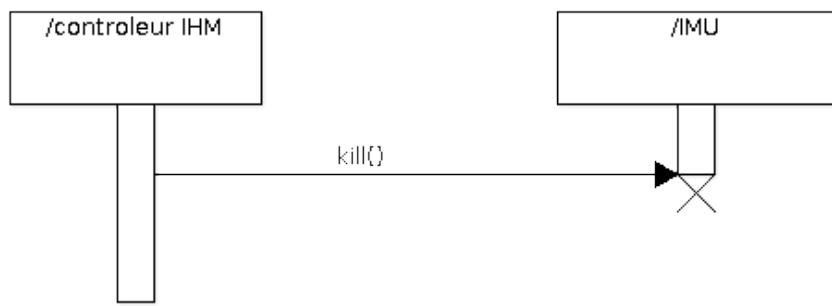


Figure 12 : Diagramme séquence arrêt calibration

L'arrêt se fait via une action de l'utilisateur. Puisque pour le moment seul IMU a une poignée sur le reste de l'application, on peut se contenter de détruire IMU. Le garbage collector gère ensuite la destruction des autres classes associées.

Le diagramme de conception obtenu suite à cette première phase de dessin UML est le suivant :

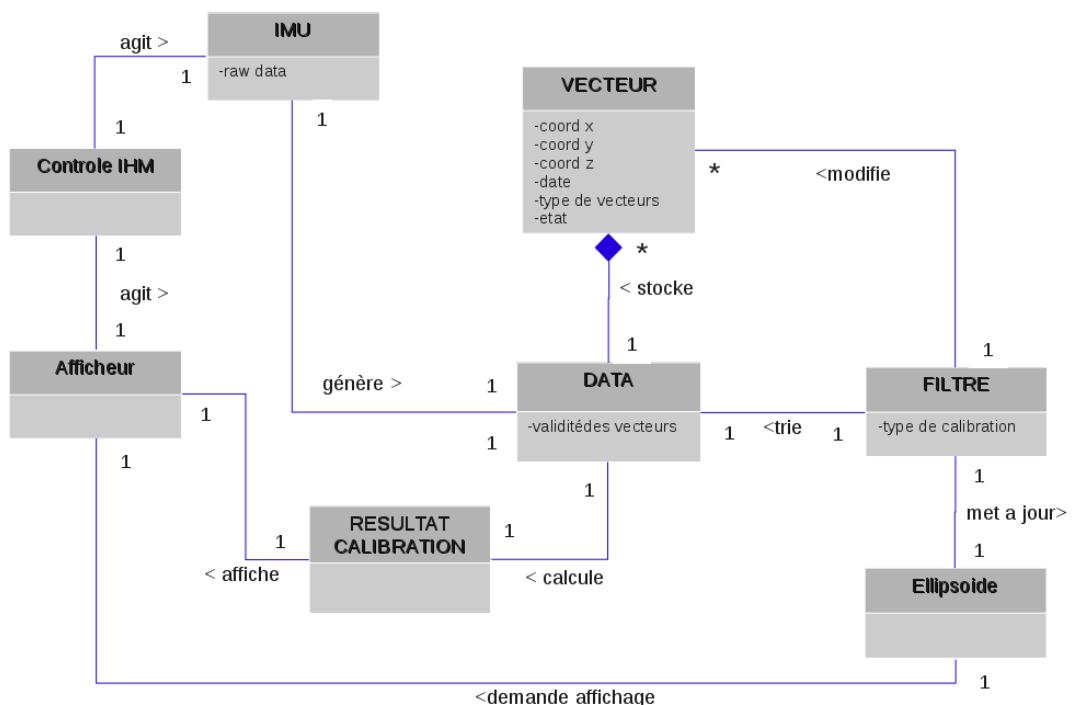


Figure 13 : Diagramme classe arrêt calibration

## 2. Use case accéléromètre

On peut alors considérer les mêmes étapes pour la calibration des accéléromètres. L'opération de stockage sera la même, la seule difficulté est que DATA sache quel type de vecteurs elle doit stocker mais ce problème est à résoudre au moment du start-up. Le use case du stockage n'est donc pas à revoir ici. En revanche, la visualisation ne sera pas la même. Pour cela on voudra à nouveau filtrer les vecteurs mais de manière différente et générer un affichage différent de celui de la calibration magnétométrique.

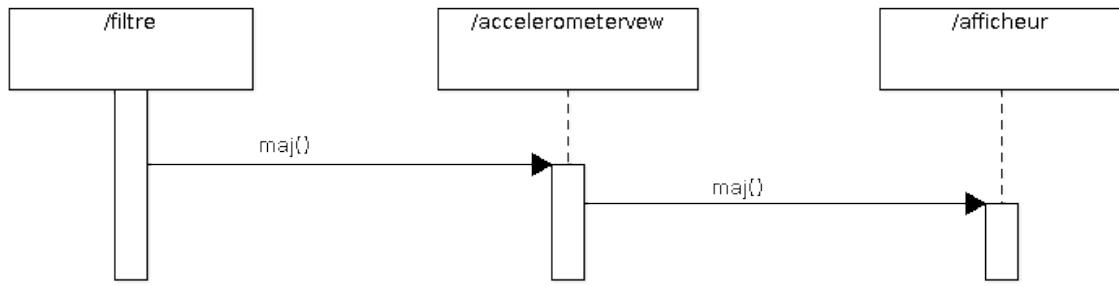


Figure 14 : Diagramme séquence filtrage et affichage de la calibration des accéléromètres

On va donc créer une nouvelle classe qui viendra tenir le rôle d'ELLIPSOIDE dans le cas de la calibration des accéléromètres en invoquant les mêmes patterns que précédemment : FCo et FaC. Cette classe sera de nouveau reliée à AFFICHEUR pour respecter le pattern MVC. On la nommera ACCELEROMETRE\_VIEW.

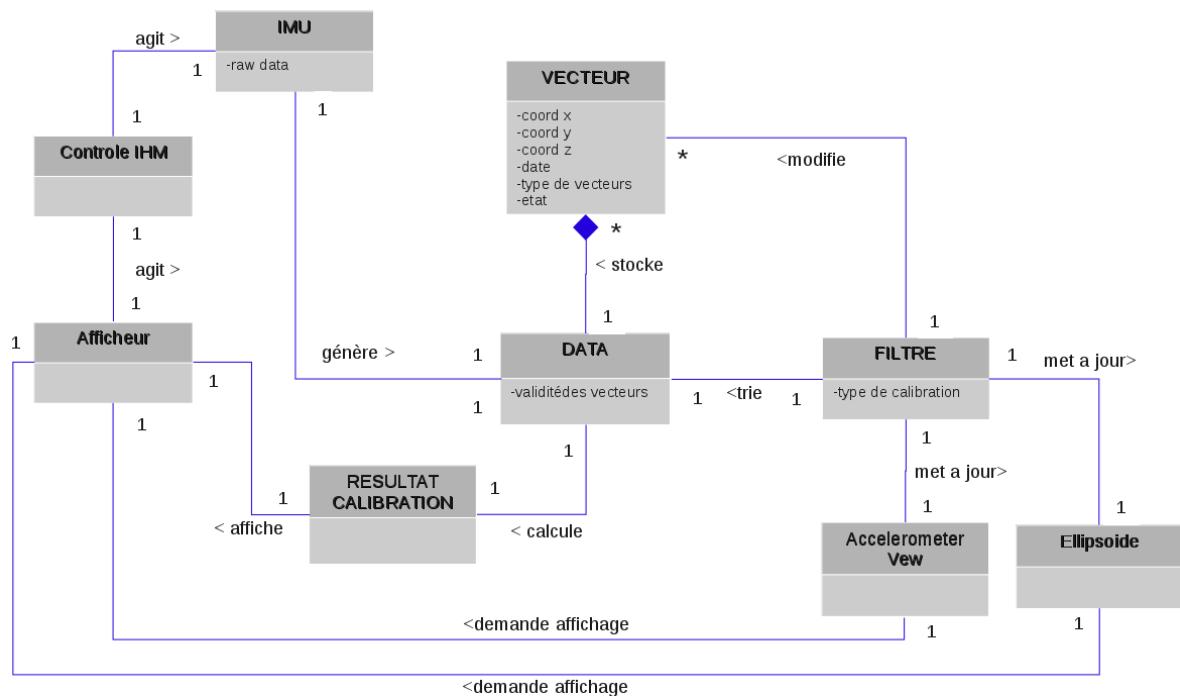


Figure 15 : Diagramme classe filtrage et affichage de la calibration des accéléromètres

A nouveau se pose le problème de la gestion de l'arrêt et de la reprise de la calibration. Puisque les diagrammes de classe sont très similaires comme on peut le constater en Figure 5 et Figure 12, on

s'attend à ce que les fonctionnalités d'arrêt puissent être gérées de la même façon. En effet, l'arrêt du recueil de données par IMU suffit à ce que la totalité de la chaîne s'arrête en commandant l'affichage du résultat par DATA. A nouveau, rien n'est ici à modifier.

Les deux use cases principaux étant désormais en place, il convient de s'intéresser au démarrage de l'application et à son arrêt.

### 3. Start-Up et arrêt

On l'a vu aussi bien dans les scénarios de travail que dans les use cases le démarrage d'une calibration se déroule en deux temps, le lancement de l'application permettant de choisir la calibration que ce soit une console ou une interface dédiée puis le démarrage de la calibration elle-même. Il convient donc de distinguer convenablement ces deux temps de démarrage. Pour cela on va s'appuyer sur le pattern MVC qui a guidé en grande partie la conception jusqu'alors.

#### a. Démarrage de l'application

Après le démarrage de l'application, l'utilisateur doit pouvoir choisir la calibration qu'il souhaite effectuer. Pour cela, nous avons besoin de deux classes, une informant l'utilisateur qu'il dispose d'un choix et une seconde qui récupère le choix de l'utilisateur. Par rapport à la conception effectuée jusqu'alors, il semble logique d'utiliser l'AFFICHEUR en accord avec le pattern FCo puisque cette classe se charge déjà d'afficher toutes les informations à l'utilisateur. C'est CONTROLE\_IHM qui ordonnera à AFFICHEUR d'effectuer cet affichage puisque c'est lui qui est chargé de récupérer les ordres de l'utilisateur en provenance du clavier et de la souris. Ainsi en vertu du pattern créateur, c'est CONTROLE\_IHM qui va se charger de lancer AFFICHEUR. Il ne semble donc pas nécessaire d'avoir, pour le moment, une classe dédiée au démarrage de l'application.

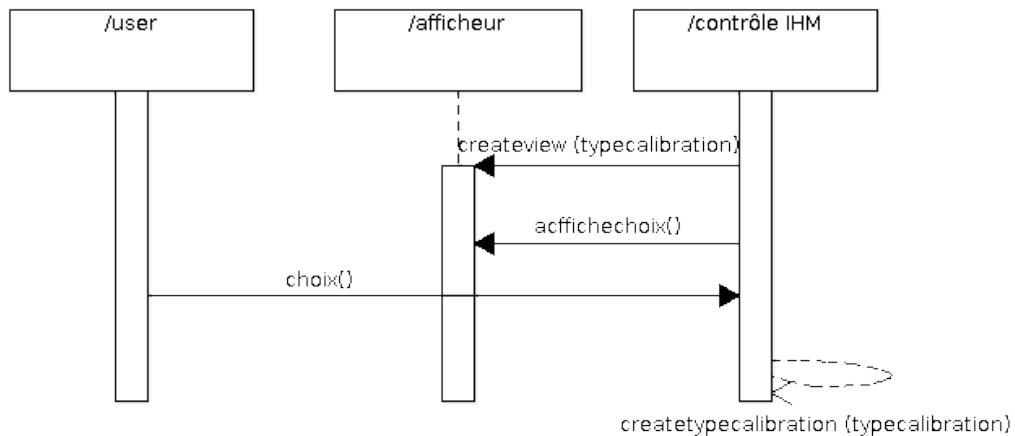


Figure 16 : Diagramme séquence start-up

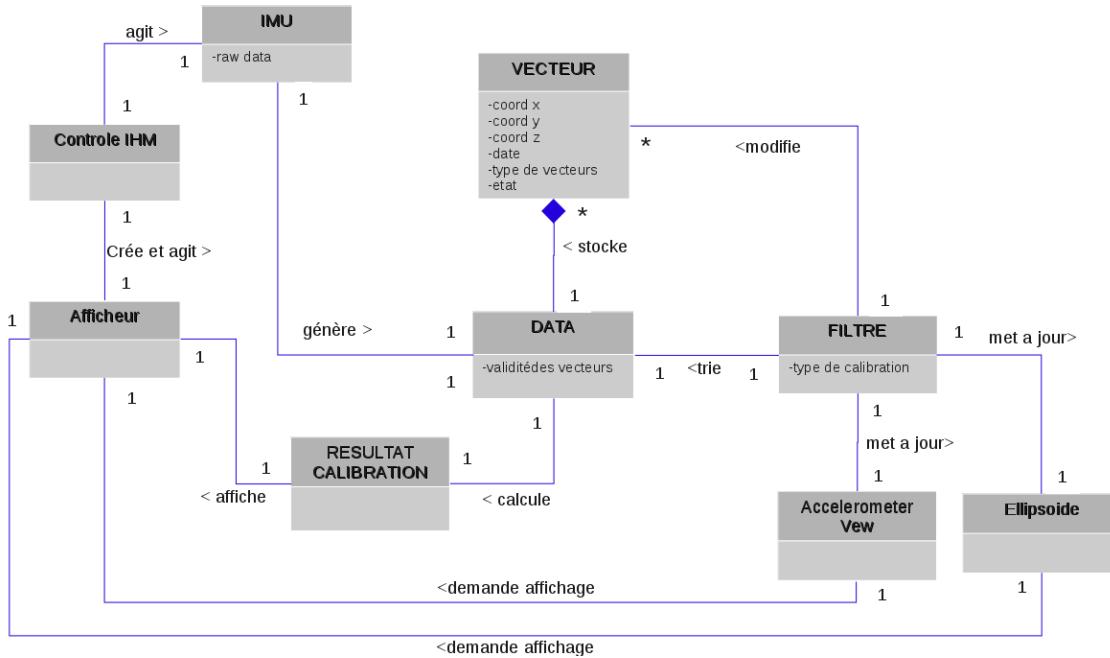


Figure 17 : Diagramme classe start-up

### b. Démarrage de la calibration

Après le choix de l'utilisateur doit apparaître la fenêtre de calibration prête à recevoir les données sur l'input approprié de l'utilisateur. Pour cela il convient de créer les classes nécessaires soit toutes les classes du diagramme dans le mode de calibration approprié. De plus, on ne doit créer qu'ELLIPSOIDE ou ACCELEROMETRE\_VIEW pour faire l'affichage adéquat. C'est clairement CONTROLE\_IHM l'expert en information sur le type de calibration choisi, cependant lui faire créer toutes les classes qui dépendent de cette information est peu pertinent au vu du diagramme de classe. De plus faire une création en cascade forcerait toutes les classes à connaître l'afficheur utilisé puisque les classes qui apparaissent au bout des branches telles RESULTAT et ELLIPSOIDE doivent être rattachées à AFFICHEUR comme on peut le voir sur le diagramme suivant :

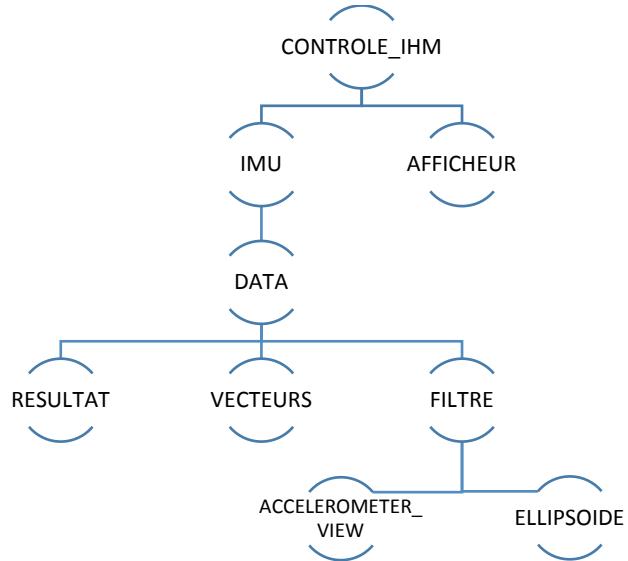


Figure 18 : Crédit en cascade

On préférera donc une troisième option consistant à créer temporairement une nouvelle classe permettant de créer l'ensemble du modèle avec les liens avant de les laisser interagir seuls.

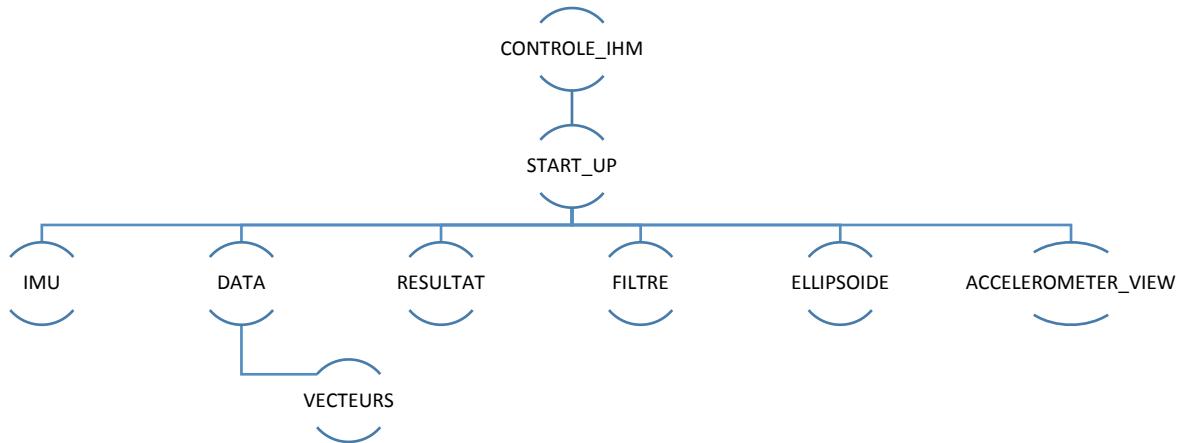


Figure 19 : Crédit avec classe intermédiaire

Cette dernière méthode permet, comme on le voit sur le graphique ci-dessus de créer facilement les liens avec pour seul coût supplémentaire la création d'une nouvelle classe. Un objet de ce type n'aura plus de raison d'exister après cette phase d'initialisation et pourra donc être détruit. On obtient alors le diagramme d'initialisation suivant :

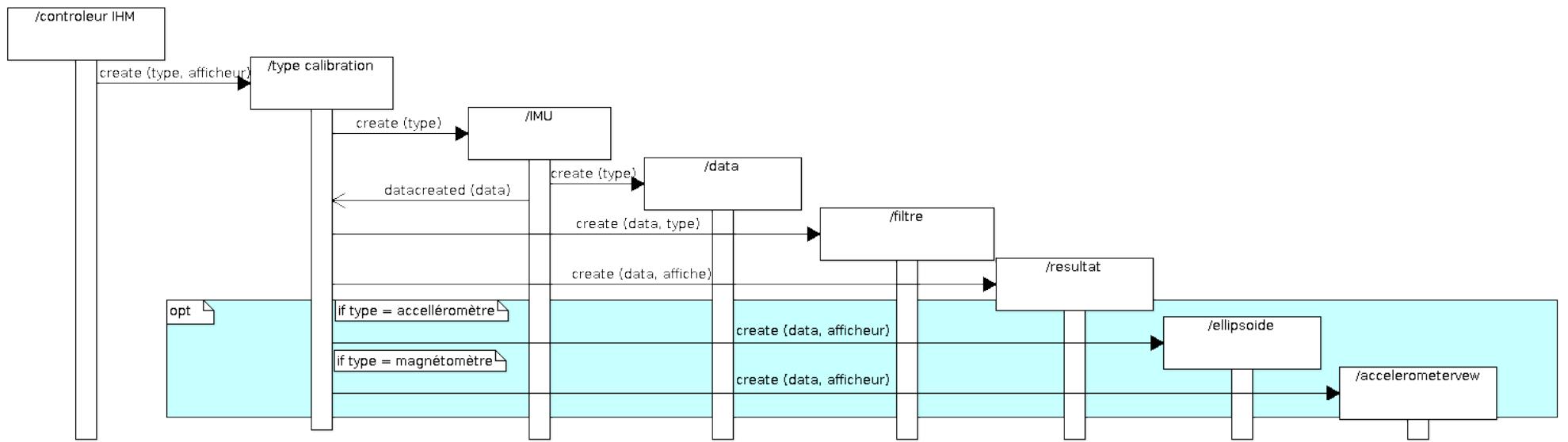


Figure 20 : Diagramme séquence de démarrage de la calibration

Le diagramme classe s'en trouve modifié de la façon suivante :

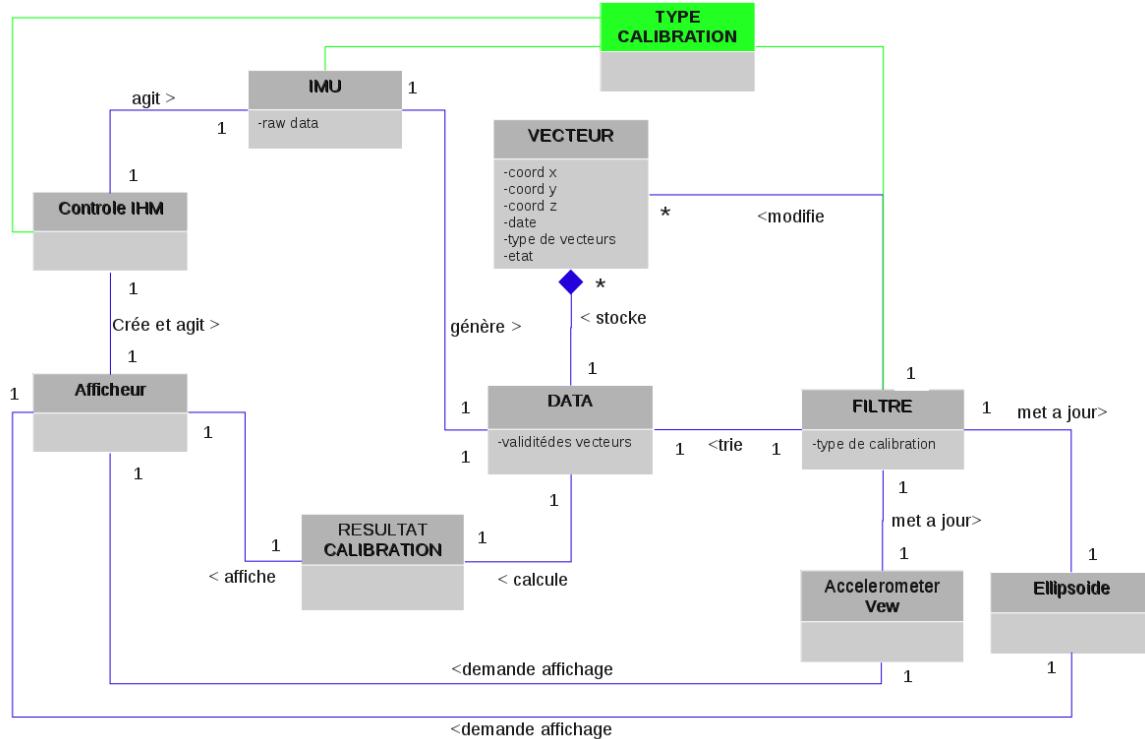


Figure 21 : Diagramme classe de démarrage de la calibration

Il ne reste plus qu'à gérer l'arrêt de l'application quand l'utilisateur veut quitter l'application. Sur action de l'utilisateur, on ramène l'affichage à l'écran de choix de calibration où il pourra choisir de quitter l'application.

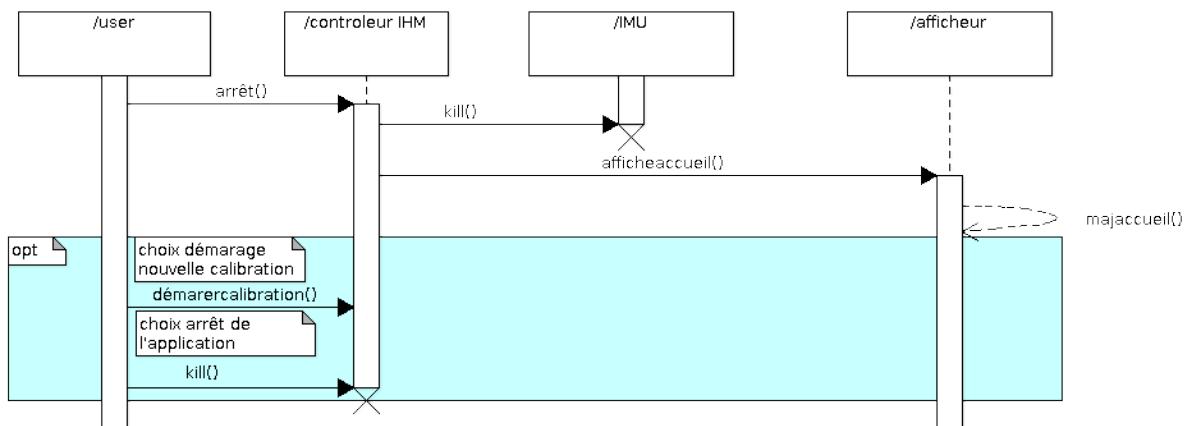


Figure 22 : Diagramme séquence d'arrêt

A nouveau, en détruisant IMU, on provoque une destruction en cascade des autres classes du domaine hors CONTROLEUR-IHM et AFFICHEUR.

Aucune modification n'est à apporter au diagramme précédent par rapport à cette nouvelle opération ; on obtient donc le diagramme suivant en fin de conception :

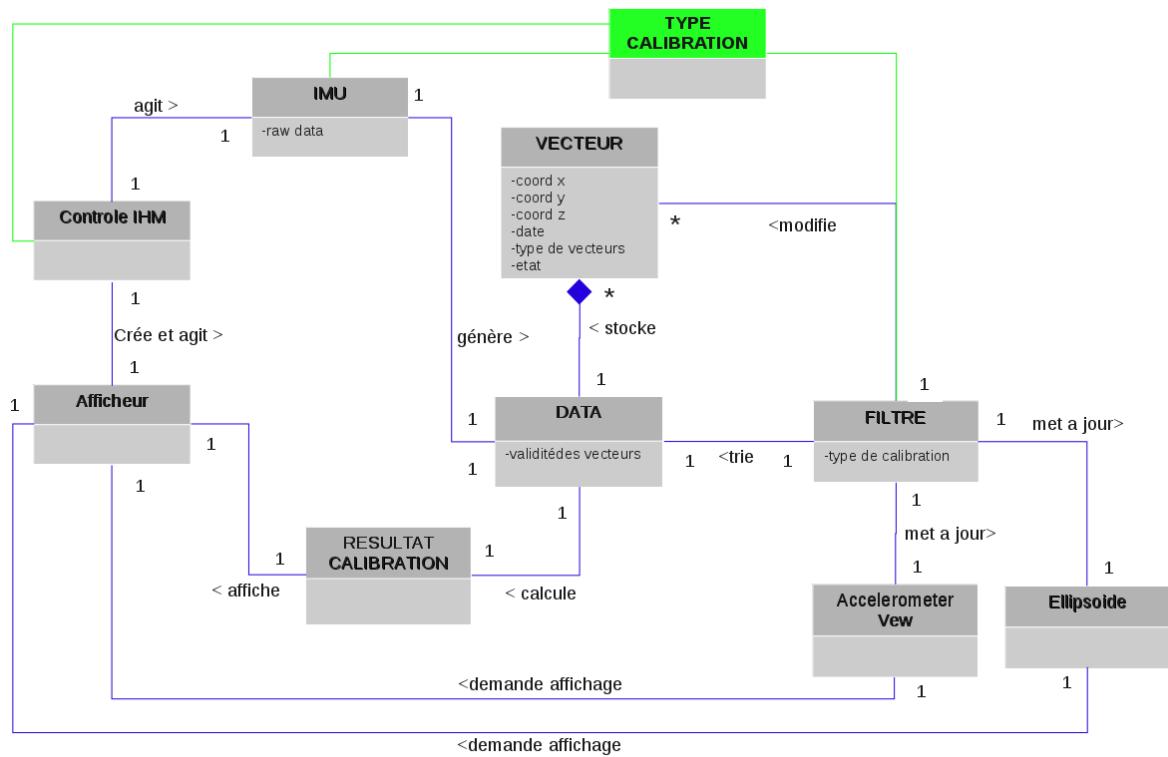


Figure 23 : Diagramme de conception intermédiaire

### III. Choix techniques issus de cette description

#### 1. Choix de conception d'interface graphique

Pour les choix de conception d'interface graphique, on l'a vu quatre classes sont importantes, CONTROLEUR\_IHM, AFFICHEUR, ELLIPSOIDE et ACCELEROMETER\_VIEW. Si les deux premières sont assez génériques puisque communes à quasiment toutes les applications exploitant une interface graphique, les secondes nécessitent des développements spécifiques. Il convient que leur conception soit très clairement adaptée du tableau final du dossier des besoins qui détaille les différents points requis pour ces interfaces.

Concernant la démarche adoptée, elle reste similaire à celle proposée dans le dossier des besoins. Un prototypage papier permet d'écrire les idées, un prototypage logiciel probablement en Java avec la librairie graphique finale afin de la prendre en main, permet de raffiner ces idées avant de ne sélectionner que la meilleure et de l'adapter à l'architecture spécifique du logiciel. Il convient donc de déterminer quelques choix de librairies potentielles qui peuvent permettre une mise en place de ces interfaces

De nombreuses librairies graphiques de Java existent et possèdent leurs propres domaines d'applications. Ainsi, deux librairies spécifiques vont être choisis : Java 3D qui vise la synthèse d'image 3D en se basant sur des graphes de scènes, puis JFreeChart qui permet de créer des graphiques et des diagrammes de bonne qualité. Ces choix ont été faits car ces deux librairies sont gratuites, de bonne qualité et enfin faciles d'utilisation. Il nous reste plus qu'à choisir une librairie graphique pour la base de notre interface (les widgets ...) et là beaucoup de choix s'offrent à nous. Pour pouvoir faire un choix constructif nous allons faire un récapitulatif des avantages et inconvénients de trois librairies qui ont retenu notre attention : SWT, SWING, QtJambi.

	SWT	SWING	QtJambi
Portabilité	-	++	+
Rapidité	++	-	-
Consommation en ressources machine	+	-	-
Esthétisme	+	-	+
Simplicité	-	++	+

La rapidité et la consommation en ressources machine sont des éléments primordiaux pour la conception de notre interface, c'est pourquoi notre choix va se porter sur la librairie graphique SWT. La difficulté de prise en main inhérente à SWT risque de rendre la phase de prototypage plus longue. De plus les choix liés au design de l'interface sont loin d'être arrêté, de nombreux retours vers l'utilisateur seront donc nécessaires afin d'obtenir une interface aussi adaptée que possible aux besoins réels.

## 2. Choix de conception algorithmique

Concernant les choix algorithmique, il apparaît par rapport au plan de projet qu'une certaine partie des algorithmes va devoir être recodée. En effet, la partie filtrage ne peut plus s'appuyer sur l'ensemble des valeurs comme elle le fait actuellement. Si l'algorithme de calibration peut-être conservé, il va falloir trouver un système permettant d'exploiter le code déjà écrit dans un autre langage.

Pour la calibration finale des mesures, nous allons regarder les possibilités d'utiliser le script de calibration existant. Le logiciel existant est un script en langage Python qui prend toute les mesures dans un fichier log, les filtre, calcule les paramètres de calibration, et génère une des vues des mesures. Notre interface devra aussi calculer les paramètres de calibration ainsi que générer des vues des mesures donc nous avons regardé si nous pouvions utiliser le script Python ou une partie du script pour notre interface. Il existe cependant quelques différences:

	<b>Script de calibration existent</b>	<b>Interface du projet</b>
<b>Langage</b>	Python	Java
<b>Tâches</b>	Filtrage, Calibration et vue effectué après la prise de mesure	Filtrage et vue effectué en temps réel et la calibration après la mesure.
<b>Données</b>	Données utilisées pour le filtrage sont issues d'un fichier log	Données utilisées pour la vue et le filtrage sont acquises en temps réel. Calibration?

Nous avons donc trois options pour utiliser le script Python dans notre logiciel de calibration.

1. Recoder le script (ou une partie du script) Python en langage Java compatible avec le reste du logiciel.
2. Utiliser une librairie qui permet d'effectuer un binding java – Python (Jython) qui est capable de lire le script Python à partir de classe Java.

Pour déterminer quelle méthode on utilisera nous avons pesé les avantages et les inconvénients des différentes méthodes dans un tableau.

	1	2
Charge de Travail (codage)	--	+
Charge cpu (rapidité)	++	-
Intégration au système (facilité à intégrer avec les classes existantes)	++	--
Utilisation client (téléchargement d'une librairie ou d'un programme de plus)	++	-
Maintenance, (complexité du programme)	+	-

Le filtrage et la vue doivent être effectuées en temps réel et c'est un processus interactif, le facteur le plus important est donc la rapidité du logiciel. Il est donc judicieux de recoder cette partie en langage Java, même si cela engendre plus de travail pour nous.

La calibration après la prise de mesure n'a pas de contrainte de rapidité. Nous voulons donc privilégier l'intégration et la maintenance de cette partie du logiciel, ainsi que la facilité d'usage pour le client.

En fonction des avantages et des inconvénients identifiés ci-dessus, nous allons recoder le script Python en Java pour satisfaire les besoins du logiciel.

On cherchera aussi au vu des besoins à rendre la classe FILTRE aussi générale que possible de façon à pouvoir la réutiliser pour les deux types de filtrage. Il convient de se renseigner sur les différents types d'algorithme possibles (sliding window...). Il faudra ensuite déterminer s'il convient d'implémenter nous même le script ou d'utiliser une librairie adaptée.

Creation d'un filtre, ou off the shelf.

## IV. Points à confirmer pour le dossier de conception final

### 1. Dossier des besoins à compléter

Le dossier des besoins est pour le moment incomplet, les résultats du sondage en ligne vont permettre de compléter les scénarios d'usage futurs. Cette méthode de collecte de données utilisateur non mentionnée dans le plan de projet devrait apporter son lot de modifications quant aux besoins.

D'autre part, une relecture de ce dossier par le client est encore à effectuer afin de juger de son adéquation aux attentes de ce dernier.

### 2. Prototypage d'interface à terminer

Le prototypage papier d'interface est en cours de validation, les premiers essais sont en ANNEXE 2. Un rendez-vous va être pris avec M. Hattenberger afin de valider et de modifier ces idées avant de lancer le prototypage logiciel à l'aide des librairies graphiques vues ci-dessus.

Cette phase va nous permettre de préciser les choix de conceptions quant à l'interface graphique car ceux-ci restent vagues pour le moment.

De même la manière de vérifier l'adéquation d'une interface par rapport aux besoins est encore à définir.

### 3. Algorithmique de l'affichage à confirmer

Toute la partie de l'affichage est à soumettre à des tests afin de vérifier sa capacité d'implémentation. Il n'est en effet pas évident que l'architecture retenue convienne aux besoins en terme de temps de réponse, ce point est à confirmer par le prototypage. Ainsi deux chaînes d'affichage vont être momentanément maintenues en parallèle afin de juger celle qui donne les meilleurs résultats une avec rafraîchissement à chaque ajout de point et modification du fond par groupe de points, le nombre de points nécessaires restant à définir et une autre où ces deux opérations s'effectuent en simultané. Une amélioration en terme de temps de réponse peut-être attendue de la disparition d'un des appels d'affichage dans la deuxième version mais une dégradation de l'expérience utilisateur dû à l'absence d'un retour immédiat est à prévoir. La première risque quant à elle de faire apparaître un retard dû à un trop grand nombre d'appels à l'affichage. Les deux options sont donc à considérer en attendant de voir quelle chaîne donne le meilleur résultat.

Le diagramme classe ci-dessous fait apparaître en rouge la deuxième possibilité ne nécessite pas de changement du diagramme classe de conception.

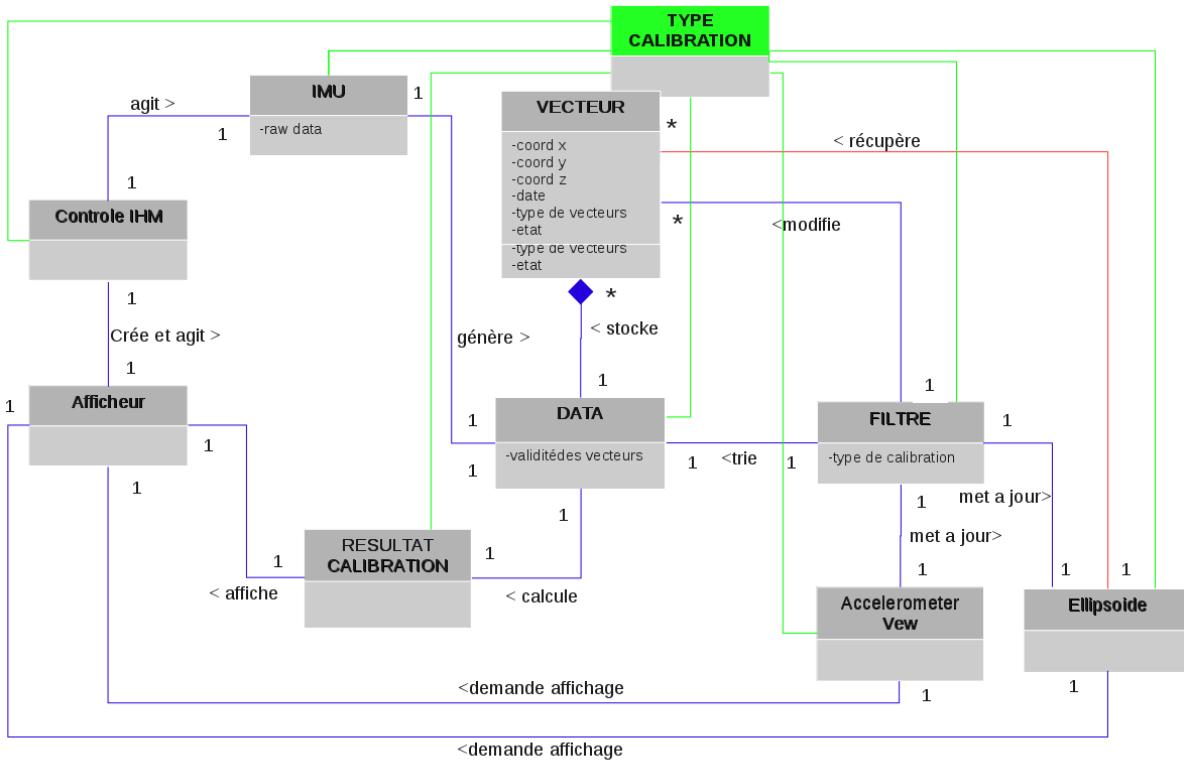


Figure 24 : Diagramme de classe choix de chaîne d'affichage

## V. Annexes

### 1. Use Cases

#### a. Use case magnétométrique

Use case	Calibration des accéléromètres
Description	La Calibration de l'accéléromètre est effectuée en suivant les étapes dictée par le logiciel de calibration.
Acteurs	L'utilisateur, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Des accéléromètres compatibles. Un ordi pour le logiciel. Un plan de travail stable pour effectuer les mesures.
Etapes	<ol style="list-style-type: none"><li>1. L'utilisateur démarre le logiciel</li><li>2. Le logiciel vérifie la présence du drone</li><li>3. L'utilisateur sélectionne l'option « calibration de l'accéléromètre » dans la fenêtre du logiciel</li><li>4. Le système affiche une vue et un indicateur de progression de la mesure en cours ainsi que de la calibration totale</li><li>5. L'utilisateur déclenche la saisie des mesures en choisissant l'option dans la fenêtre du programme</li><li>6. Le système enregistre les mesures accélérométriques</li><li>7. Le système affiche les consignes de calibration à l'utilisateur</li><li>8. L'utilisateur manipule le drone en accord avec les consignes</li><li>9. Le système filtre et affiche les mesures accélérométriques</li><li>10. L'utilisateur arrête les mesures en choisissant l'option dans la fenêtre du programme</li><li>11. Le système arrête la prise de mesures accélérométriques.</li><li>12. Génération des résultats de la calibration accélérométrique</li><li>13. Fermeture du logiciel ou démarrage d'une nouvelle calibration.</li></ol>

b. Use case accéléromètres

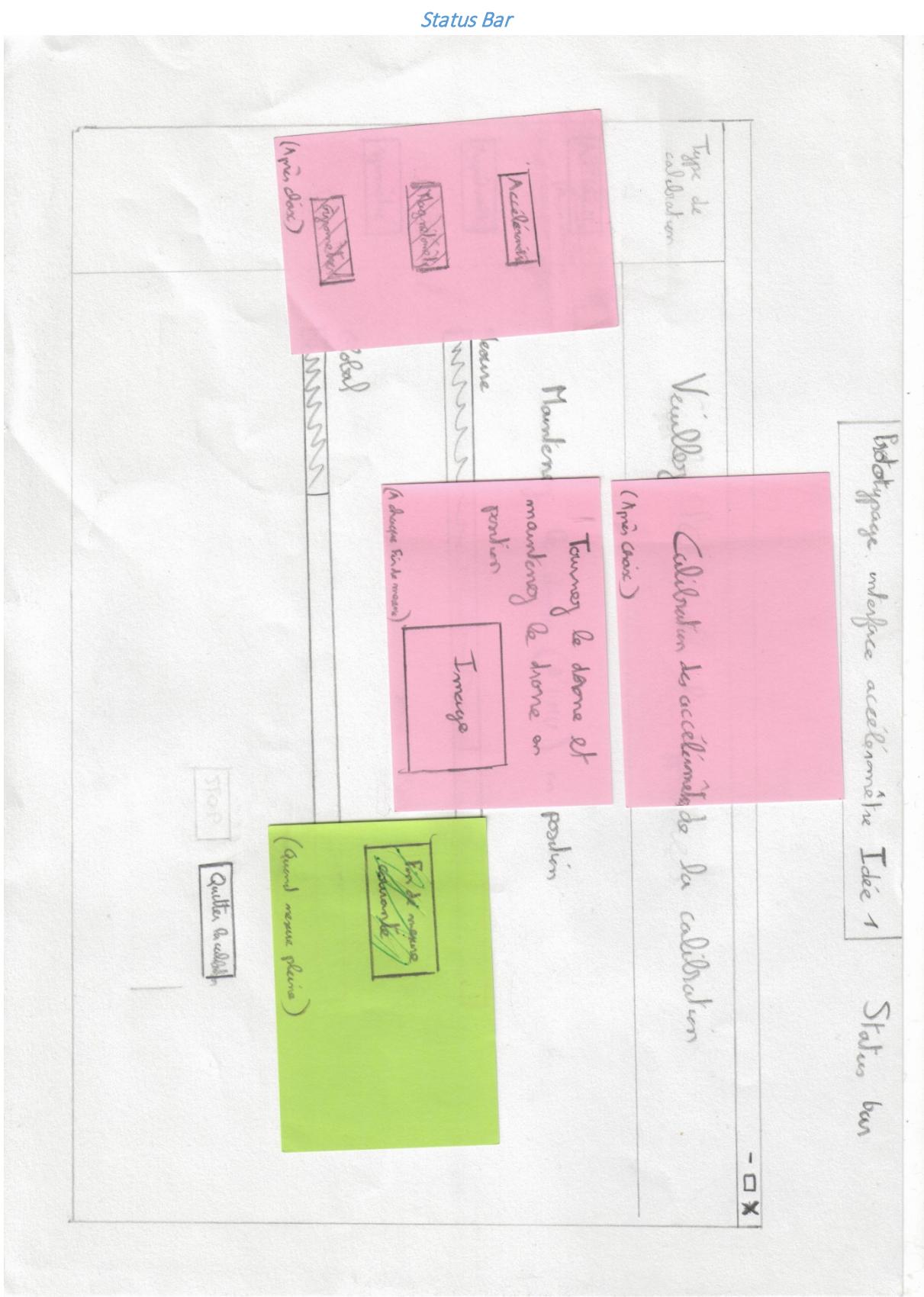
<b>Use Case</b>	<b>Calibration du magnétomètre</b>
<b>Description</b>	La Calibration du magnétomètre est effectuée en suivant les étapes dictée par le logiciel de calibration.
<b>Acteurs</b>	Le client, le logiciel, le drone/IMU
<b>Assomptions</b>	Une connexion du drone avec le bus Ivy est nécessaire. Un magnétomètre compatible. Un ordi pour le logiciel.
<b>Etapes</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur démarre le logiciel</li> <li>2. Le logiciel vérifie la présence du drone</li> <li>3. L'utilisateur sélectionne l'option « calibration du magnétomètre » dans la fenêtre du logiciel</li> <li>4. Le système affiche une vue et un indicateur de progression de la mesure en cours ainsi que de la calibration totale</li> <li>5. L'utilisateur déclenche la saisie des mesures en choisissant l'option dans la fenêtre du programme</li> <li>6. Le système enregistre les mesures magnétométriques</li> <li>7. Le système affiche les consignes de calibration à l'utilisateur</li> <li>8. L'utilisateur manipule le drone en accord avec les consignes</li> <li>9. Le système filtre et affiche les mesures magnétométriques</li> <li>10. L'utilisateur arrête les mesures en choisissant l'option dans la fenêtre du programme</li> <li>11. Le système arrête la prise de mesures magnétométriques.</li> <li>12. Génération des résultats de la calibration magnétométriques</li> <li>13. Fermeture du logiciel ou démarrage d'une nouvelle calibration</li> </ol>

2. Prototypage papier d'interface

a. Prototypage accueil

Prototypage accueil	
Type de la calibration	Veuillez choisir le type de la calibration
<input type="checkbox"/> Acc	
<input type="checkbox"/> Mag	
<input type="checkbox"/> Gyro	
Présentation	
<input type="checkbox"/> Oui	
<input type="checkbox"/> Non	

b. Prototypage interface accéléromètre



Two White Square

Prototypage interface magnétométrique Tâche 8

Planophore

Type de

Mag

Axe

Tournez le boîtier lentement dans les deux sens pour remplir les différentes zones

Venuleux

(Axe)

Calibration des magnétôdes

de la calibration



Méthode de l'asservissement par zone avec la zone en cercles de remplacement en substitution, la moitié au dessus. La zone de préférence planifie la forme ronde en fait des points égales.

STOP

Quel

## Poste d'usage interface magnétronique idée 3

Planquère V2

Type de

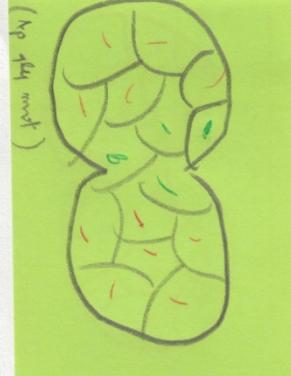
Veuillez

Calibration des magnétos

Da calibration

- □ X

Tournez le bouton lentement dans tous les sens pour remplir les différentes zones



Item 2 mais la planquère  
ne fait pas  
transformation de  
l'espace et fusion  
en respectant les  
bordures

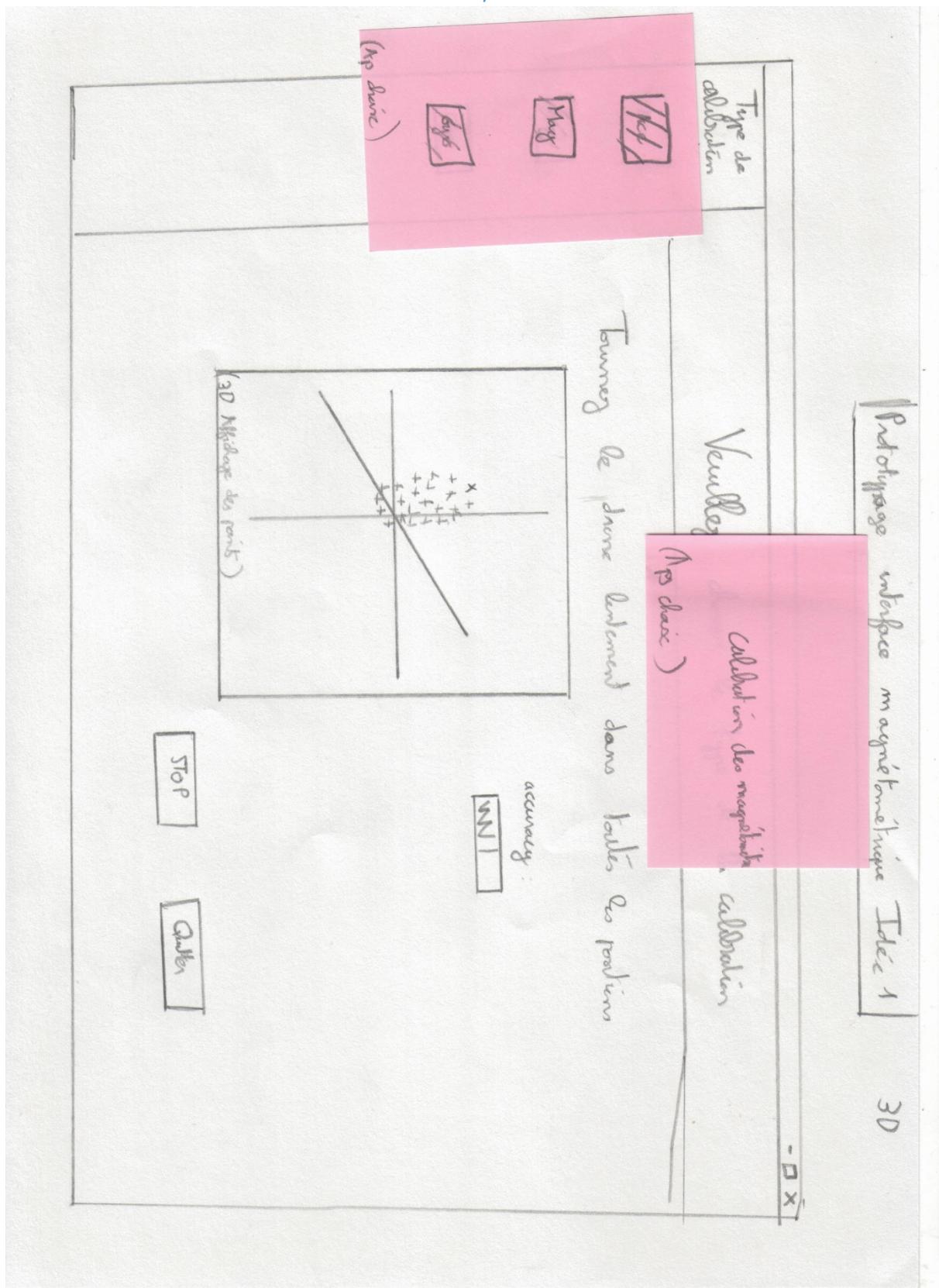
(Explication)

[STOP]

[Quel]

c. Prototypage interface magnétomètre

3D Graphics



Prototypage interface accéléromètre Idée 2

Trois White Spots

Vous

Calibration des accéléromètres

Da calibration

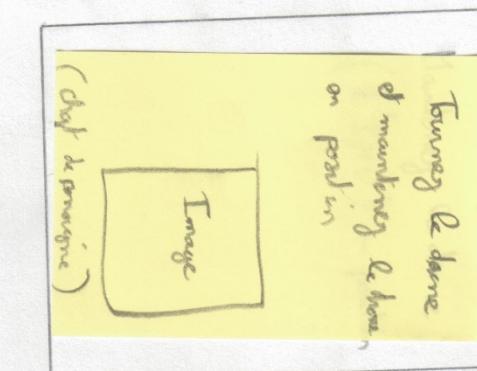
- □ X

Type de  
calibration

[Accel]

[Mag]

(Après choix)



[STOP] [Quitter]

Planisphere

Prototypage interface accélérmètre Idée 3

3D Two White Square

Type de collecteur	Vauflay
Collecteur des accél.	Albatrem
(Avec Drac)	
	X -
	- X
Markinez la position de l'axe de rotation du membre le donc un point	
(Translation)	
Image	
STOP	
Quitter	