PROJET CHRONO-CROSS 2019

Dossier Technique v1.0

ANDRÉO Michaël et TURLIN Suzie

(Étudiants BTS SN-IR)

I - Partie générale

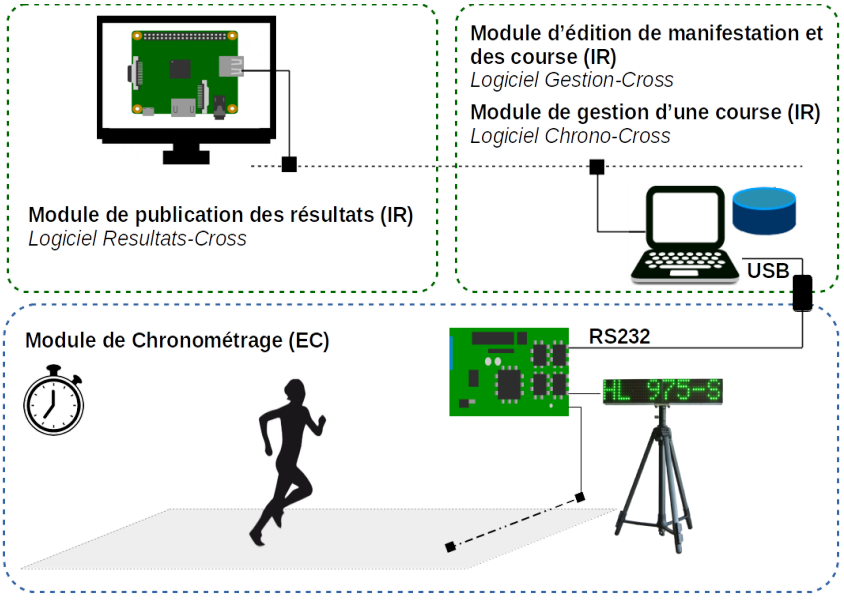
Présentation du projet

Il s'agit de développer pour l'établissement La Salle un système informatisé de gestion de bout en bout de courses à pied. Ce système a été nommé Chrono-Cross. L'établissement La Salle organise chaque année un cross pour les élèves du collège et il recherche une solution informatique qui devra :

- gérer plusieurs courses pour une même manifestation
    - chronométrer des temps avec une précision d'une seconde
    - classer les coureurs à l'arrivée
    - publier les résultats pour le public et les coureurs (affichage sur grand écran)

On distinguera les modules suivants :

- module de chronométrage (avec affichage lumineux) (EC)
- module d'édition de manifestation et des courses (IR)
- module de gestion de la course (IR)
- module de publication des résultats (IR)



Analyse de l’existant

Le système informatique réalisé ici est proposé par de nombreuses sociétés spécialisées dans le secteur des sports : entraînement et préparation, organisation des compétitions et ... chronométrage.

Dans le domaine sportif, l'intérêt et les atouts de ce type de système ne sont plus réservés aux grandes structures et manifestations du fait de la baisse des coûts des matériels. Le système proposé met l'informatique au service des organisateurs de courses à pied d'ampleur moyenne (quelques centaines de concurrents et des distances comptées en kilomètres).

Cette informatique répond aux attentes grandissantes des utilisateurs :

- pour les organisateurs : économie de temps, fiabilité, information en temps réel
- pour les coureurs : mise à disposition rapide et fiable des classements
- pour le public : disponibilité des résultats sans délai

Expression du besoin

L'équipe des enseignants d'EPS, organisateur du cross du collège, est intéressée par un système de chronométrage et classement des concurrents de courses à pied.

Le développement de l'application doit répondre aux exigences des utilisateurs :

- simplicité d'utilisation,
- correspondre aux contraintes définies,
- réalisable dans un délai de 200 heures (IR) et 170 heures (EC).

Le système Chrono-Cross devra remplir les missions suivantes :

- éditer une manifestation (créer, modifier et supprimer)
- éditer des courses (créer, modifier et supprimer)
- inscrire des coureurs à des courses
- chronométrer une course
- afficher les résultats
- imprimer les résultats

Tout ceci peut donc être résumé en une phrase claire et synthétique :

Gérer des courses de cross chronométrées pour un établissement scolaire.

II - Identification du travail à réaliser

Étudiants en charge du projet

OPTION EC :

- CHAÏB Salim Étudiant 1

OPTION IR :

- TURLIN Suzie Étudiant 3
- ANDRÉO Michaël Étudiant 4

Répartition des tâches entre étudiants

Étudiant EC : CHAÏB Salim

Installation :	Le système embarqué, la matrice à leds.
Mise en oeuvre :	L'environnement de développement.
Configuration :	La liaison série RS232.
Réalisation :	Les diagrammes SysML, Le code source et les schémas du module.
Documentation :	Le dossier technique et les documents relatifs au module, Un guide de mise en route et d'utilisation du module.
Cas d'utilisation :	Gérer le chronométrage d'une course Démarrer une course Chronométrer une course Afficher le temps sur une matrice à leds Détecter les arrivées Transmettre les temps d'arrivée Transmettre les classement d'arrivée

Étudiant IR 3 : TURLIN Suzie

Installation :	La Raspberry Pi, l'écran.
Mise en oeuvre :	L'environnement de développement.
Configuration :	La communication réseau.
Réalisation	Les diagrammes UML, L'IHM du module, Le code source de l'application.
Cas d'utilisation :	Éditer une manifestation Publier les résultats Visualiser le classement et les temps Imprimer les résultats

Étudiant IR 4 : ANDRÉO Michaël

Installation:	La base de données
Mise en oeuvre :	L'environnement de développement
Configuration :	Les communications (réseau et série), la base de données
Réalisation :	Les diagrammes UML, l'IHM du module, le code source de l'application
Documentation :	Le dossier technique et les documents relatifs au module. Un guide de mise en route et d'utilisation du module.
Cas d'utilisation :	Éditer les coureurs
	Inscrire les coureurs
	Démarrer une course
	Chronométrer et classer les arrivées

Cas d'utilisation

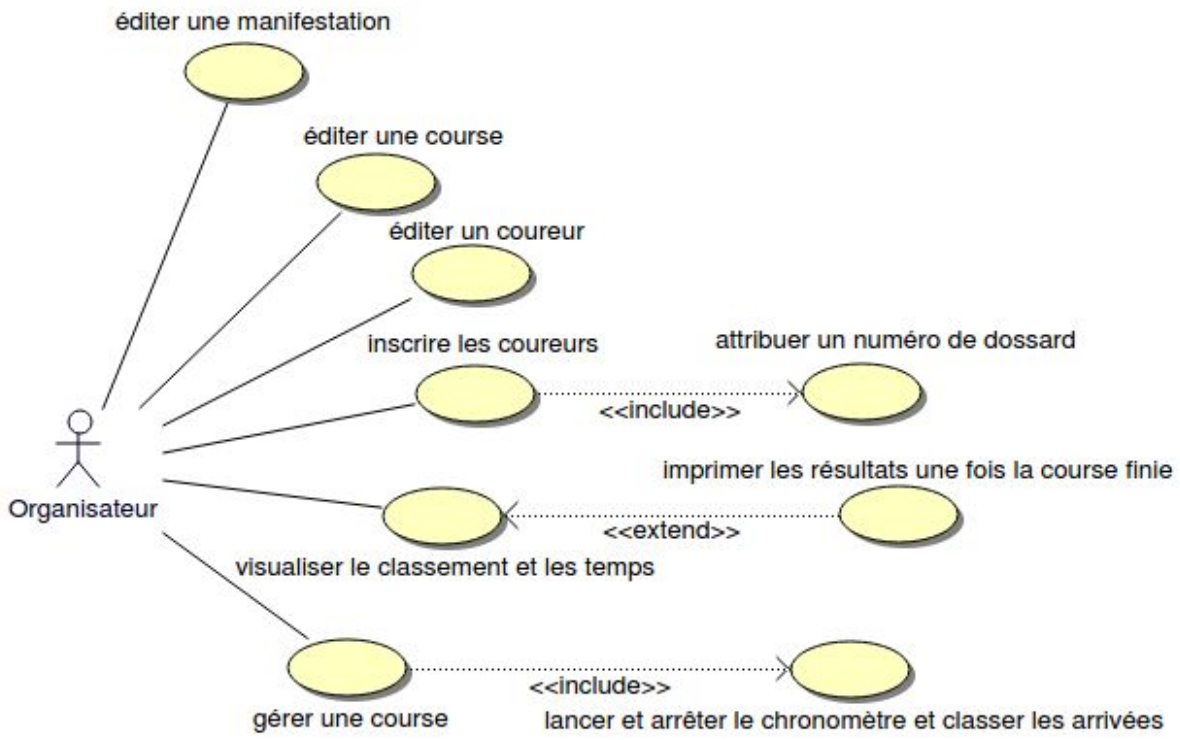


Diagramme des cas d'utilisation

Description des cas d'utilisation

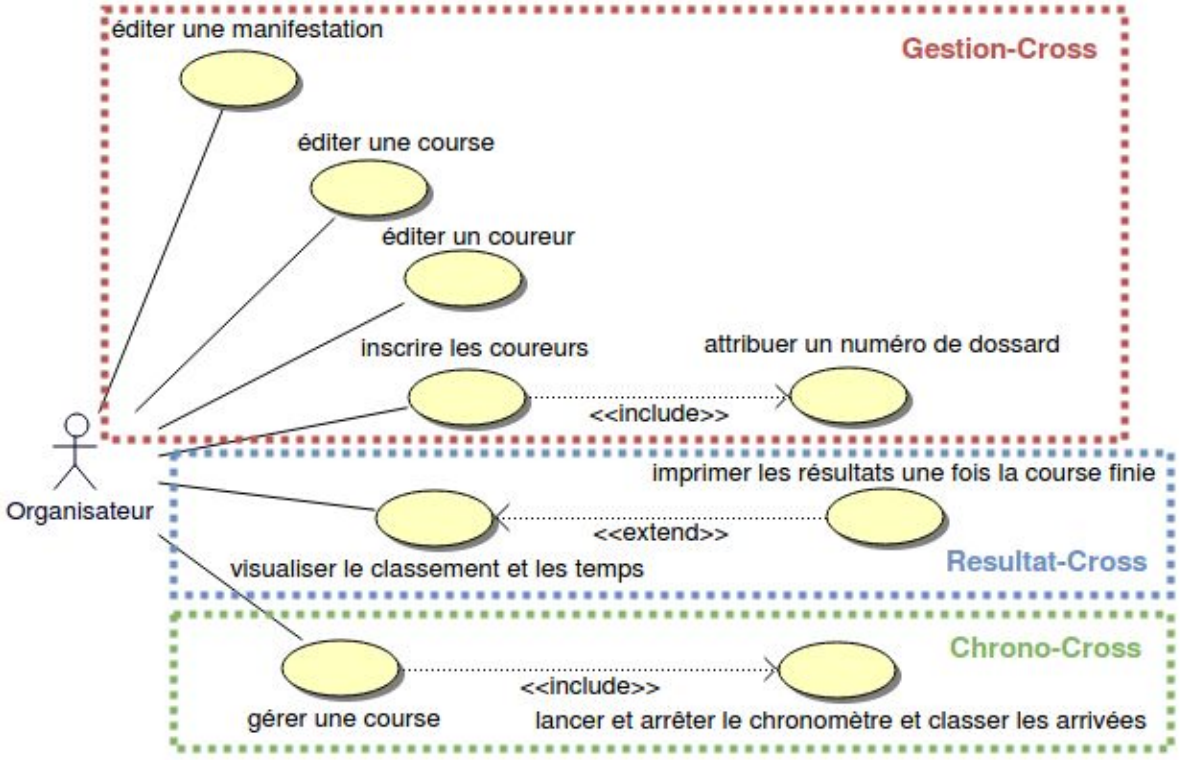
Ce système peut être utilisé par un professeur. Il est composé de 3 logiciels :

- Le logiciel **Chrono-Cross** qui permet de :
  - *Afficher les courses disponibles pour une manifestation*
  - *Lancer et arrêter le chronomètre*
  - *Classer les arrivées et affecter un dossard à un temps pour effectuer le classement*
- Le logiciel **Gestion-Cross** qui permet de :
  - *Éditer une manifestation (pouvoir créer, modifier ou supprimer une manifestation),*
  - *Éditer une course (pouvoir créer, modifier ou supprimer une course),*

- *Éditer des coureurs (pouvoir créer, modifier ou supprimer un coureur),*
- *Inscrire les coureurs à une course et leur attribuer des dossards*
- *Visualiser le classement et les temps une fois une course fini et pouvoir imprimer les résultats*

- Le logiciel **Resultat-Cross** qui permet de :
  - *Visualiser le classement et les temps pour une course*

On peut donc répartir les différents logiciels selon les **cas d'utilisation**.



Description des ressources matérielles à mettre en oeuvre

Seront livré à l'établissement avec les logiciels un certain nombre de ressources matérielles :

- Un **PC "course"** sur lequel seront installés les logiciels Gestion-Cross et Chrono-Cross,
- Un **Écran avec une Raspberry Pi** qui affichera les résultats en temps réel,
- Un **afficheur matriciel à LEDS** qui affichera le chronomètre de la course en cours,
- Ainsi qu'une **cellule infrarouge** d'arrivée qui permettra de détecter une arrivée.

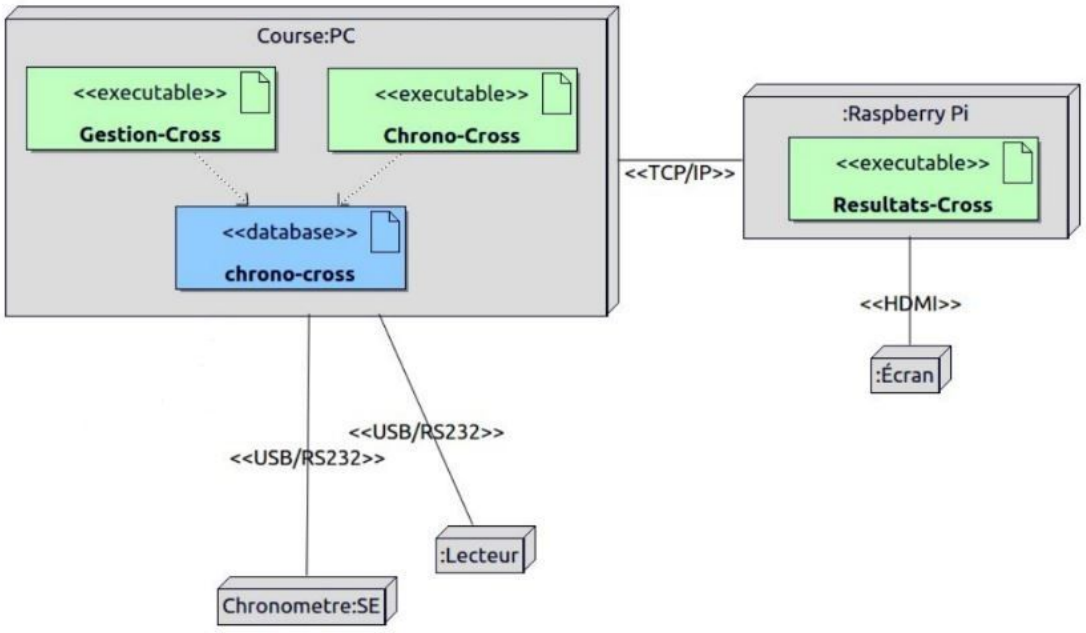


Diagramme de déploiement

III - Plan de tests de validation

Désignation	Démarche à suivre	Résultat attendu	Oui / Non	Remarque
Connecter le chronomètre et l'interface homme-machine	- Lancer le logiciel "Chrono-Cross" - Sélectionner une manifestation dans la liste puis une course	La led chrono passe de rouge à orange.		
Synchroniser le chronomètre, l'interface homme-machine et la course	- Lorsque le chronomètre est connecté à l'interface, cliquer sur le bouton "Démarrer"	La led course passe de rouge à orange. La course passe à l'état “Prete”		
Lancer une course	- Lorsque le PC et le chronomètre sont synchronisés - Cliquer sur le bouton “Lancer”	La led course passe au vert. La led chrono passe au vert. Le chronomètre de l'IHM se lance. Le chronomètre TAGHEUER se lance. La course passe à l'état “EnCours”		
Réception d'un nouveau temps	- Un coureur passe devant le capteur infrarouge	Le chronomètre TAGHEUER émet une trame “TN”. La liste des temps non classés affiche un nouveau temps en “HH:MM:SS”.		
Associer un numéro de dossard à un temps non classé	- Lorsqu'un nouveau temps s'affiche, entrer un numéro de dossard valide	Le temps non classés est transféré au classement avec le numéro de dossard et les informations du coureurs.		
Différencier l'affichage pour les trois premiers	- Assigner trois numéros de dossard à des temps non classés	Les temps s'affiche dans le tableau classement, le premier est couleur or, le deuxième couleur argent et le troisième couleur bronze.		

Entrer un numéro de dossard invalide	- Entrer un numéro de dossard invalide à un temps non classé	Un message rouge d'erreur apparaît		
Supprimer le premier temps non-classé	- Entrer le numéro de dossard "0000"	Un page s'affiche demandant la confirmation pour la suppression Si l'on confirme le temps est retiré de la liste		
Arrêter une course	- Après que la course a été lancée cliquer sur le bouton "Arreter"	Le chronomètre de l'ihm s'arrête Le chronomètre TAGHEUER s'arrête La course passe à l'état "Arretee"		
Terminer une course	- Après que la course a été arrêtée cliquer sur le bouton "Terminer"	La course passe à l'état "Terminee"		
Créer un coureur	- Lancer le logiciel "Gestion-Cross" - Cliquer sur le bouton Coureur - Entrer les informations puis cliquer sur créer	Un coureur est ajouté à la liste et à la base de donnée		
Modifier un coureur	- Dans le logiciel "Gestion-Cross" cliquer sur le bouton coureur - Sélectionner un coureur puis modifier les valeurs - Cliquer sur modifier	Les informations du coureur ont été modifiés dans la liste et dans la table coureur de la base de données		
Supprimer un coureur	- Dans le logiciel "Gestion-Cross" cliquer sur le bouton coureur - Sélectionner un coureur que vous souhaitez supprimer - Cliquer sur le bouton "Supprimer" et confirmer	Le coureur a été supprimé de la liste des coureurs et dans la table coureur de la base de données		
Entrer des informations erronées	- Dans le logiciel "Gestion-Cross" cliquer sur le bouton coureur - Cliquer sur modifier puis entrez des informations erronées	Un message d'erreur apparaît et la partie fausse devient rouge Les informations ne sont pas ajoutées à la base de		

		données		
Inscrire un coureur à une course	- Dans le logiciel “Gestion-Cross” cliquer sur le bouton coureur - Sélectionner le coureur que vous souhaitez inscrire puis sélectionner la course, entrer un numéro de dossard valide puis cliquez sur le bouton “Inscrire” et enfin confirmé	Le coureur est ajouté à la tabe Inscrit de la base de données avec son idCoureur et son numéro de dossard		
Afficher tous les coureurs	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur	Le tableau de coureurs se remplit de tous les coureurs enregistré dans la base de données		
Afficher les manifestations disponibles pour un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur	La liste des manifestation disponible se met à jour et montre les manifestations disponible pour ce coureur		
Afficher les courses disponibles pour un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur - Sélectionner une manifestation disponible	La liste des courses disponible se met à jour et montre les courses disponible pour ce coureur		

IV - Ressources logicielles utilisées

Qt Creator : Environnement de développement intégré

Modules spécifiques **Qt** :

➔ **QtSerialport**

➢ Permet la gestion d'un port série RS-232

➢ Disponible depuis Qt 5.1.0

➔ **QtSql**

➢ Permet la prise en charge de bases de données relationnelles (SQL : Structured Query Language)

MySQL : Serveur de bases de données relationnelles

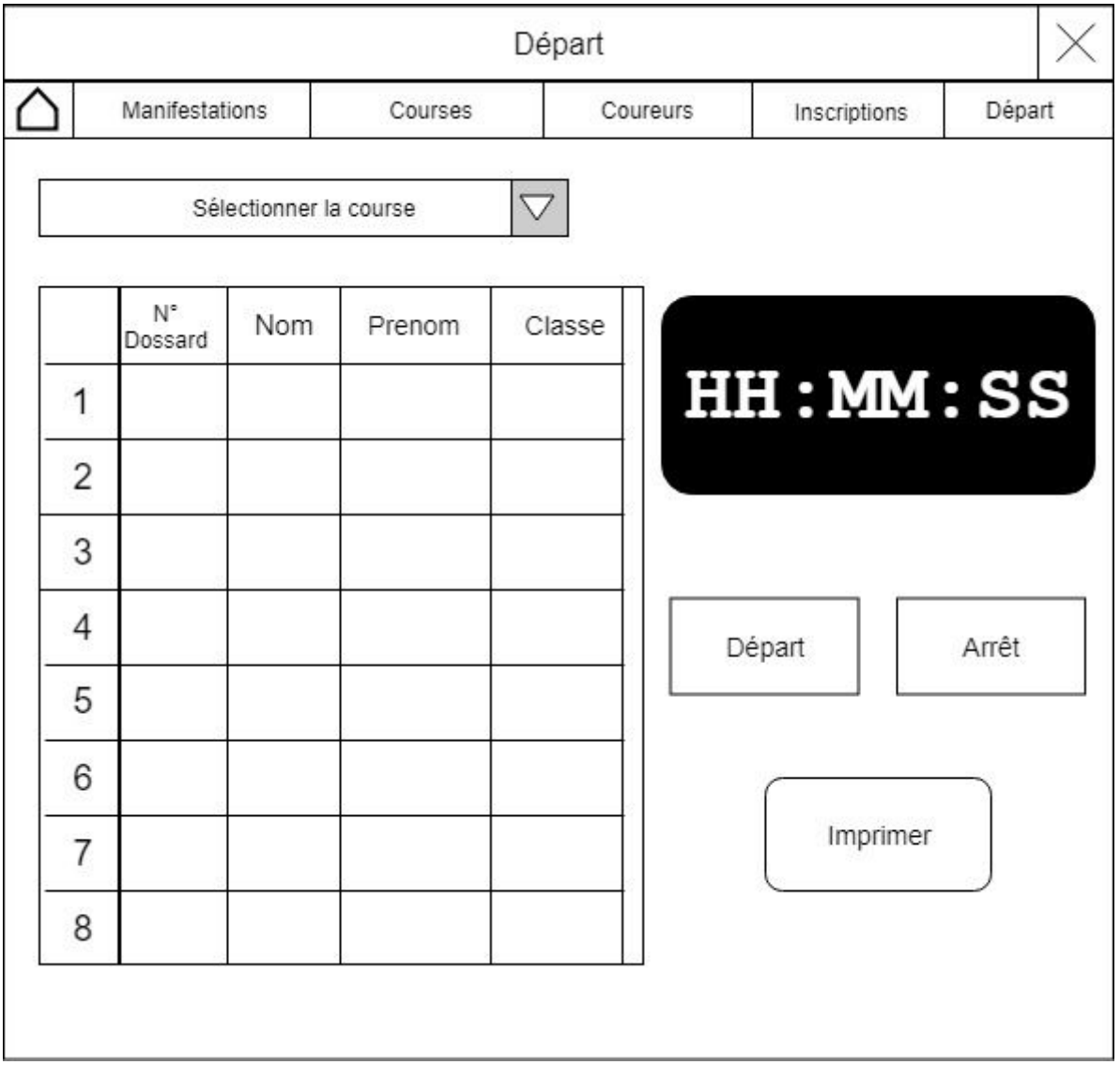
Versions utilisées :

- **Qt** 5.11.2
- **Qt Creator** 4.7.1
- **MySQL** 5.7.26

V - Prototypage et maquettes de l'IHM

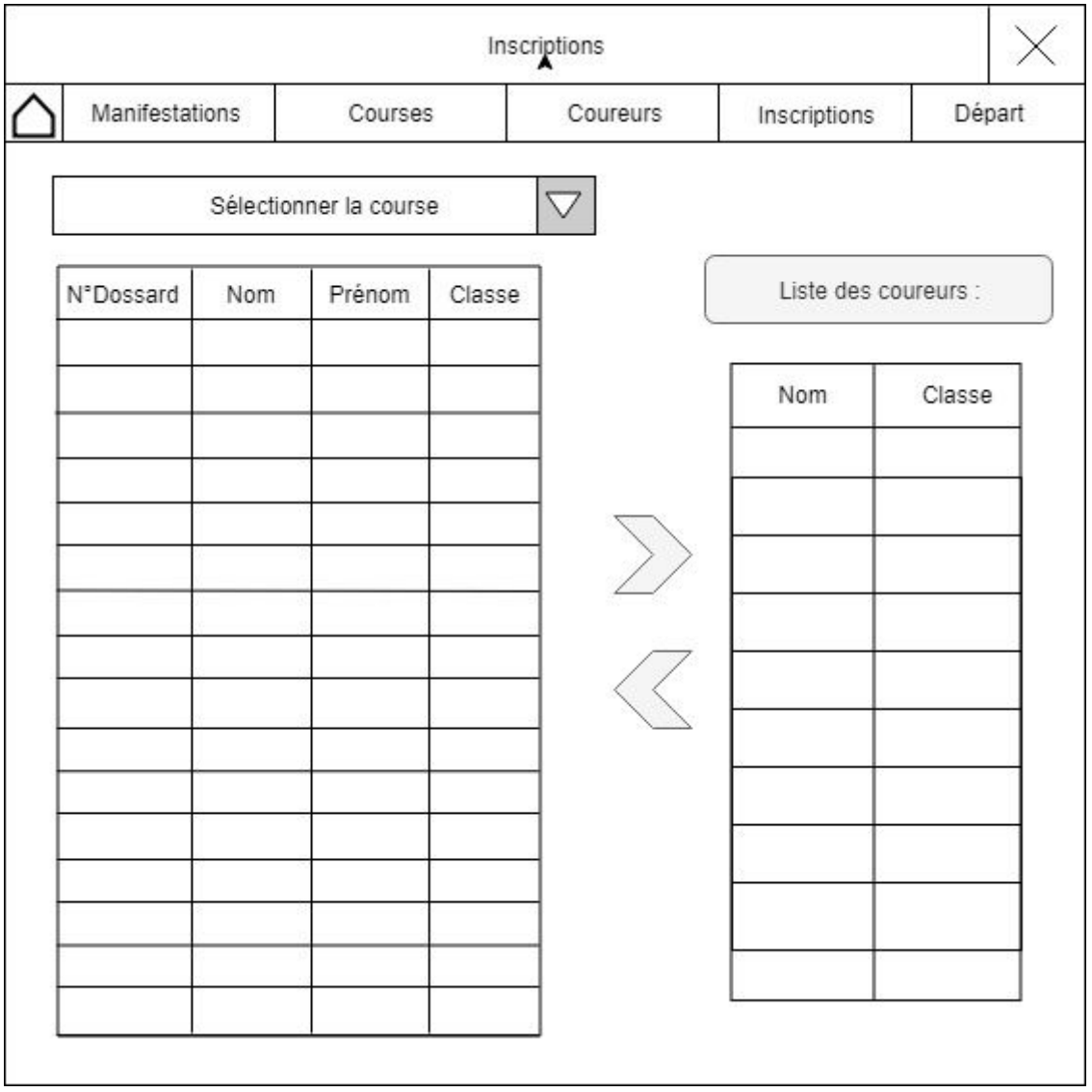
Voici les différentes maquettes de l'IHM imaginées en début de projet :

1. Cette maquette concerne la page "Départ" du logiciel **Chrono-Cross**. Depuis celle-ci, on pourra choisir une course d'une manifestation et la démarrer :



2. Cette maquette représente une partie du logiciel **Gestion-Cross**. Sur cette IHM, on doit pouvoir gérer les manifestations.

3. Cette dernière maquette représente l'autre partie du logiciel **Gestion-Cross**. On doit pouvoir ici gérer les courses et les attribuer à une manifestation.

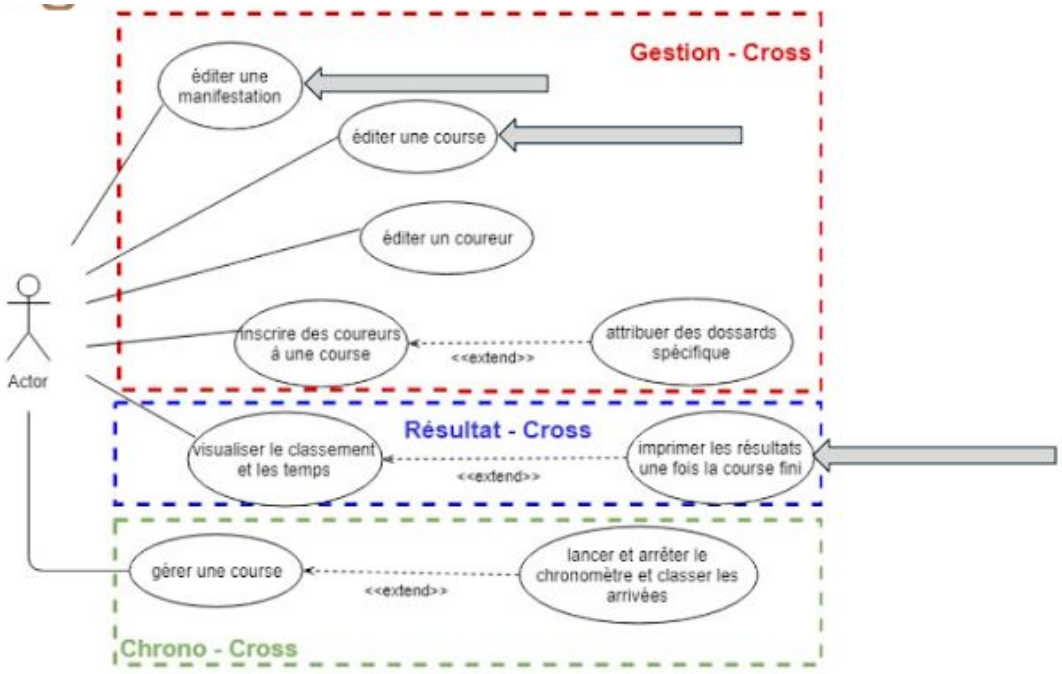


VII - Partie personnelle Turlin Suzie

Objectifs

Pour commencer, ma partie est de pouvoir gérer une manifestation ou événement pour le collège la Salle, pour cela de deux applications qui seront réalisées avec le logiciel Qt Creator. La première est Resultat-Cross, elle sera faite pour afficher en temps réel les classements des coureurs à la fin de la course. La seconde est Gestion-Cross qui servira à gérer, les manifestations (événement), et gérer des courses. Grâce à celle-ci on pourra créer des manifestation, et créer des courses. Les organisateurs auront plus de facilité à organiser leur événement grâce à ces applications. On aura aussi une option pour pouvoir imprimer les résultats d'une course.

Diagramme de cas d'utilisation



Sur le diagramme de cas d'utilisation, je dois m'occuper de la partie Resultat-Cross, et sur le Gestion-Cross je m'occupe en coopération de la partie éditer une manifestation et éditer une course. Grâce à ce diagramme nous pouvons donner une vision globale du comportement fonctionnel de notre système.

Notre objectif est donc de réaliser un logiciel, facile et simple d'utilisation, pour les organisateurs du cross du collège.

Présentation

Pour cela, nous avons pris connaissance du cahier des charges, et nous avons fait un brainstorming pour répondre le mieux à nos besoins. Ce qui nous à mener à comprendre le besoin du client et à comprendre exactement ce que je devais réaliser.

J’ai réalisé un croquis d’une IHM, en effet cela nous a permis de visualiser au mieux, ce que pourrait donner le rendu et nous permettre de nous rapprocher au plus près de ce que le client souhaite.

Actuellement j’ai réalisé le début de l’interface homme machine, nous avons un tableau avec 4 lignes (nom, prénom, classe, ine). dans ce tableau sera répertorié le classement des coureurs que Michael Andréo enverra dans la base de données et que je récupère par la suite, grâce au numéro de dossard qui sont attribués avant le début de la course. Mais pour cela il faudra sélectionner la course correspondante pour avoir le bon classement.

Maquette IHM Manifestations

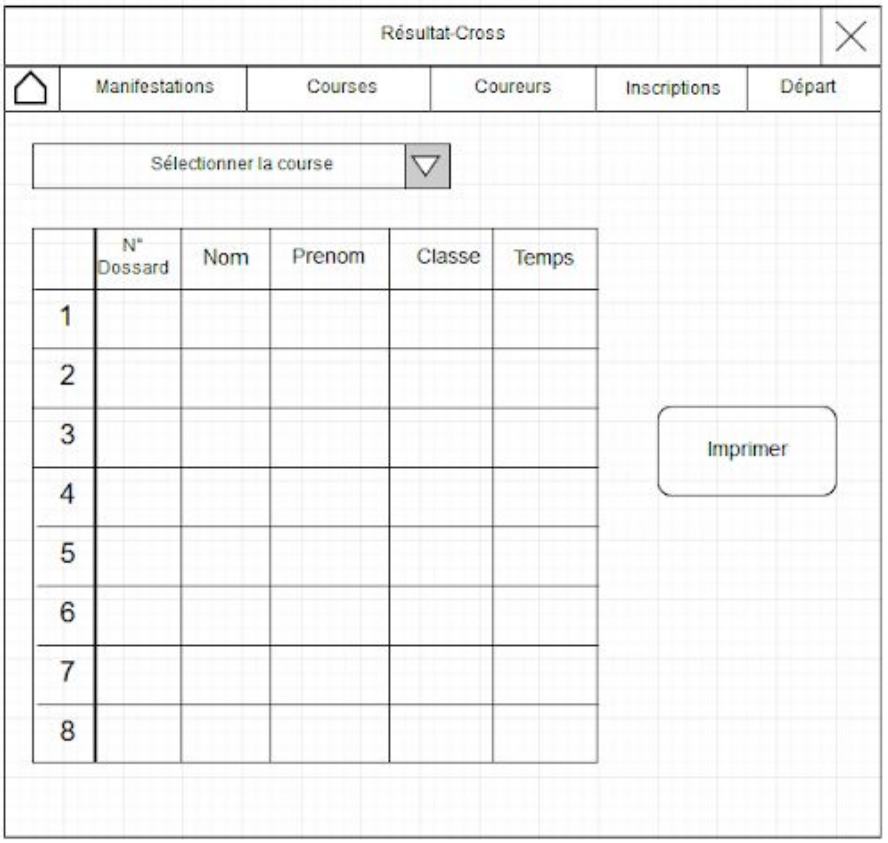
L'application sera disposée de la manière suivante :

Cette interface homme machine dispose de 3 boutons :

- créer : ce bouton sert à créer une nouvelle manifestation, on pourra alors ajouter un nom est une date.
- modifier : ce bouton sert à modifier une manifestation
- supprimer: ce bouton supprime la manifestation sélectionner

Maquette IHM Résultats

Le classement sur l'écran sera représenté comme ci-dessous :



Nous pouvons voir que sur cette application nous avons accès aux différentes courses via un menu déroulant. Une fois la bonne course sélectionnée, nous pourrons voir en direct le classement des coureurs. Les coureurs ayant passé la ligne d'arrivé seront caractérisés par leur numéro de dossard, nom, prénom, classe et leur temps d'arriver à la fin de la course.

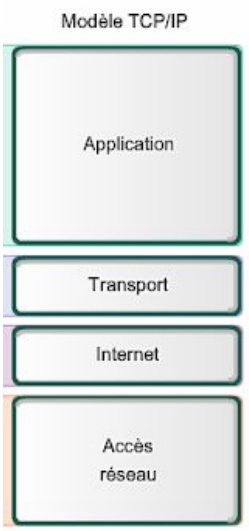
Quand la course est terminée, nous pourrons imprimer les résultat. Ces résultats seront enregistrés en format texte et imprimable facilement.

La Raspberry Pi

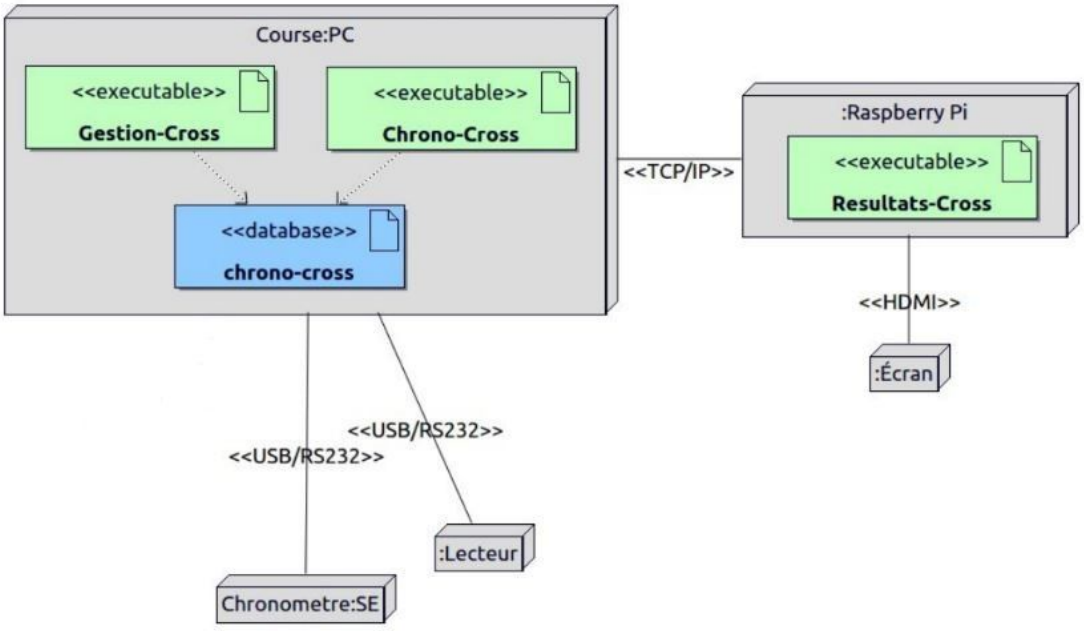
La Raspberry Pi est un nano-ordinateur monocarte à processeur ARM.

Ici, elle servira à afficher, le classement sur un écran en temps réel pour que le public puisse suivre la course.

Elle sera donc connectée à mon application via une liaison TCP/IP (TCP, Transmission Control Protocol et IP, Internet Protocol). La liaison TCP/IP est un ensemble de protocoles utilisés pour le transfert de données sur internet. Il est composé de 4 couches :

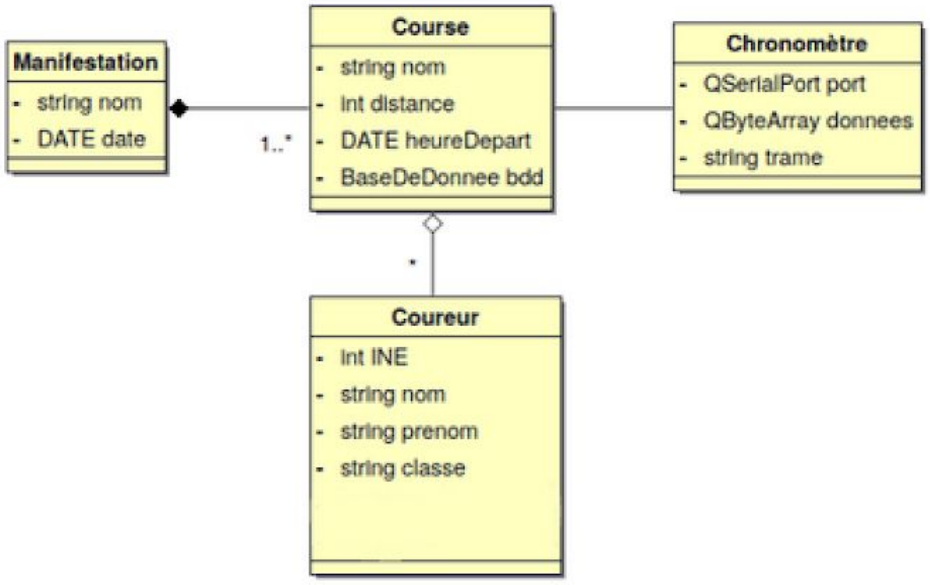


On pourra donc accéder à distance à la raspberry pi pour lui transmettre les informations souhaitées.



Avec les applications qui seront sur un même ordinateur Course:PC comme on peut le voir ci dessus, on se connectera via le SHELL (que Secure SHell (SSH) est à la fois une programme informatique et un protocole de communication) . C'est à dire que grâce à cette commande : $ssh -X pi@192.168.52.252, on pourra afficher les applications du pc sur l'écran de la raspberry pi directement.

Voici le diagramme de classes :



Donc sur ce diagramme je m'occupe de la classe Manifestation et de la Classe Coureur.

On peut voir qu'une manifestation est caractérisé par un nom et une date, elle peut contenir de 1 à plusieurs courses.

Les coureurs sont caractérisés par un INE, nom, prénom, classe.

Dans une course il peut il y avoir un ou plusieurs coureurs.

VIII - Partie personnelle Andréo Michaël

Objectifs

L'objectif numéro 1 est de réaliser le module de gestion d'une course (le logiciel **Chrono-Cross**). L'utilisateur doit pouvoir sélectionner une course d'une manifestation (récupérées depuis la base de données) et lancer le chronomètre. Par la suite, il recevra les temps d'arrivée des coureurs et il associera manuellement leur numéro de dossard à leur temps. La communication avec le chronomètre se fera via liaison RS-232.

L'objectif numéro 2 est de cogérer la base de données à travers le module d'édition de manifestations et des courses depuis le logiciel **Gestion-Cross**. Cette partie se fait en collaboration avec l'autre étudiant IR, TURLIN Suzie.

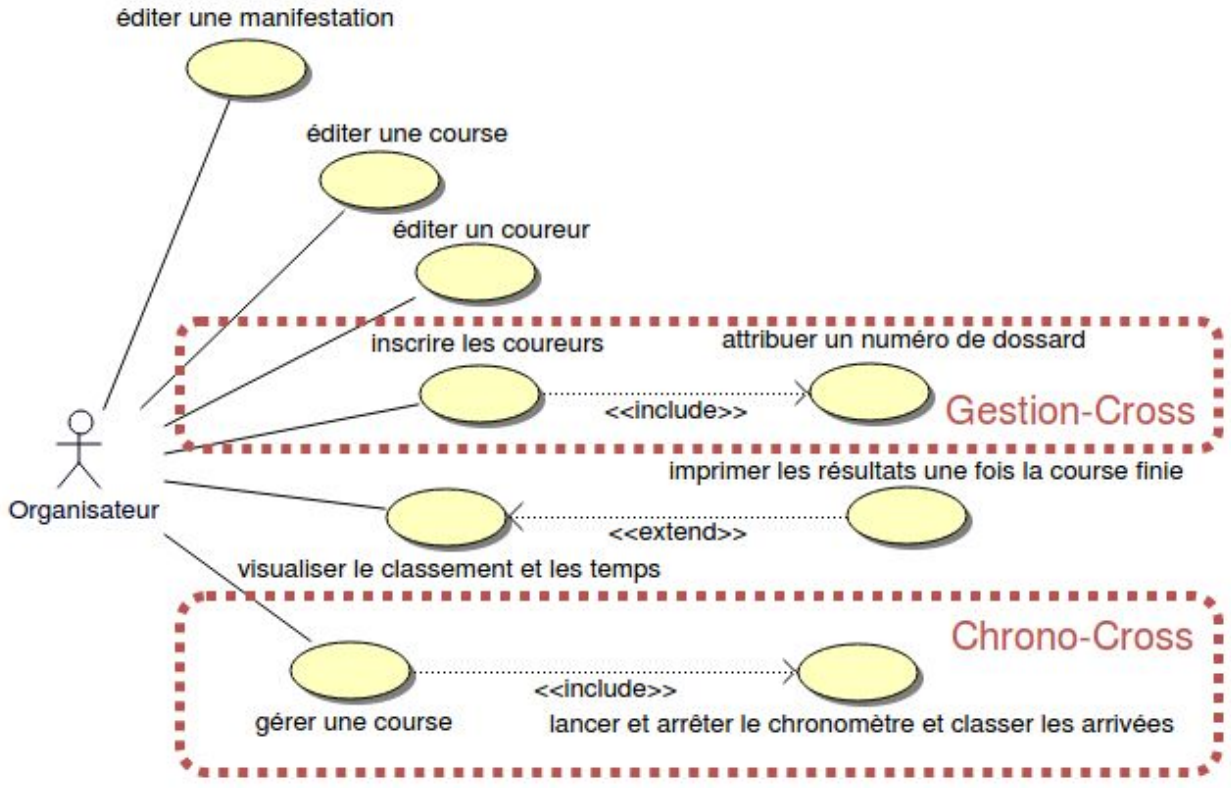
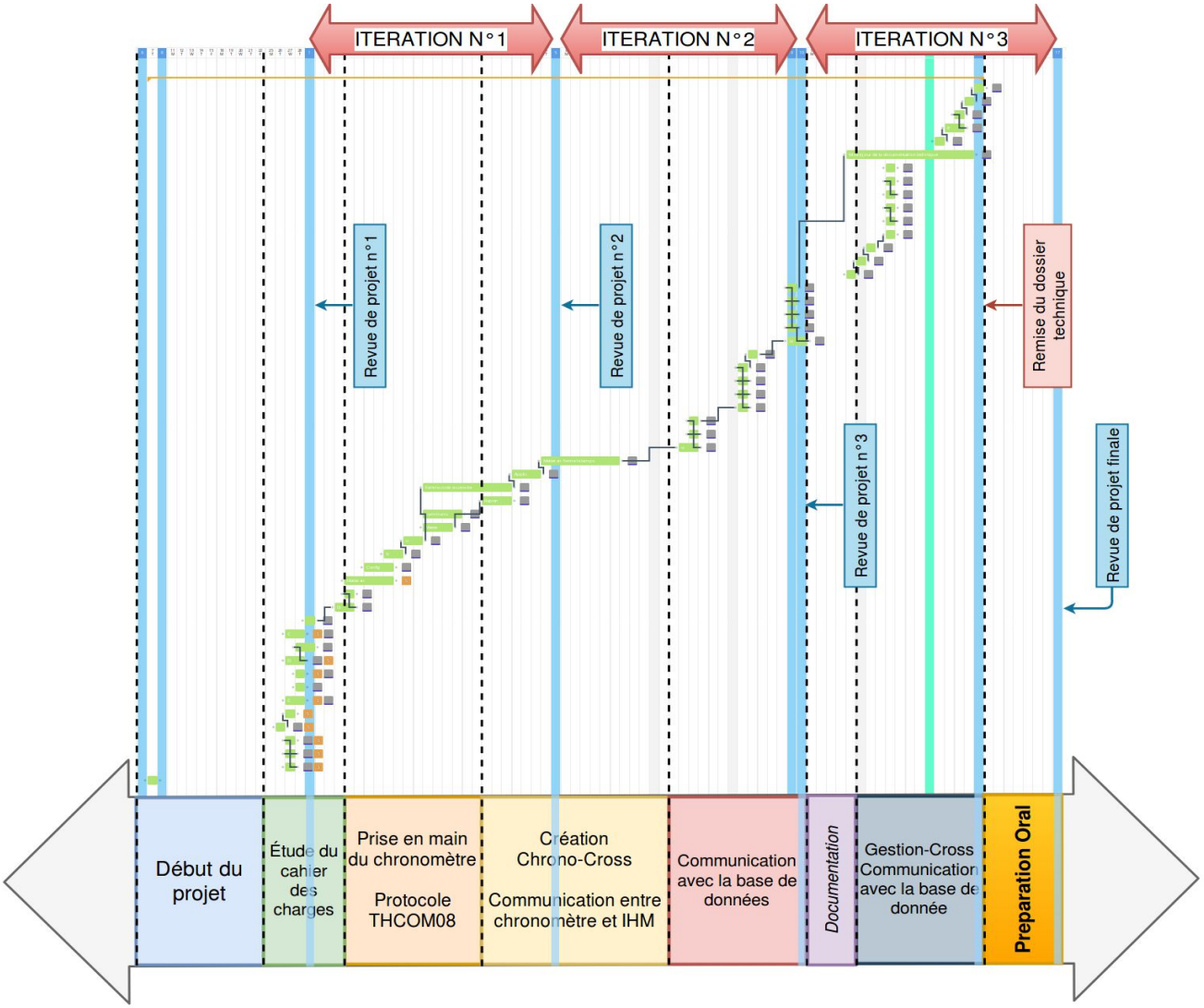


Diagramme de cas d'utilisation (les cas encadrés seront à ma charge)

Planification des tâches

Le projet a débuté le 5 février, nous avions 220 heures pour le réaliser. Nous avons découpé le développement en 3 itérations entre chaque revue de projet :

- La **première itération** concerne le développement du logiciel Chrono-Cross et de la communication entre l'IHM et le chronomètre
- La **deuxième itération** représente la communication entre l'IHM et la base de données
- La **troisième itération** concerne le développement du logiciel Gestion-Cross et de la gestion de la table Coureur de la base de données.



Logiciel Chrono-Cross

Présentation du chronomètre

Pour pouvoir débuter le projet, nous utiliserons un chronomètre de la marque TAG HEUER (modèle HL 975) ce qui nous permettra de mettre en place la communication entre l'IHM et le futur chronomètre.

L'utilisation de ce chronomètre passera par le protocole **THCOM08** fourni par le constructeur.

Modèle TAG HEUER HL 975 utilisé en développement

Communication avec le chronomètre

Nous utilisons une liaison point à point en RS232 (norme V24) bidirectionnelle asynchrone avec un connecteur DB-9 (son appellation réelle est DE-9). Nous utilisons aussi un convertisseur RS232 USB pour le branchement avec le PC.

Le brochage est (DB9 femelle) :

- **broche n°2 :** Tx - Conducteur d'émission de données
- **broche n°3 :** Rx - Conducteur de réception de données.
- **broche n°5 :** GND - Conducteur pour la masse.

Nous n'utilisons pas de contrôle de flux. La vitesse de transmission est 9600 bits par seconde.

Le protocole de transmission du HL975 est le suivant :

- 1 bit de START
- 8 bits de données
- Pas de parité
- 1 bit de STOP

Connecteurs DB-9

Niveau logique	Polarité	Intervalle	Typique
1	Basse	- 3 V - 15 V	- 12 V
0	Haute	3 V 15 V	12 V

Communication RS232

Il était imposé dans le cahier des charges d'utiliser un protocoles en RS232 mais on est en droit de se demander si un autre protocole de communication aurait pu être préférable.

- Tout d'abord le RS232 à une distance limitée
- Son débit est moins important
- Communication point à point

	RS232 V24	**RS422 V11**	**RS485 V11**
Type d'interface	Unipolaire	Différentiel	Différentiel
Distance théorique	15 m	1200 m	1200 m
Débit max	19200 Bauds	10 MBds	10 MBds
Multipoint	non	oui	oui
Nombre d'émetteurs	1	1	32
Nombre de récepteurs	1	10	32

Initialement, la norme RS232 est fonctionnelle avec un connecteur 25 broches (DB25) mais nous utiliserons un connecteur 9 broches (DB9)

N° de broche DB9	**N° de broche DB25**	**Signal**	**Description RS232**	**Description V24**
1	8	DCD	Data Carrier	DP - Détection de porteuse
2	3	RD	Received Data	RD - Réception de données
3	2	TD	Transmitted Data	TD - Transmission de données
4	20	DTR	Data Terminal Ready	ETDP - Equipement Terminal de données prêt
5	7	SG	Signal Ground	TS - Terre de signal
6	6	DSR	Data Set Ready	PDP - Poste de données prêt
7	4	RTS	Request To Send	DPE - Demande pour émettre
8	5	CTS	Clear To Send	PAE - Prêt à émettre
9	22	RI	Ring Indicator	IA - Indicateur d'appel

Protocole THCOM08

Les trames

Une trame du protocole **THCOM08** en RS232 est composé de:

- DATA + TAB (tabulation : 0x09) + checksum sur 16 bits + CR (retour chariot : 0x0D) + LF (retour à la ligne : 0x0A)
- Le caractère de début est " # "
- Pour les trames émises on commence soit par **WP** qui permet d'écrire (W) un paramètre (P), soit par **WC** qui permet d'écrire (W) une commande (C)
    - Les paramètres sont liés aux modes et aux paramètres du chronomètre (mode clock, mode StartLight ou la langue ...)
    - Les commandes sont des actions (départ manuelle, déclarer une nouvelle course, lancer une nouvelle course)

Exemple :

trame **#WP 120 3\t01AD\r\n** qui permet de passer le chronomètre en mode StartLight

#	WP	120	3	\t	01AD	\r	\n
caractère de début	écrire un paramètre	Mode de fonctionnement	StartLight	tabulation	Checksum	retour chariot	retour à la ligne

Lors de l'exécution du logiciel **Chrono-Cross** 6 différentes trames pour communiquer avec le TAGHEUER. Voici les différentes commandes utilisées :

- #WP 120 5 :
    - Permet de passer en mode clock
- #WC 002 :
    - Déclare une nouvelle course

#WP : Write Parameter
#WC : Write Command

AK : Acknolwedge of a received command (reconnaissance d'une commande reçu)

- #WC 007 02 00:00 00/00/01 :
    - Créer un nouveau chrono qui sera initialisé à 00:00,

      Le 02 symbolise un départ manuel

La date peut représenter le jour de la course (on utilise pas cette information)

- #WC 008 01 : Départ manuel lancé
- #WC 001 : Met fin à la course
- #WP 120 3 : Passe en mode StartLight et éteint artificiellement le HL 975

L'acquittement

Lorsqu'une trame est envoyée, le chronomètre TAGHEUER nous répond par une **trame d'acquittement** sous la forme suivante :

"AK_X + \t + checksum + \r + \n"

Où X représente l'état de l'acquittement :

- C pour accepté : "AK C\t00EF\r\n"
- F pour rejeté : "AK F\t00F2\r\n"
- R pour non supporté : "AK R\t00FE\r\n"

Exemple :

Par exemple on émet la trame **#WP 120 3\t01AD\r\n**

hôte ---------- #WP 120 5\t014B\r\n ----------> Machine

hôte <---------- #AK C\t00EF\r\ ---------- Machine

Le chronomètre TAGHEUER émet le temps de passage lorsqu'un coureur traverse le faisceau infra-rouge. La trame est sous cette forme: TN 1 2 5.53800 366\t05C7

TN	1	2	5.53800	365
New Time	Classement	N° de canal	Temps	jours depuis le 01.01.2000

Le classement est initialisé lorsque l'on envoi la trame NEWRUN ("#WC 002\t014C\r\n") et s'auto-incrémente à chaque nouveau temps.

Le TAGHEUER possède deux canaux, on a ajouté un simulateur de faisceau infrarouge que l'on a branché sur le canal 2. Ainsi on peut simuler une arrivée.

Le temps est sous le format "HH:MM:SS.DDDDD".

Et enfin la dernière information est le nombre de jours entre le jour de la course (déclaré avec la commande NEW SYNCRO) et le 01/01/2000. Cependant on utilise pas cette information.

Capture de la documentation du HL-975

Time (TN, T-, T*, T+, T=, TC, TI):

```
<S>Tx_NNNN_SSSS_CC_HH:MM:SS.FFFFF_DDDDD<E>

N = Candidate number (0 - 9999)
S = Sequential number (0 - 9999)
C = Channel number (1 - 99) in case of manual entry (M1 - M4)
H = Hours (0 - 23)
M = Minutes (0 - 59)
S = Seconds (0 - 59)
F = decimal part (0 - 99999)
D = Days (0 - 32767) counting from 01.01.2000
```

Prise en charge du chronomètre HL975 sous Linux

On a choisi de configurer directement le port depuis le terminal du PC. Le HL975 est connecté au PC par un convertisseur USB RS232.

Les étapes à suivre:

- Débrancher le HL975.
- On utilise tout d'abord la commande dmesg qui permet d'afficher les messages systèmes :

```
$ dmesg > dmesg-avant.txt
```

- Rebrancher le HL975.
- On utilise de nouveau la commande dmesg :

```
$ dmesg > dmesg-apres.txt
```

- On compare les deux fichiers pour conserver seulement les messages de détection du HL975 :

```
790a791,799
> [22794.200122] usb 2-1.7: new full-speed USB device number 4 using
ehci-pci
> [22794.315776] usb 2-1.7: New USB device found, idVendor=0403,
idProduct=6001
> [22794.315779] usb 2-1.7: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
> [22794.315781] usb 2-1.7: Product: USB Serial Converter
> [22794.315783] usb 2-1.7: Manufacturer: FTDI
> [22794.315784] usb 2-1.7: SerialNumber: FTE21XLS
> [22794.319035] ftdi_sio 2-1.7:1.0: FTDI USB Serial Device converter
detected
> [22794.319095] usb 2-1.7: Detected FT232RL
> [22794.319544] usb 2-1.7: FTDI USB Serial Device converter now
attached to ttyUSB0
```

Le périphérique USB a été détecté physiquement par le noyau (cf. dmesg), c'est un "**USB Serial Converter**" de FTDI (**idVendor=0403, idProduct=6001**).

Par défaut, le système applique la politique des droits d'accès suivant pour ce type de périphérique USB:

```
$ ls -l /dev/ | grep ttyUSB0
crw-rw----  1 root dialout   188,   0 mai   28 14:20 ttyUSB0
```

Il ne sera accessible qu'en lecture/écriture pour l'utilisateur (*root*) et le groupe (*dialout*). Un utilisateur autre (*other*) n'aura aucun droit d'accès.

Pour une gestion automatique, il faudra passer par `udev` qui est maintenant le service qui prend en charge le répertoire `/dev`.

```
$ sudo vim /etc/udev/rules.d/51.ttyusb.rules

#adaptateur HL 975 TAG HEUER
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001",
MODE="0666", SYMLINK+="hl975"
```

Sous Linux, le HL975 sera maintenant accessible par le fichier **`/dev/hl975`** avec les droits lecture/écriture pour tous les utilisateur.

Diagramme de classes

Le logiciel Chrono-Cross est composé de trois classes. Les classes IHMChronoCross, Course et Chronomètre. La classe IHMChronoCross s'occupe de l'affichage des informations et de l'interaction avec l'organisateur, Course s'occupe de la communication avec la base de données MySQL et la classe Chronomètre de la communication avec le chronomètre HL975 en USB/RS232.

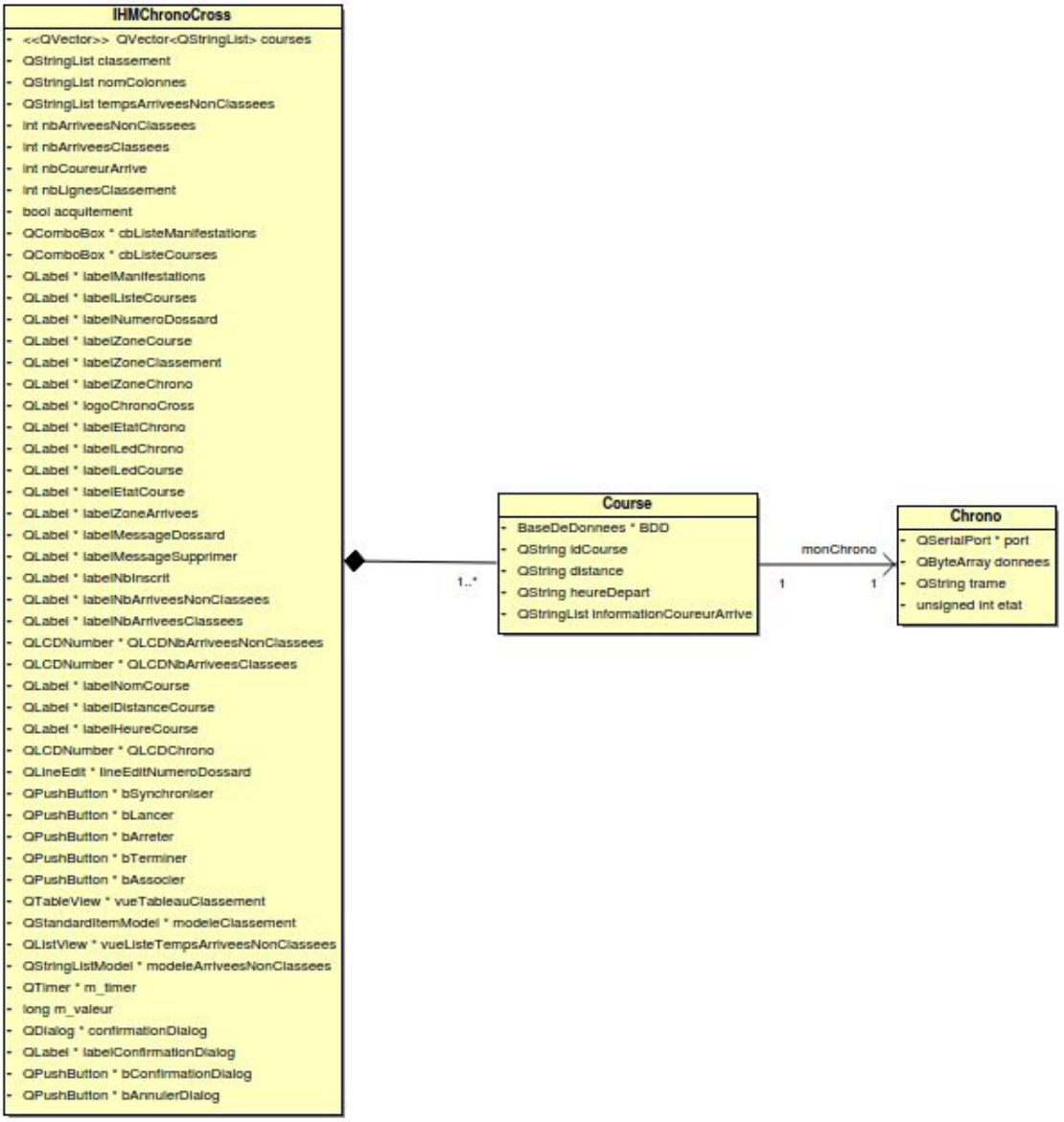
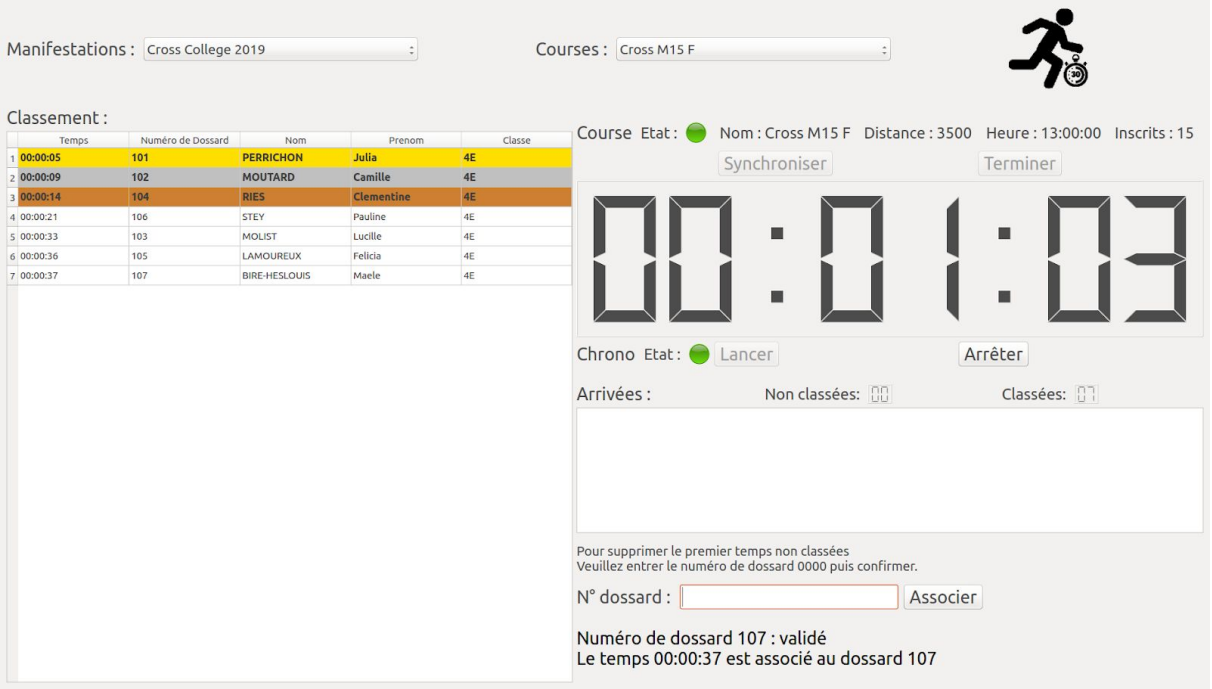


Diagramme de classes du logiciel Chrono-Cross

IHM Chrono-Cross

L'interface homme machine du logiciel Chrono-Cross est la suivante :

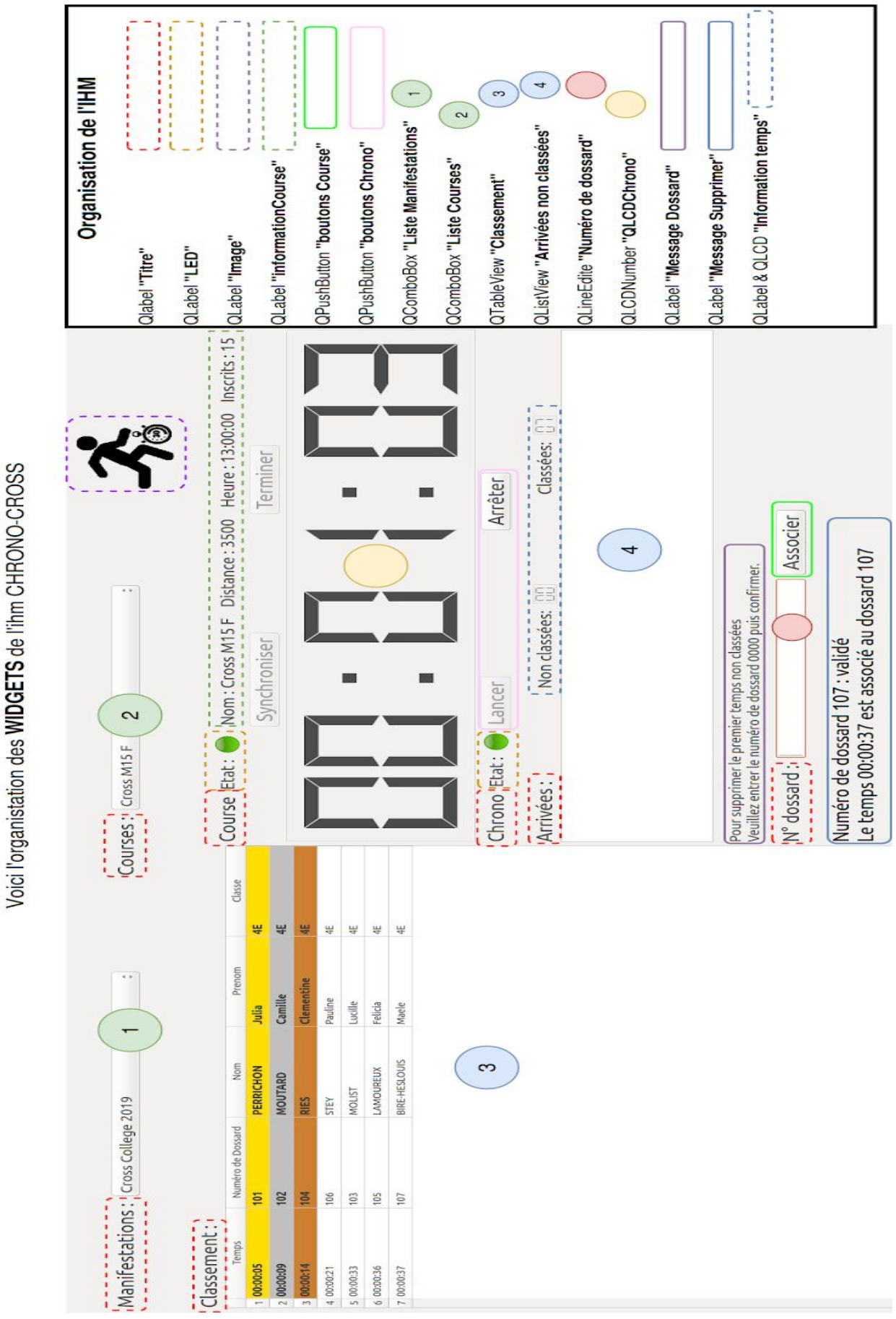


On peut constater la présence de :

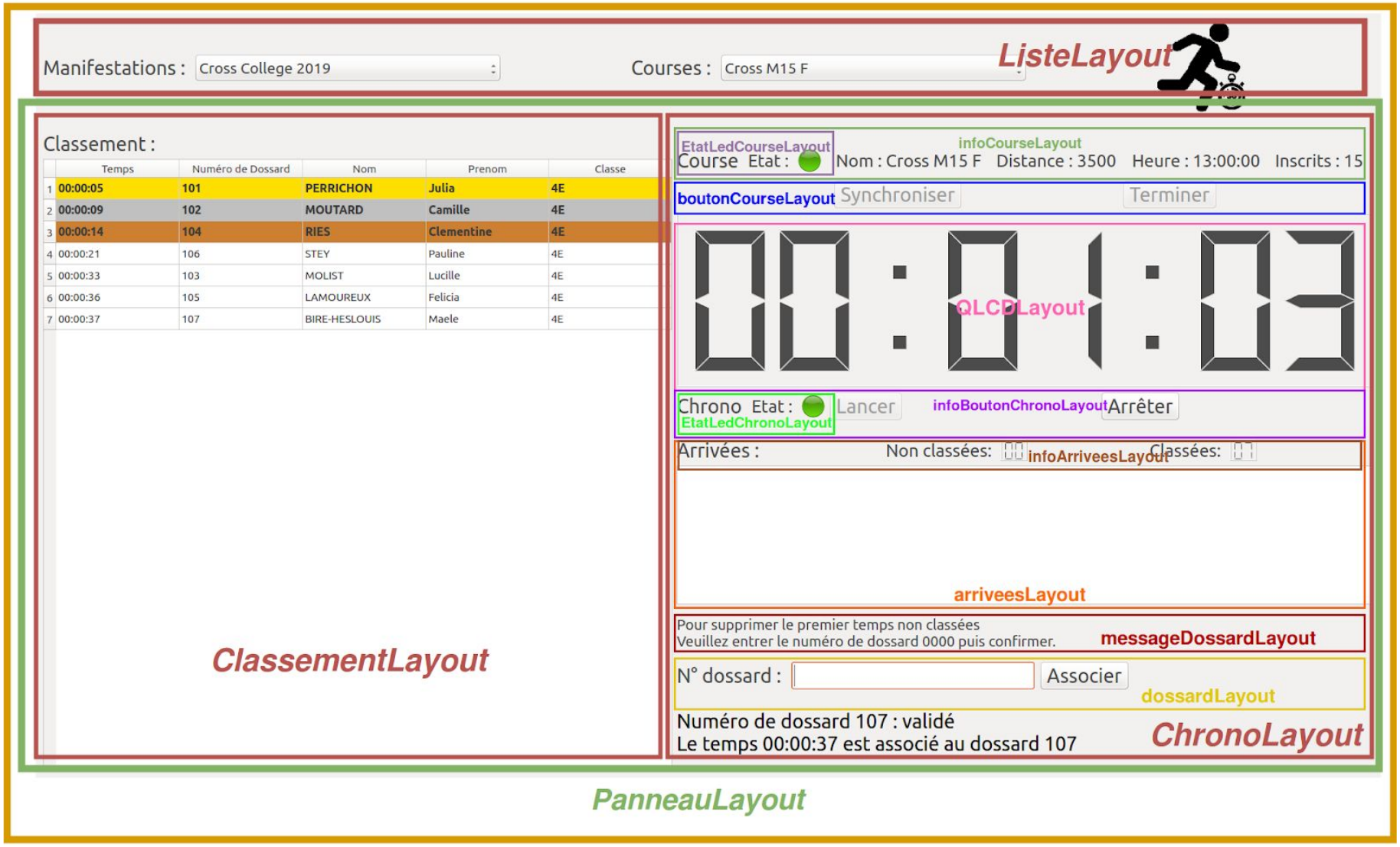
- Deux listes déroulantes qui contiennent la **liste des manifestations** et **des courses**.
- Une zone intitulée **Classement** où s'affiche le classement, les performances des coureurs qui ont déjà franchi la ligne et leur numéro de dossard
- Une zone intitulée Course où l'on peut voir :
  - une **led** qui symbolise l'état du chrono (verte "chrono prêt" ou rouge "chrono pas prêt")
  - un **affichage numérique** qui représente le temps actuelle de la course.
- Une zone chrono où l'on peut voir :
  - une **led** qui symbolise la connection avec le chrono (vert "connectée" ou rouge "déconnectée")
  - quatre boutons : **Démarrer**, **Chronométrer**, **Arrêter** et **Terminer**

- **Bouton Démarrer** : permet de synchroniser l'interface homme-machine avec le chronomètre.
- **Bouton Chronométrer** : lance le chronomètre de la course.
- **Bouton Arrêter** : met fin à la course et au classement.
- **Bouton Terminer** : éteint l'affichage du chronomètre.

- Une zone Arrivées où l'on peut voir :
  - une **zone où s'affiche les performances** dans l'ordre d'arrivée et une zone d'édition du numéro de dossard qui permet de classer un coureur à l'arrivée


Voici l'organisation des WIDGETS de l'ihm CHRONO-CROSS
Organisation de l'IHM
Qlabel "Titre"
QLabel "LED"
QLabel "Image"
QLabel "informationCourse"
QPushButton "boutons Course"
QPushButton "boutons Chrono"
QComboBox "Liste Manifestations"
QComboBox "Liste Courses"
QTableView "Classement"
QListView "Arrivées non classées"
QLineEdite "Numéro de dossard"
QLCDNumber "QLCDChrono"
QLabel "Message Dossard"
QLabel "Message Supprimer"
QLabel & QLCD "Information temps"
Manifestations :
Cross College 2019
Courses :
Cross M15 F
Course
Etat :
Nom : Cross M15 F Distance : 3500 Heure : 13:00:00 Inscrits : 15
Synchroniser
Terminer
Classement :
Temps
Numéro de Dossard
Nom
Prenom
Classe
1 00:00:05 101 PERRICHON Julia 4E
2 00:00:09 102 MOUTARD Camille 4E
3 00:00:14 104 RIES Clementine 4E
4 00:00:21 106 STEY Pauline 4E
5 00:00:33 103 MOLIST Lucille 4E
6 00:00:36 105 LAMOUREUX Felicia 4E
7 00:00:37 107 BIRE-HESLOUIS Maele 4E
Chrono
Etat :
Lancer
Arrêter
Arrivées :
Non classées:
Classées:
Pour supprimer le premier temps non classées
Veuillez entrer le numéro de dossard 0000 puis confirmer.
N° dossard :
Associer
Numéro de dossard 107 : validé
Le temps 00:00:37 est associé au dossard 107
1
2
3
4

Voici l'organistation des **LAYOUTS** de l'ihm CHRONO-CROSS



La classe Course

Elle s'occupe de la communication entre la classe IHMChronoCross, la classe Chrono et la base de données.

Course
- BaseDeDonnees * BDD - QString idCourse - QString distance - QString heureDepart - QStringList informationCoureurArrive
- QString formulerRequeteSelect(QString renseignements, QString sources, QString conditions) + Course(QObject * parent = nullptr) + ~Course() + void creerChrono() + void preparerChrono() + bool estChronometragePret() + void chronometrer() + void arreterClassement() + void arreterChrono() + int verifierDossard(QString dossard) + int getNbInscrit(QString course) + int getDistance(QString course) + QString getHeure(QString course) + int getNbArrivee() + QString getNomCourse(QString dossard) + void setIdCourse(QString nomCourse) + void ajouteArriveeBDD(QString dossard, QString tempsArrivee) + QStringList getListeManifestations() + QVector<QString> getListeCourses(QString manifestation) + void aChronoCree() + void aChronoSynchronise() + void aChronoPret() + void aCommencee() + void aClassementArrete() + void estFinie() + void traiterArriveeCoureur(QString tempsArrivee) + void getInformationCoureur(QString dossard, QString tempsArrivee) + bool setEtatCourse(QString etat) + void chronoCreer() + void chronoCoursePret() + void courseCommence() + void classementArrete() + void courseFinie() + void nouveauTempsArrivee(QString tempsArrivee) + void arriveeAjouteeBDD(QString dossard, QString tempsArrivee) + void informationCoureurRecuperees(QStringList informationCoureur)

*Diagramme de classe de la classe **Course***

Méthodes de classe Course

- string **formulerRequeteSelect**(string renseignements, string sources, string conditions)

  Renvoie une requête select avec les renseignements souhaités, les sources voulues en respectant la condition formulée sous forme d'un string.

- void **CreerChrono**()

  Utilise la méthode creer() de la classe Chrono.

- void **preparerChrono**()

  *Utilise la méthode **bool synchroniser()** de la classe Chrono et affiche en debug l'état retourné.*

- bool **estChronometragePret**()

  *Émet un signal **estConnecte**() pour signaler à la classe IHM que le chronomètre est prêt.*

- void **chronometrer**()

  *Utilise la méthode **bool demarrer()** de la classe Chrono et affiche en debug l'état retourné.*

- void **arreterClassement**()

  *Utilise la méthode **void arreterChrono()** de la classe Chrono.*

- void **arreterChrono()**

  *Utilise la méthode **void arreterChrono()** de la classe Chrono.*

- void **getNbInscrit()**

  Renvoie le nombre d'inscrit d'une course d'après son ID stocké la base de données.

- void **getDistance()**

  Renvoie la distance d'une course d'après son ID stockée la base de données.

- void **getHeure()**

  Renvoie l'heure d'une course d'après son ID stocké dans la base de données

- void **getNbArrivee()**

  Renvoie le nombre d'enregistrement de la table Arrivee.

- void **getNomCourse()**

  Renvoie le nom d'une course d'après le numéro de dossard d'un coureur inscrit à celle-ci.

- void **setIdCourse()**

  Récupère l'idCourse d'un enrgistrement d'après son nom et définit l'attribut idCourse de la classe Course

- void **ajouterArriveeBDD**(string dossard, string tempsArrivee)

  Récupère un idInscrit d'après un numéro de dossard sotké dans la base de données. Ajoute à la table Arrivee un enregistrement avec le tempsArrivee et le numéro de dossard, le classement est l'idArrivee qui s'incrémente automatiquement.

- <list> string **getListeManifestations**()

  Récupère le nombre d'enregistrement de la table Manifestation, puis ajoute toute les manifestations dans un QStringList qu'il renvoie à la classe IHMChronoCross .

- <vector> string **getListeCourses**(string manifestation)

  Récupère l'idManifestation de la manifestation en argument stocké dans la base de données. Ensuite elle récupère le nom des courses qui sont enregistrées avec l'idManifestation et elle les stockent les nom dans un conteneur de type QVector<Qstring> et le renvoie.

Slots de la classe Course

Remarque : un slot est une méthode qui peut être déclenchée en présence d'un signal dans Qt.

- void **aChronoCree()**

  *Émet le signal **chronoCreer()** pour signaler à la classe IHMChronoCross que l'acquitement de la trame MODECLOCK est validé.*

- void **aChronoSynchronise**()

  *Reçoit le signal **chronoSynchronise()** de la classe Chrono qui signal que l'acquitement de la trame NEWSYNCHRO est validé. Utilise alors la méthode **bool creerClassement()** de la classe Chrono et affiche en debug() l'état reçu.*

- void **aClassementCree**()

  *Reçoit le signal **chronoPret()** de la classe Chrono qui signal que l'acquitement de la trame NEWRUN est validé. Émet un signal **chronoCoursePret()** pour signaler à la classe IHM que le chronomètre est prête à être lancé.*

- void **aCommencee()**

  *Reçoit le signal **chronoLance()** de la classe Chrono qui signal que l'acquitement de la trame STARTMANUALSYNCHRO est validé. Émet un signal **courseCommence()** pour signaler à la classe IHM que la course est prête à être lancée.*

- void **aClassementArrete()**

  *Reçoit le signal **courseArretee()** de la classe Chrono qui signal que l'acquitement de la trame CLOSERUN est validé.Émet un signal **classementArrete()** pour signaler à la classe IHM que la course est arretée.*

- void **estFinie()**

  *Reçoit le signal **chronoArrete()** de la classe Chrono qui signal que l'acquitement de la trame MODEND est validé. Émet un signal **courseFinie()** pour signaler à la classe IHM que la course est terminée.*

- void **traiterArriveeCoureur** (string tempsArrivee)

  *Reçoit le signal **nouvelleArrivee()** avec en argument le temps d'arrivée du dernier coureur qui vient de franchir la ligne d'arrivée. Ensuite on formate le temps pour ne récupérer que les heures, les*

*minutes et les secondes. Enfin on comble les trou avec des 0 et l'on émet le signal **nouveauTempsArrivee(tempsArrivee)** pour transmettre à l'IHM le nouveau temps.*

- void **getInformationCoureur**(string dossard, string tempsArrivee)

  Reçoit en argument le numéro de dossard et le temps d'un coureur, on récupère avec son dossard son ID stocké dans la table Inscrit de la base de données.

  *Avec cette information on obtient le nom, le prénom et l'idClasse (qui nous permettra de récupérer la classe de l'élève) de la table coureur. Pour finir on initialise un conteneur de type list<string> que l'on transmet à la classe IHM avec le signal **informationCoureurRecuperees(informationCoureurArrive)**.*

La classe Chrono

Elle s'occupe de la communication avec le chronomètre TAGHEUER via le port RS232.

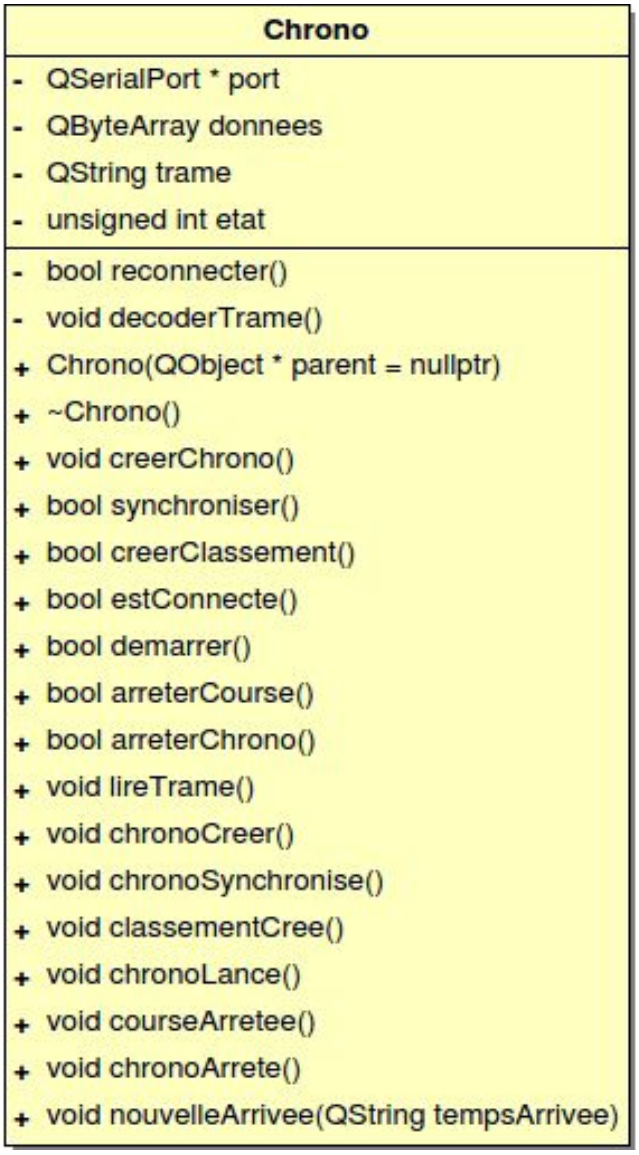


*Diagramme de classe de la classe **Chrono***

Méthodes de classe Chrono

- bool **reconnecter()**

  *Si le port est ouvert, on le ferme puis on essaye de nouveau de l'ouvrir. Si le port est ouvert alors on utilise la méthode **creer()**. On retourne ensuite si la trame a bien été transmise.*

- bool **creer()**

  Si le port est ouvert on émet la trame MODECLOCK ("#WP 120 5\t01AF\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **synchroniser()**

  Si le port est ouvert on émet la trame NEWSYNCHRO ("#WC 007 02 00:00 01/01/01\t048E\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **creerClassement()**

  Si le port est ouvert on émet la trame NEWRUN ("#WC 002\t014C\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **demarrer()**

  Si le port est ouvert on émet la trame STARTMANUALSYNCHRO ("#WC 008 01\t01D3\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **arreterCourse()**

  Si le port est ouvert on émet la trame CLOSERUN ("#WC 001\t014B\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **arreterChrono()**

  Si le port est ouvert on émet la trame MODEND ("#WP 120 3\t01AD\r\n").On retourne ensuite si la trame a bien été transmise.

- bool **estConnecte()**

  Lit le port et retourne l'état du port.

Slots de la classe Chrono

- void **lireTrame**()

  *Stocke dans la variable **donnees** le contenu de la trame tant que le port envoie des lignes de donénes. Si l'attribut **donnees** contient les délimiteurs de fin de trame (" \r\n ") alors on utilise la méthode **decoderTrame()**.*

Méthode decoderTrame()

La méthode **decoderTrame()** permet de décoder la trame et signal les acquittement. Elle permet de faire le tri dans les trames et selon les caractères de début et selon le nombre d'acquittement déjà rencontré effectue une action différentes.

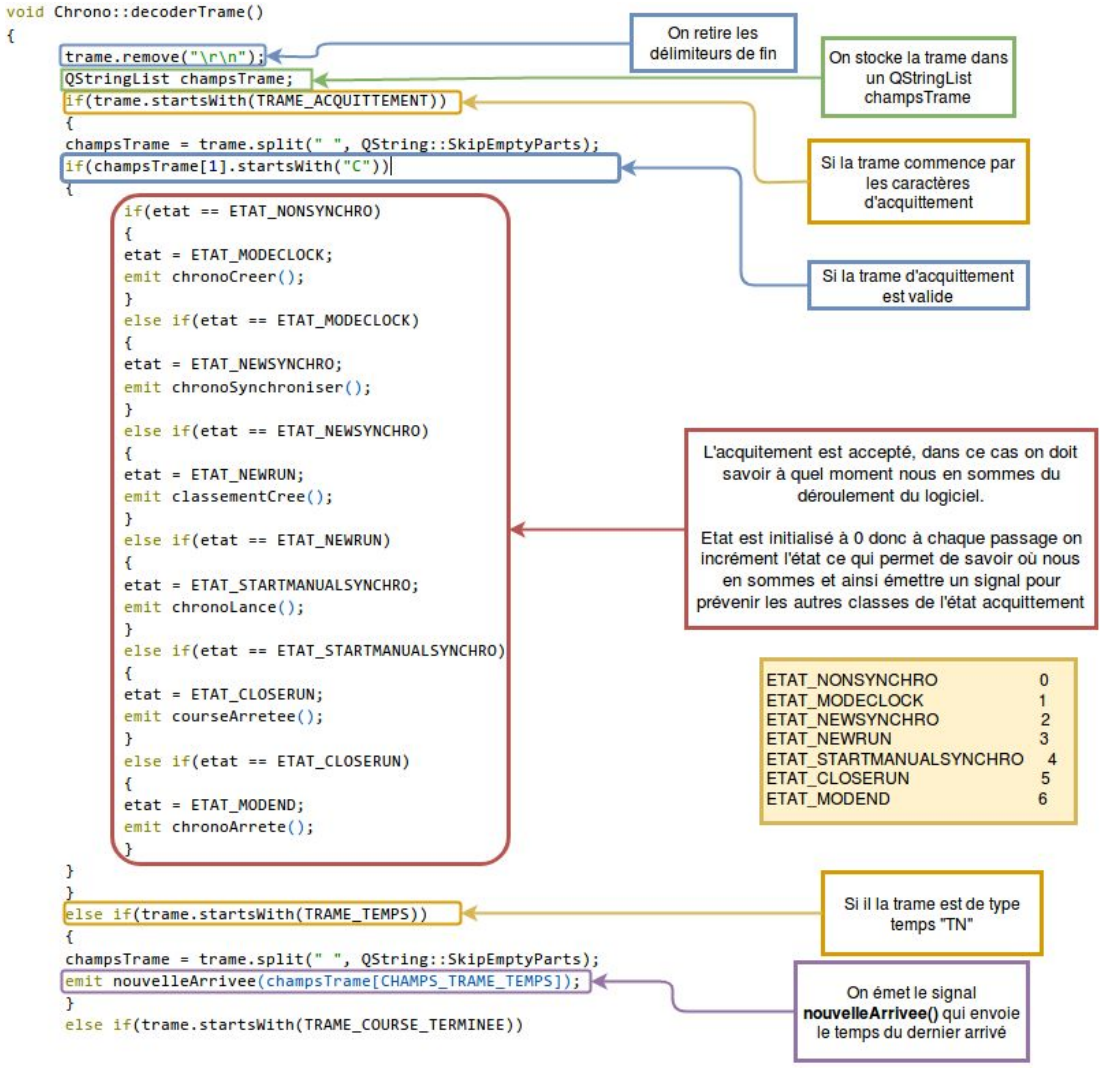


Schéma explicatif de la méthode decoderTrame()

Méthode verifierDossard()

La méthode `verifierDossard()` récupère un numéro de dossard et doit vérifier et celui-ci est valide avant de l'ajouter à la base de données et de le renvoyer vers l'IHM qui le passera de la liste des temps non-classés au classement

```
int Course::verifierDossard(QString dossard)
{
// 0 = Invalide , 1 = numéro valide mais mauvaise course , 2 = valide mais dossard déjà entré , 3 = valide
      int verification = 0;

      QString idInscrit;
      QString coureurs;
      QString condition = QString("NumeroDossard = %1").arg(dossard);

      bool retour = BDD->recuperer(this->formulerRequeteSelect("idInscrit", "Inscrit", condition),idInscrit);

      if(retour)
      {
            // le numéro de dossard existe
            verification = 1;
            condition = QString("idCourse = %1 AND idInscrit = %2;").arg(idCourse).arg(idInscrit);
            retour = BDD->recuperer(this->formulerRequeteSelect("*", "Inscrit", condition), coureurs);
            if(!coureurs.isEmpty())
            {
                  // le numéro de dossard est enregistré pour la bonne course
                  verification = 2;
                  coureurs.clear();
                  condition = QString("idInscrit = %1").arg(idInscrit);
                  retour = BDD->recuperer(this->formulerRequeteSelect("*","Arrivee",condition), coureurs);
                  if(coureurs.isEmpty())
                  {
                        // le coureur n'a pas déjà franchi la ligne d'arrivée
                        verification = 3;
                        return  verification;
                  }
                  else
                  {
                        return verification;
                  }
            }
            else
            {
                  return verification;
            }
      }
      else
      {
            return verification;
      }
}
```

on definit verification que l'on retournera par la suite

on vérifie si le numéro de dossard a été enregistrer

On vérifie si le numéro de dossard est enregistré pour la bonne course

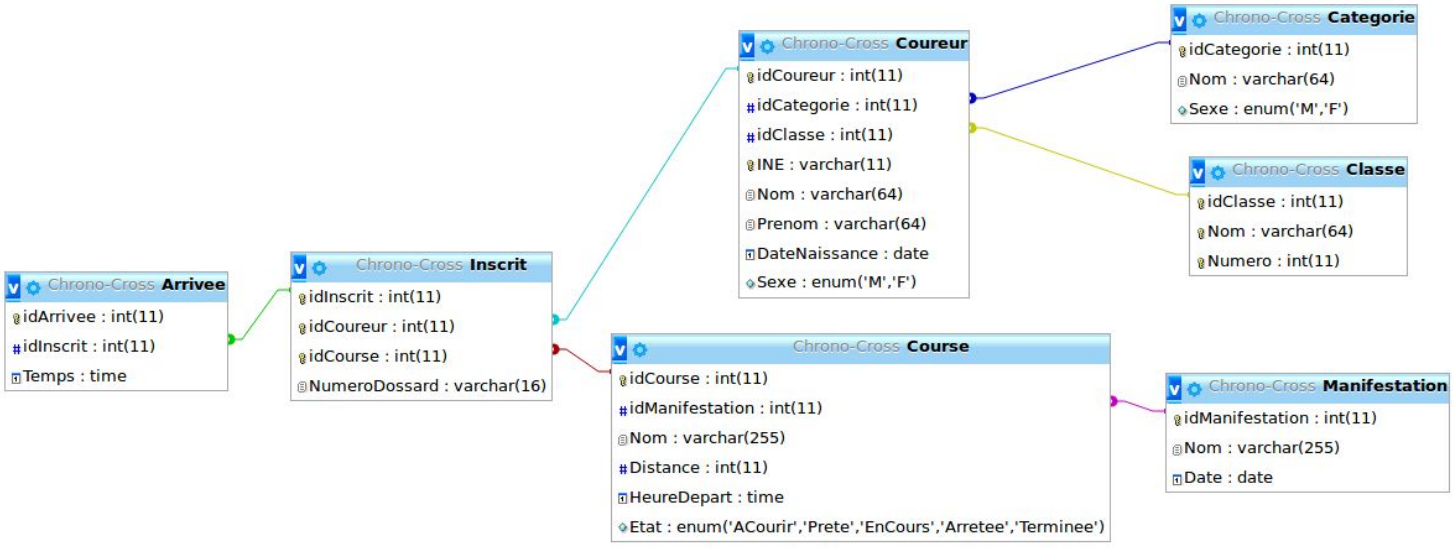
On vérifie si le coureur n'est pas déjà arrivé

Schéma explicatif de la méthode verifierDossard()

Base de données

La base de données Chrono-Cross est composé de **7 tables** :

- Arrivee
  - **idArrivee (primary key)**
  - *idInscrit (foreign key)*
  - Temps
- Inscrit
  - **idInscrit (primary key)**
  - *idCoureur (foreign key)*
  - *idCourse (foreign key)*
  - NumeroDossard
- Coureur
  - **idCoureur (primary key)**
  - *idCategorie (foreign key)*
  - *idClasse (foreign key)*
  - INE
  - Nom
  - Prenom
  - DateNaissance
- Course
  - **idCourse (primary key)**
  - *idManifestation(foreign key)*
  - Nom
  - Distance
  - HeureDepart
  - Etat
- Manifestation
  - **idManifestation(primary key)**
  - Nom
  - Date
- Classe
  - **idClasse (primary key)**
  - Nom
  - Numero
- Catégorie
  - **idCategorie (primary key)**
  - Nom
  - Sexe



Au sein du logiciel Chrono-Cross, on utilise dans la plupart des cas des requêtes **SELECT**. Par exemple, lorsque l'on veut récupérer la liste de course avec la méthode `getListeManifestations()`

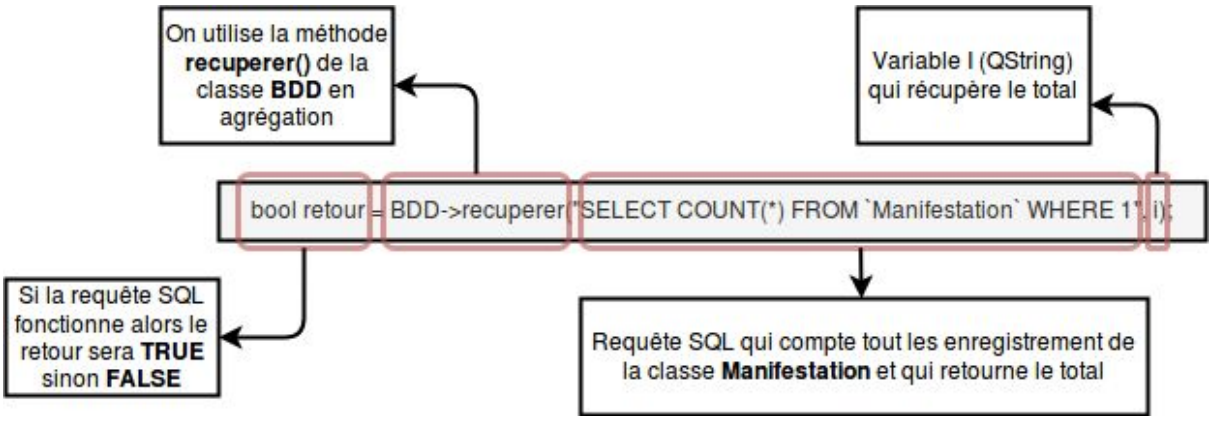
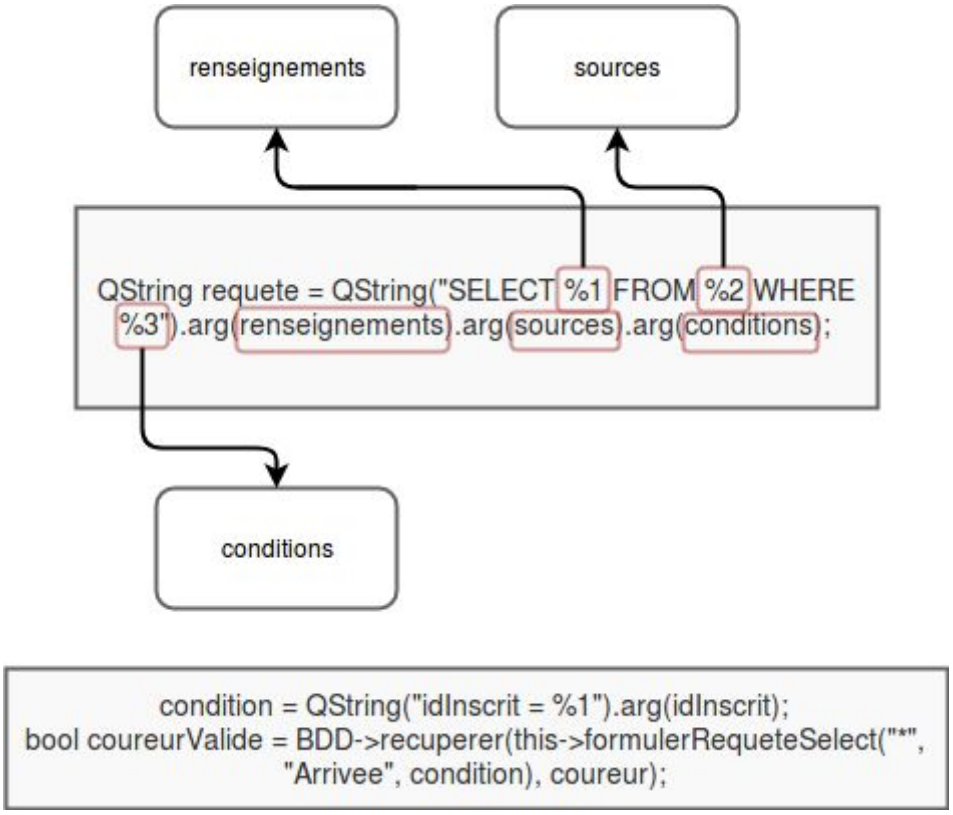


Schéma explicatif de la méthode getListeManifestations()

Ici la requête SELECT permet de retourner le nombre de manifestations dans la table Manifestation.Pour les requêtes SELECT plus classique on a créé une méthode `formulerRequeteSelect()` qui renvoie un string contenant la requête :



Ici la requête SQL récupère les arrivées avec un idInscrit spécifique

Diagrammes de séquence

Sélectionner une manifestation et une course

Dans ce scénario, l'organisateur vient de lancer le logiciel Chrono-Cross. Il doit donc Tout d'abord sélectionner une manifestation puis une course pour pouvoir lancer le chronomètre.

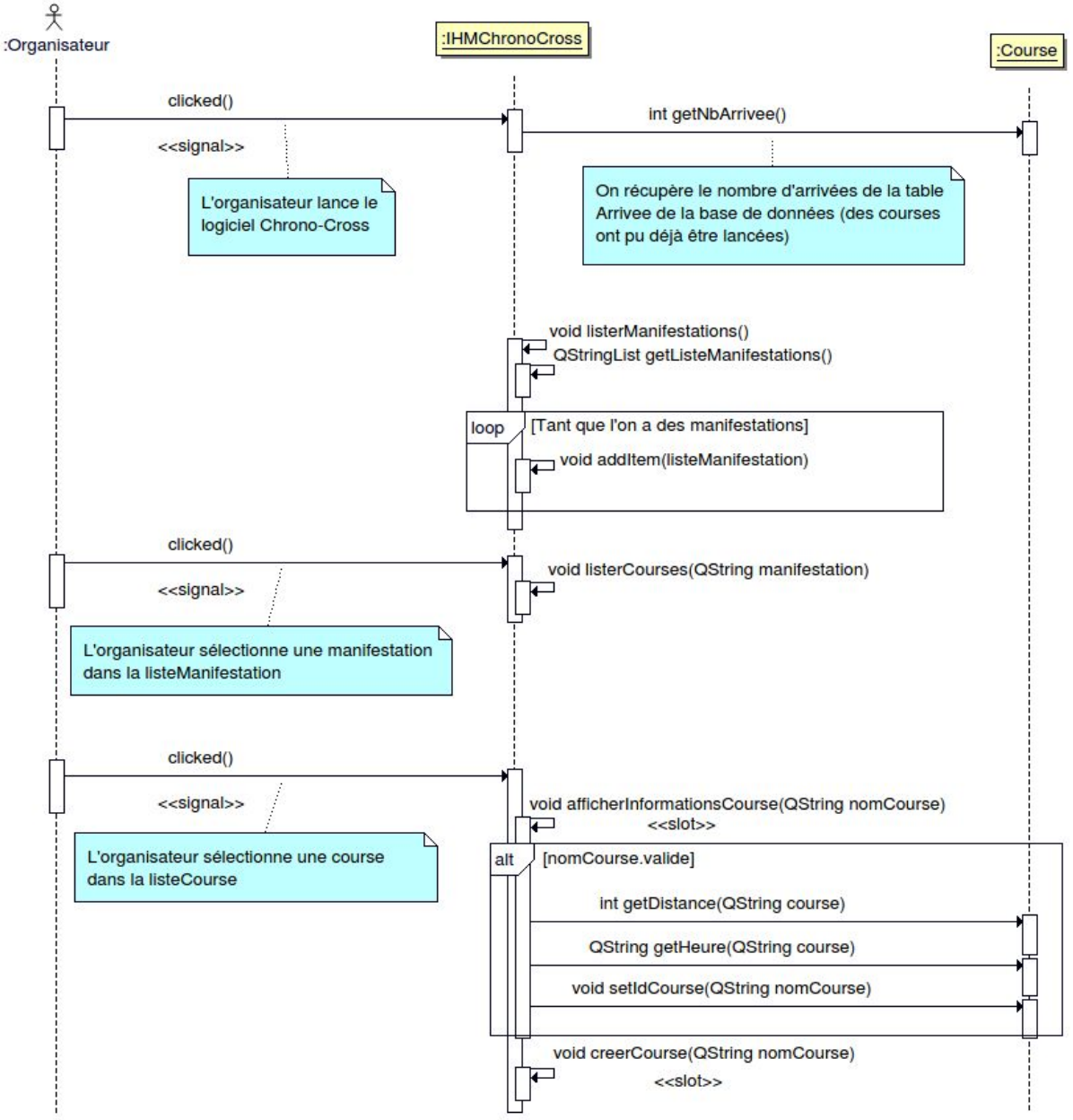


diagramme de séquence "selectionnerManifestationCourse"

Synchroniser le chronomètre et l'IHM

Le scénario suivant consiste à suivre le déroulement du logiciel Chrono-Cross lors de la synchronisation entre le logiciel Chrono-Cross et le chronomètre. L'organisateur au préalable a déjà sélectionné une manifestation dans la liste déroulante puis a sélectionné une course dans la liste déroulante (voir Scénario : Sélectionner une manifestation et une course).

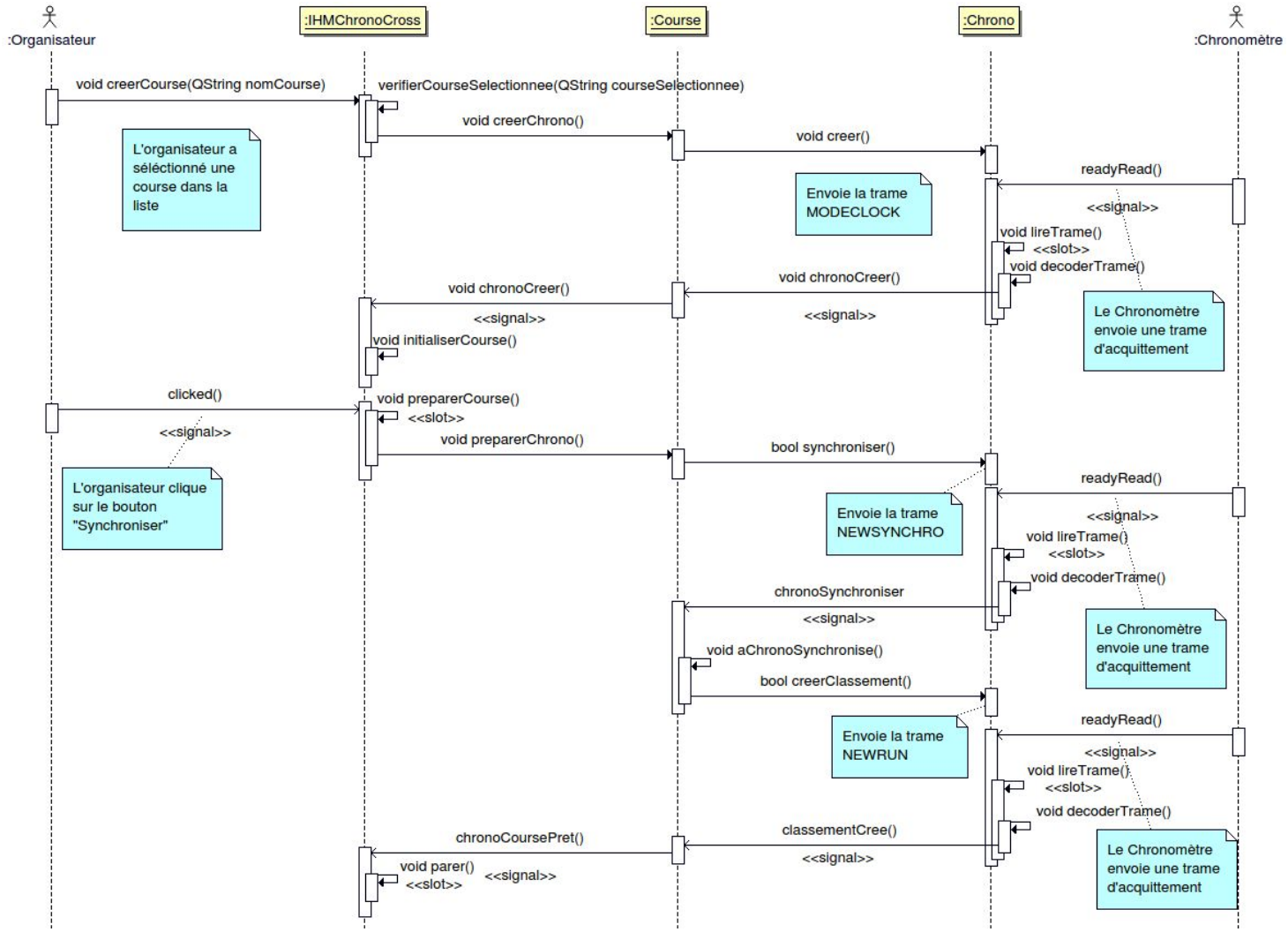


Diagramme de séquence "SynchroniserEtPreparerChronometre"

Lancer et arrêter le chronomètre

Le scénario suivant consiste à suivre le déroulement du logiciel Chrono-Cross lors du lancement à l'arrêt du chronomètre. On part du principe que le chronomètre est déjà synchroniser avec l'IHM (voir Scénario : Synchroniser le chronomètre et l'IHM).

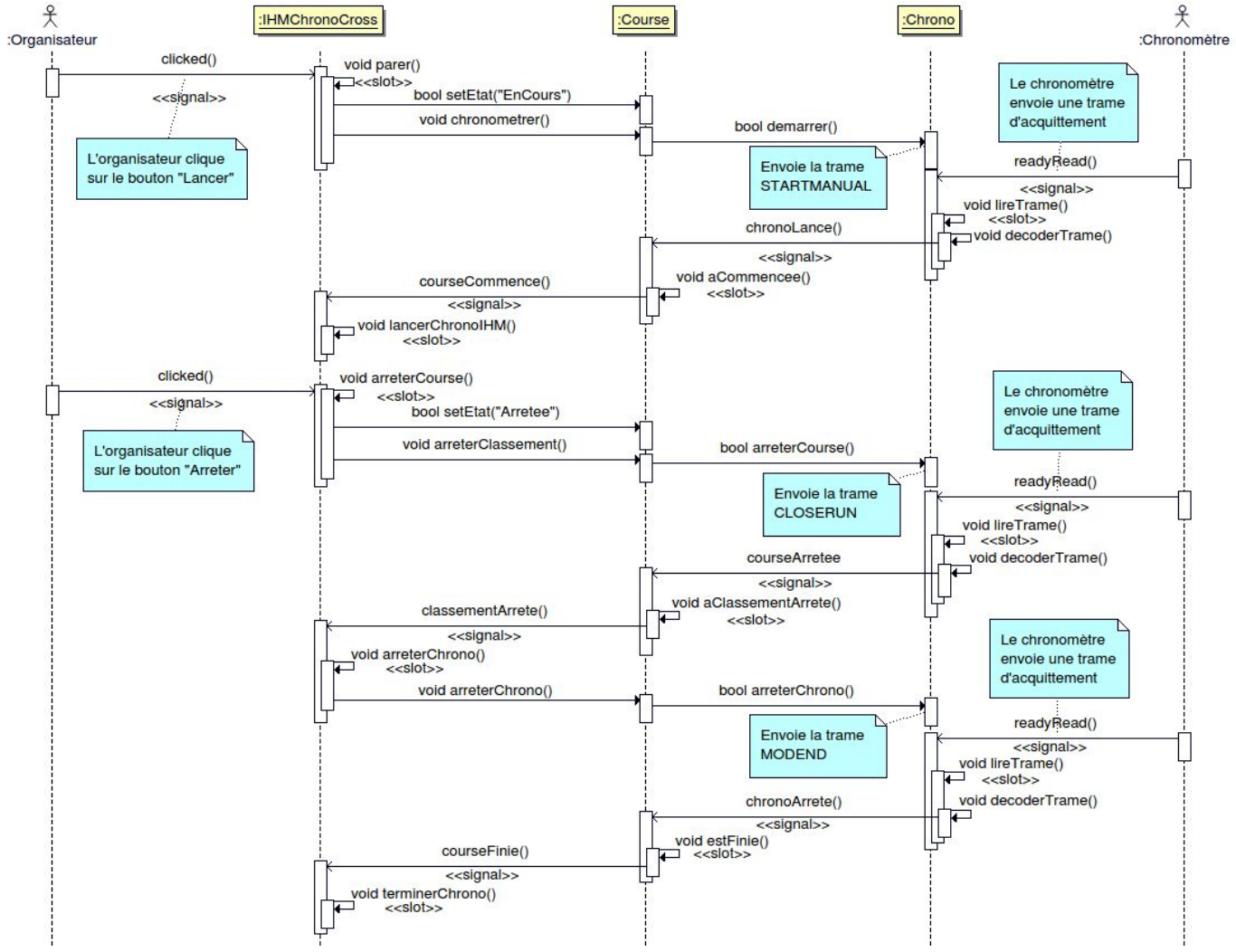


Diagramme de séquence "lancerEtArrêterLeChronomètre"

Réception d'un nouveau temps non-classé

Le scénario suivant consiste à suivre le déroulement du logiciel Chrono-Cross lors de la réception d'un nouveau temps émis par le chronomètre. On part du principe que le chronomètre est déjà synchronisé avec l'IHM et qu'il est lancé.

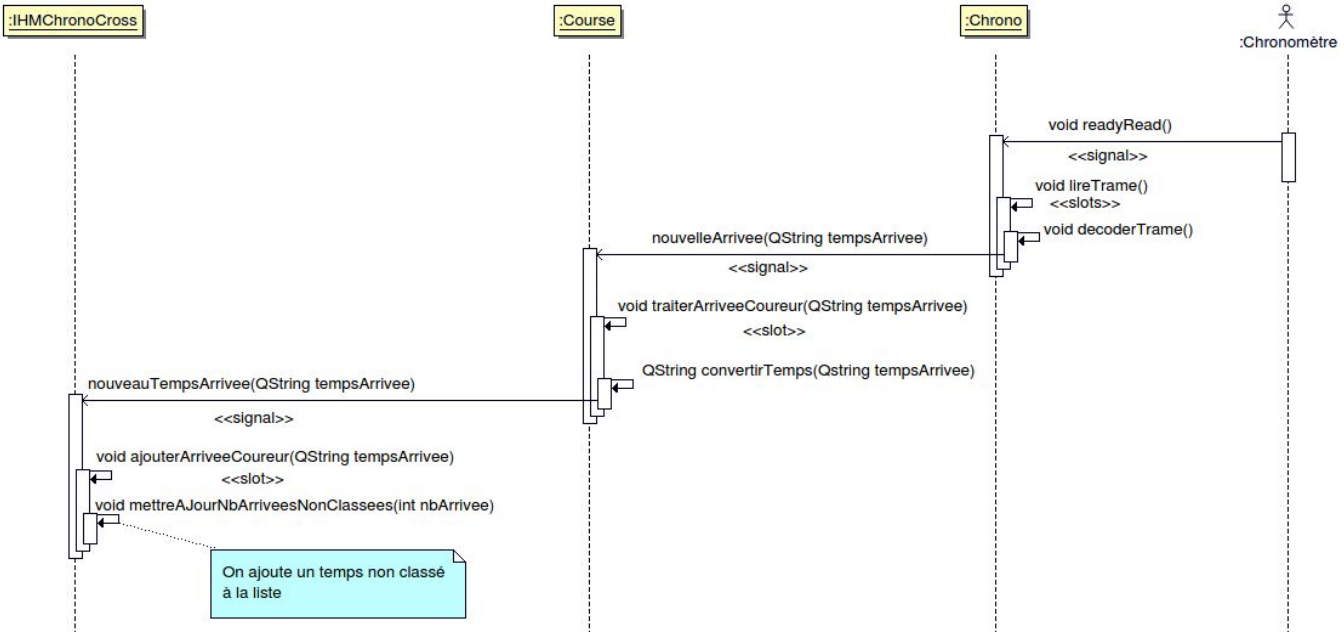


Diagramme de séquence "receptionNouveauTemps"

Associer un numéro de dossard à un temps non-classé

Le scénario suivant consiste à suivre le déroulement du logiciel Chrono-Cross lors de l'association entre un numéro de dossard préalablement entré par l'organisateur et un temps non-classé. On part du principe que le chronomètre est lancé et que l'on a reçu un nouveau temps (voir Scénario : réception d'un nouveau temps non-classé).

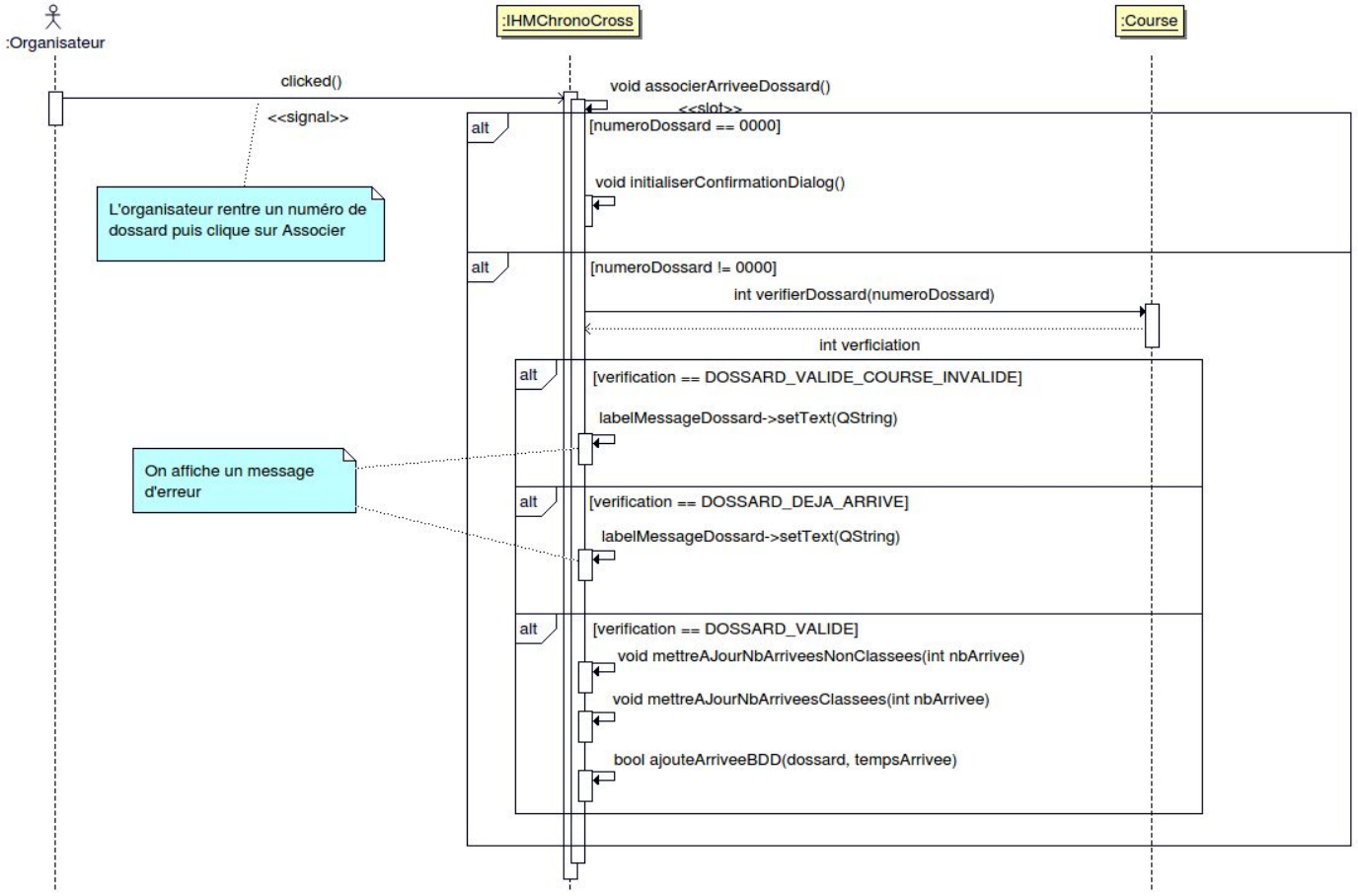


Diagramme de séquence "associerDossardTempsNon-classe"

Terminer une course

Le scénario suivant consiste à suivre le déroulement du logiciel Chrono-Cross lorsque l'organisateur clique sur le bouton "Terminer" pour mettre fin à une course. On part du principe que la course et le chronomètre étaient déjà lancés puis arrêtés.

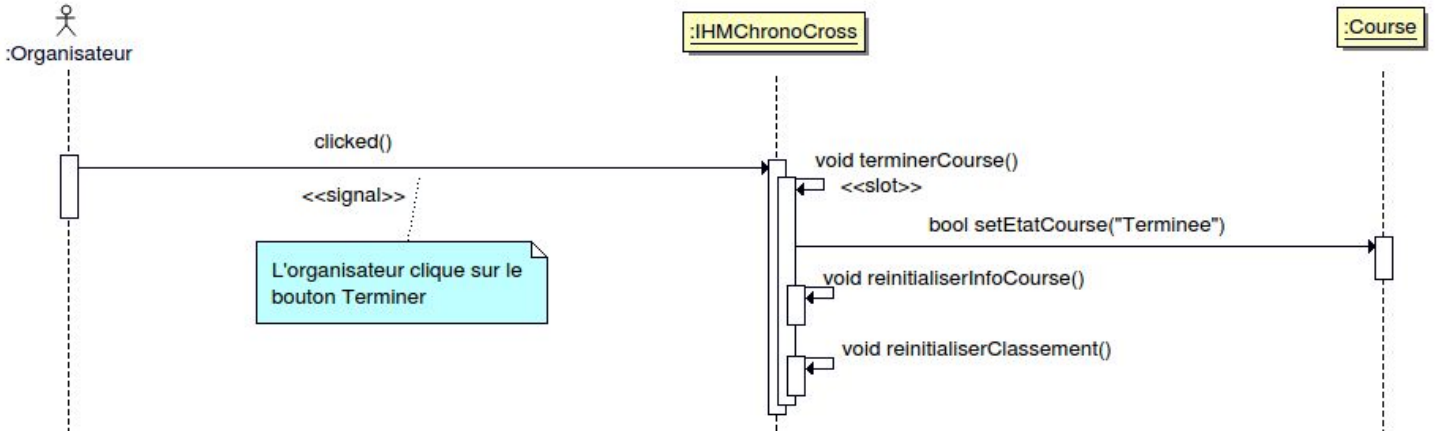


Diagramme de séquence "terminerCourse"

Tests de validation

Désignation	Démarche à suivre	Résultat attendu	Oui / Non	Remarque
Connecter le chronomètre et l'interface homme-machine	- Lancer le logiciel "Chrono-Cross" - Sélectionner une manifestation dans la liste puis une course	La led chrono passe de rouge à orange.	oui	
Synchroniser le chronomètre, l'interface homme-machine et la course	- Lorsque le chronomètre est connecté à l'interface, cliquer sur le bouton "Démarrer"	La led course passe de rouge à orange. La course passe à l'état “Prete”	oui	
Lancer une course	- Lorsque le PC et le chronomètre sont synchronisés - Cliquer sur le bouton “Lancer”	La led course passe au vert. La led chrono passe au vert. Le chronomètre de l'IHM se lance. Le chronomètre TAGHEUER se lance. La course passe à l'état “EnCours”	oui	
Réception d'un nouveau temps	- Un coureur passe devant le capteur infrarouge	Le chronomètre TAGHEUER émet une trame “TN”. La liste des temps non classés affiche un nouveau temps en “HH:MM:SS”.	oui	
Associer un numéro de dossard à un temps non classé	- Lorsqu'un nouveau temps s'affiche, entrer un numéro de dossard valide	Le temps non classés est transféré au classement avec le numéro de dossard et les informations du coureurs.	oui	
Différencier l'affichage pour les trois premiers	- Assigner trois numéros de dossard à des temps non classés	Les temps s'affiche dans le tableau classement avec un texte en gras, le premier est couleur or, le deuxième couleur argent et le troisième couleur bronze.	oui	

Entrer un numéro de dossard invalide	- Entrer un numéro de dossard invalide à un temps non classé	Un message rouge d'erreur apparaît	oui	
Supprimer le premier temps non-classé	- Entrer le numéro de dossard "0000"	Un page s'affiche demandant la confirmation pour la suppression Si l'on confirme le temps est retiré de la liste	oui	
Arrêter une course	- Après que la course a été lancée cliquer sur le bouton "Arreter"	Le chronomètre de l'ihm s'arrête Le chronomètre TAGHEUER s'arrête La course passe à l'état "Arretee"	oui	
Terminer une course	- Après que la course a été arrêtée cliquer sur le bouton "Terminer"	La course passe à l'état "Terminee"	oui	

Logiciel Gestion-Cross

Cas d'utilisation

Le logiciel Gestion-Cross a pour objectif de gérer les **manifestations**, les **courses**, et les **coureurs**. Dans cette partie on s'occupera uniquement de la partie **gestion des coureurs**. Cela passe par la création , la modification ,la suppression et l'inscription d'un coureur à une course.

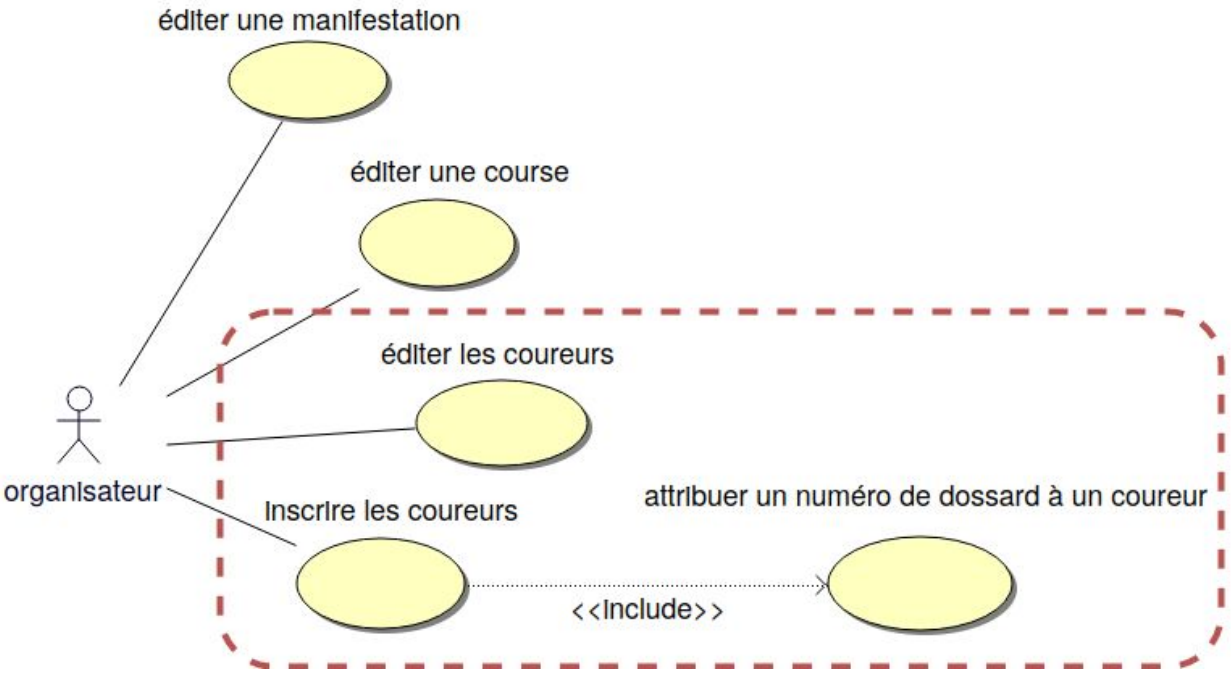
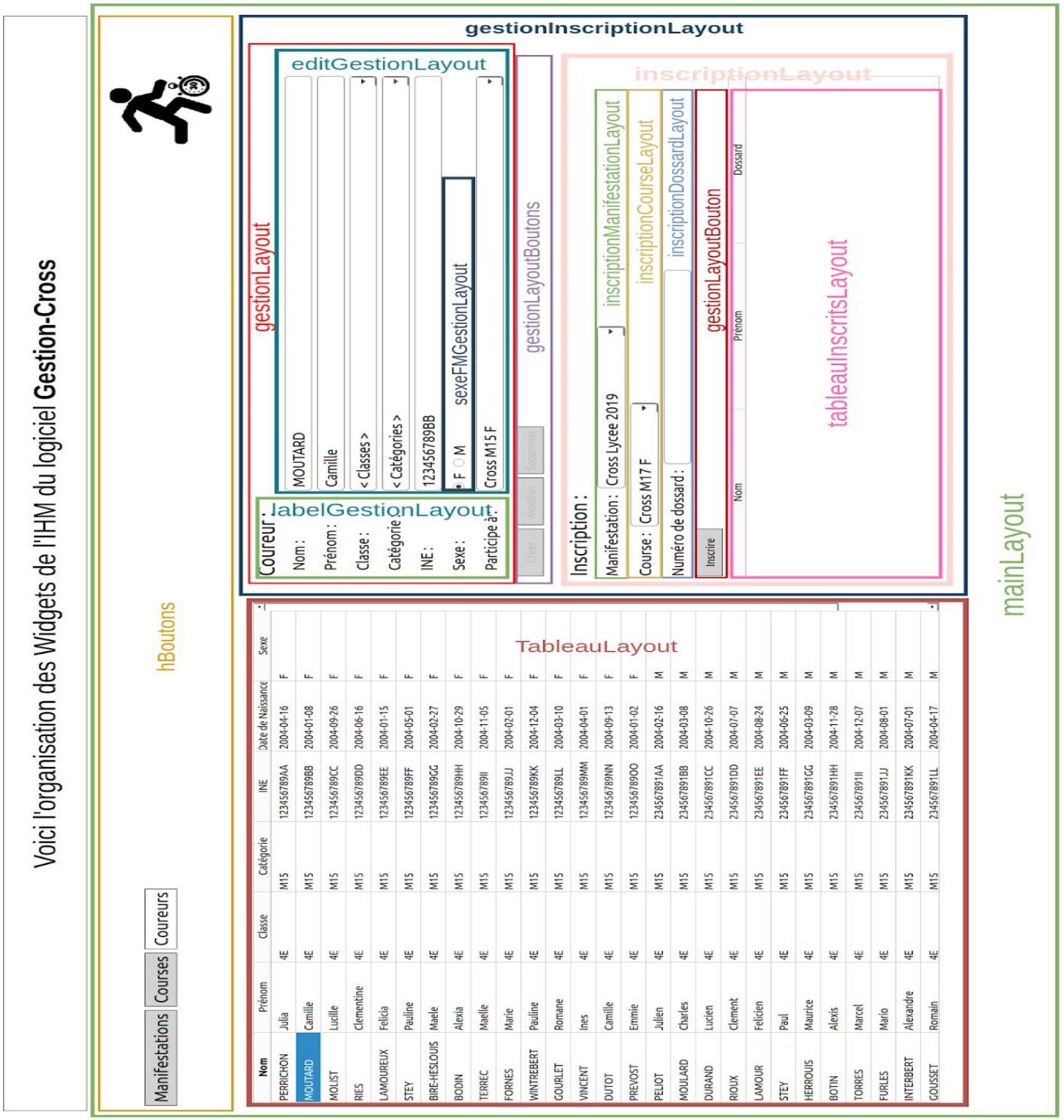


Diagramme de cas d'utilisation pour le logiciel Gestion-Cross

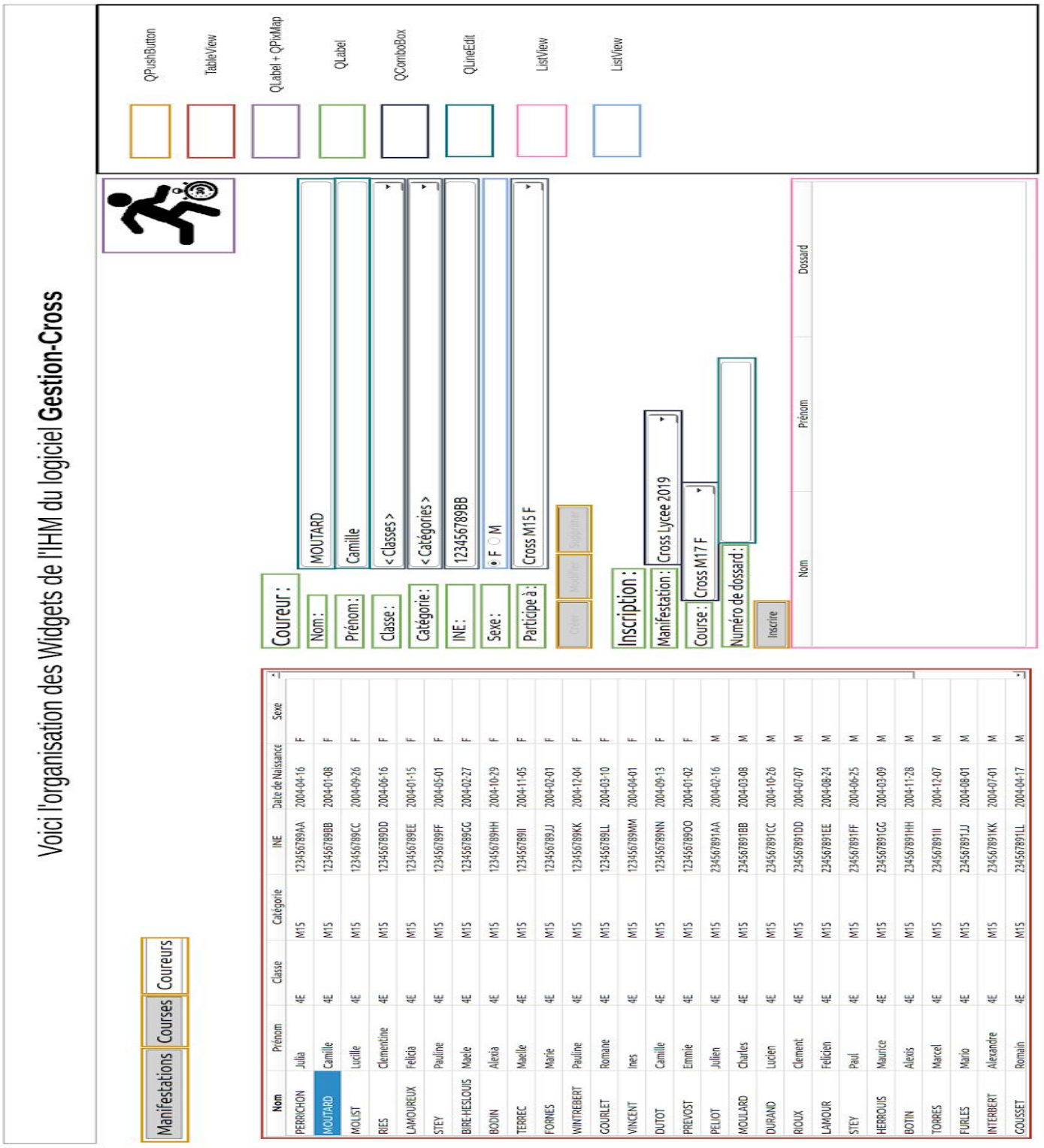
IHM

L'interface homme-machine du logiciel Gestion-Cross est plus sobre que celle du logiciel Chrono-Cross. Voici comment s'organise ses layouts:



Nom	Prénom	Classe	Catégorie	INE	Date de Naissance	Sexe
PERRICHON	Julia	4E	M15	123456789AA	2004-04-16	F
MOUTARD	Camille	4E	M15	123456789BB	2004-01-08	F
MOLIST	Lucille	4E	M15	123456789CC	2004-09-26	F
RIES	Clementine	4E	M15	123456789DD	2004-06-16	F
LAMOUREUX	Felicia	4E	M15	123456789EE	2004-01-15	F
STEY	Pauline	4E	M15	123456789FF	2004-05-01	F
BIRE-HESLOUIS	Maele	4E	M15	123456789GG	2004-02-27	F
BODIN	Alexia	4E	M15	123456789HH	2004-10-29	F
TERREC	Maelle	4E	M15	123456789II	2004-11-05	F
FORNES	Marie	4E	M15	123456789JJ	2004-02-01	F
WINTREBERT	Pauline	4E	M15	123456789KK	2004-12-04	F
COURLET	Romane	4E	M15	123456789LL	2004-03-10	F
VINCENT	Ines	4E	M15	123456789MM	2004-04-01	F
DUTOT	Camille	4E	M15	123456789NN	2004-09-13	F
PREVOST	Emmie	4E	M15	123456789OO	2004-01-02	F
PELIOT	Julien	4E	M15	234567891AA	2004-02-16	M
MOULARD	Charles	4E	M15	234567891BB	2004-03-08	M
DURAND	Lucien	4E	M15	234567891CC	2004-10-26	M
RIOUX	Clement	4E	M15	234567891DD	2004-07-07	M
LAMOUR	Felicien	4E	M15	234567891EE	2004-08-24	M
STEY	Paul	4E	M15	234567891FF	2004-06-25	M
HERROUIS	Maurice	4E	M15	234567891GG	2004-03-09	M
BOTIN	Alexis	4E	M15	234567891HH	2004-11-28	M
TORRES	Marcel	4E	M15	234567891II	2004-12-07	M
FURLES	Mario	4E	M15	234567891JJ	2004-08-01	M
INTERBERT	Alexandre	4E	M15	234567891KK	2004-07-01	M
COUSSET	Romain	4E	M15	234567891LL	2004-04-17	M

De même pour les widgets utilisés, on se retrouve avec des **labels** pour le texte, des **Pushbuttons** pour les boutons, des **ComboBoxs** pour la sélection des informations des coureurs, on a deux **TableViews** pour les deux tableaux des coureurs et le tableaux des coureurs inscrits et enfin on a deux **RadioButton** pour le sexe du coureur.



Nom	Prénom	Classe	Catégorie	INE	Date de Naissance	Sexe
PERRICHON	Julia	4E	M15	123456789AA	2004-04-16	F
MOUTARD	Camille	4E	M15	123456789BB	2004-01-08	F
MOLIST	Lucille	4E	M15	123456789CC	2004-09-26	F
RIES	Clementine	4E	M15	123456789DD	2004-06-16	F
LAMOUREUX	Felicia	4E	M15	123456789EE	2004-01-15	F
STEY	Pauline	4E	M15	123456789FF	2004-05-01	F
BIRE-HESLOUIS	Maele	4E	M15	123456789GG	2004-02-27	F
BODIN	Alexia	4E	M15	123456789HH	2004-10-29	F
TERREC	Maelle	4E	M15	123456789II	2004-11-05	F
FORNES	Marie	4E	M15	123456789JJ	2004-02-01	F
WINTREBERT	Pauline	4E	M15	123456789KK	2004-12-04	F
GOURLET	Romane	4E	M15	123456789LL	2004-03-10	F
VINCENT	Ines	4E	M15	123456789MM	2004-04-01	F
DUTOT	Camille	4E	M15	123456789NN	2004-09-13	F
PREVOST	Emmie	4E	M15	123456789OO	2004-01-02	F
PELIOT	Julien	4E	M15	234567891AA	2004-02-16	M
MOULARD	Charles	4E	M15	234567891BB	2004-03-08	M
DURAND	Lucien	4E	M15	234567891CC	2004-10-26	M
RIOUX	Clement	4E	M15	234567891DD	2004-07-07	M
LAMOUR	Felicien	4E	M15	234567891EE	2004-08-24	M
STEY	Paul	4E	M15	234567891FF	2004-06-25	M
HERROUIS	Maurice	4E	M15	234567891GG	2004-03-09	M
BOTIN	Alexis	4E	M15	234567891HH	2004-11-28	M
TORRES	Marcel	4E	M15	234567891II	2004-12-07	M
FURLES	Mario	4E	M15	234567891JJ	2004-08-01	M
INTERBERT	Alexandre	4E	M15	234567891KK	2004-07-01	M
COUSSET	Romain	4E	M15	234567891LL	2004-04-17	M

Diagramme de classe

Le logiciel **Gestion-Cross** est composé de deux classe: la classe **IHMGestionCross** qui s'occupe de l'affichage, de l'interaction avec l'organisateur et d'afficher les informations récupérées de la base de donnée, nous avons ensuite la classe **GestionBDD** qui communique avec la base de données *Chrono-Cross,* elle envoit des requêtes SQL puis stocke les informations dans des variables qu'elle retourne à l'IHM. Cette même base de donnée sera ensuite utilisée par le logiciel **Chrono-Cross**.

Diagramme de classe du logiciel Gestion-Cross

Base de donnée

Le logiciel **Gestion-Cross** gère la base de donnée du logiciel **Chrono-Cross**, on a donc la même base de donnée. Cependant par rapport au logiciel **Chrono-Cross** on a la nécessité de devoir afficher efficacement différentes informations situées dans différentes tables. On utilise donc des **jointure SQL**.

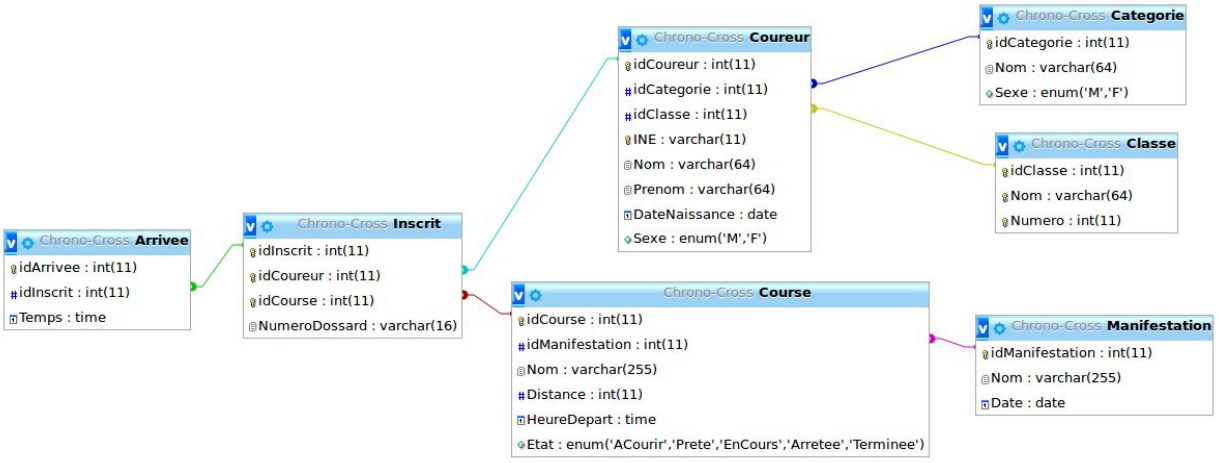


Diagramme de base de données

Jointure

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter les bases de données relationnelles pour obtenir des résultats qui combinent des données de plusieurs tables de manière efficace.

Ici par exemple on a la requête SQL de la méthode `recupererListeCoureursInscrit()`. Cette méthode renvoie une liste contenant le **Nom**, le **Prénom** et le **Numéro de dossard** des coureurs qui participe à une course sélectionnée par l'organisateur.

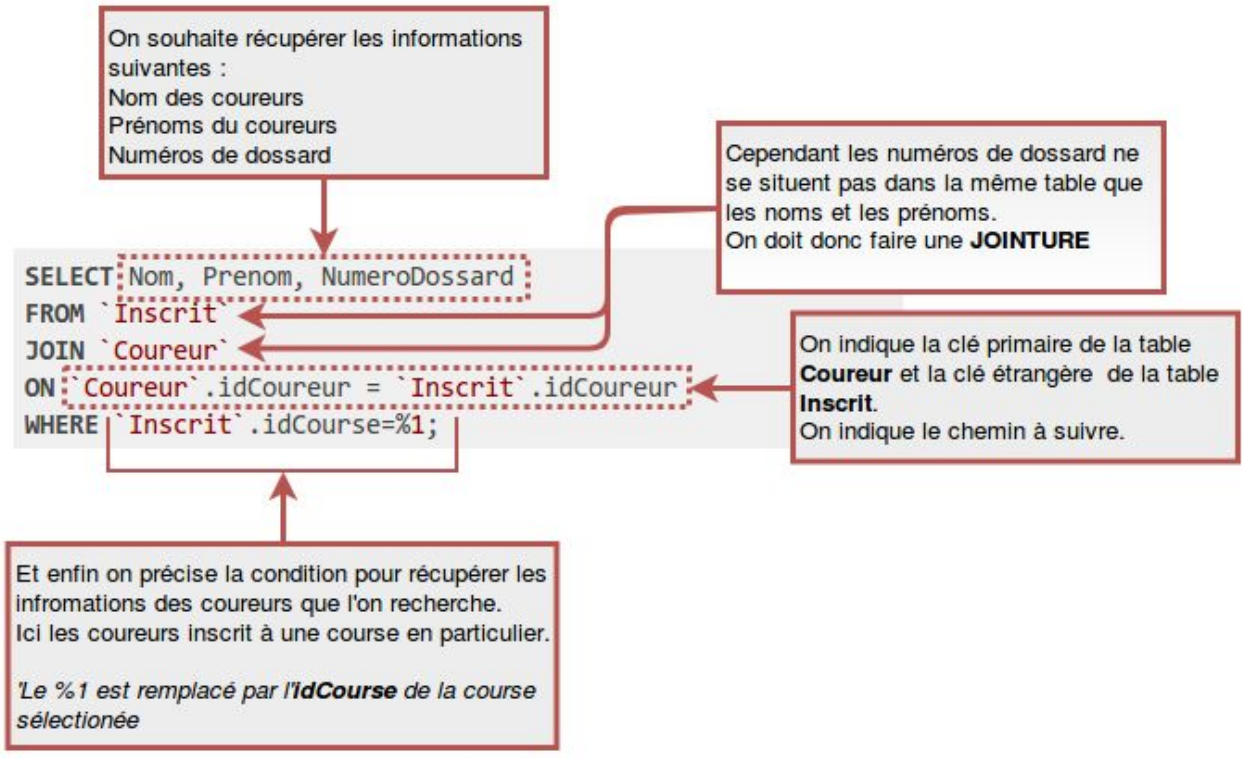


Schéma explicatif de requête SQL avec une jointure

Classe GestionBDD

La classe GestionBDD est la classe qui s'occupe de la communication avec la base de donnée. Elle permet de récupérer ou de modifier des enregistrements (un enregistrement est une ligne entière d'une table de la base de données).

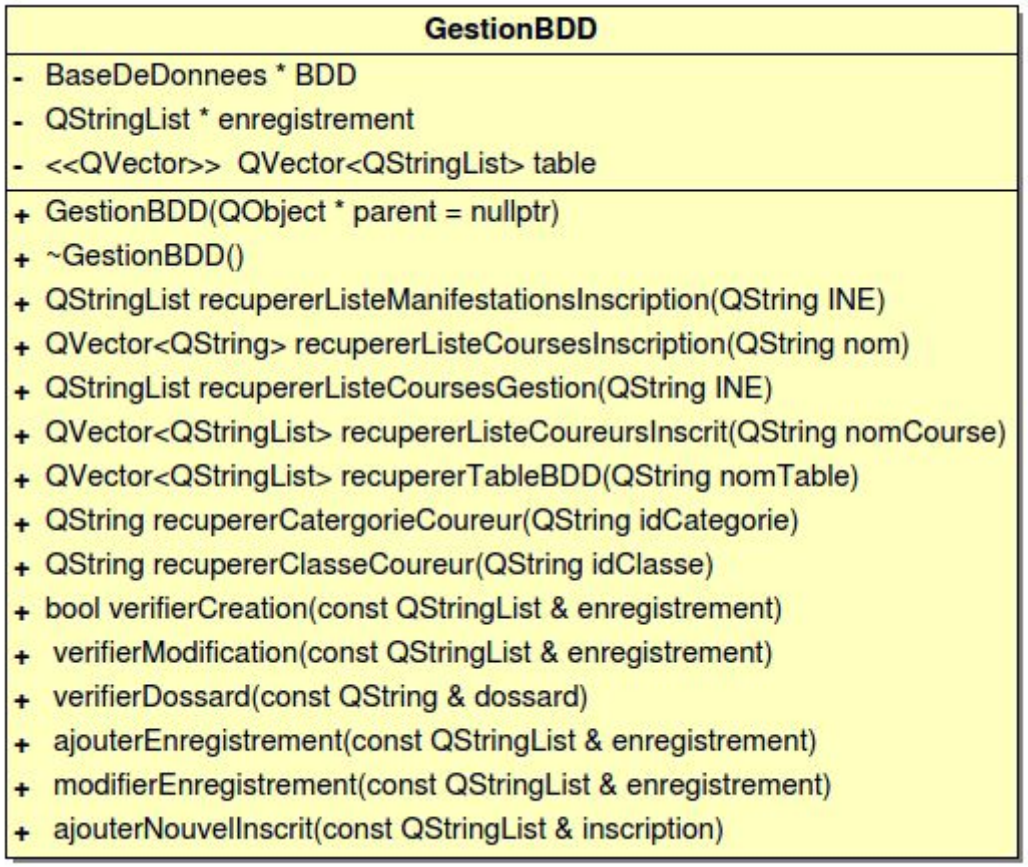


Diagramme de la classe GestionBDD

Attributs:

Cette classe est composé de trois attributs :

- **BaseDeDonnée * BDD** symbolise l'association avec la une classe BaseDeDonnée qui nous aide avec la communication avec la base de donnée
- **QStringList * enregistrements** permet de stocker un enregistrement entier d'une table.

  QStringList : une liste de string donc de chaînes de caractères.

- **QVector<QStringList> * table** qui stocke des QStringList donc des enregistrements entiers.

  QVector : est un tableau dynamique. Autrement dit, les éléments qu'ils contiennent sont stockés les uns à côté des autres.

Méthode

Cette classe est composé de 7 méthodes public qui sont utilisées soit en interne dans GestionBDD.cpp soit le plus souvent sont appelées depuis la classe IHMGestionCross.

- **QStringList recupererListeManifestationsInscription(QString INE)** a en paramètre l'INE d'un étudiants préalablement sélectionné. Elle permet de récupérer le nom des manifestations auxquelles le coureurs **n'est pas inscrit.** Cette méthode est appelée lorsque que l'organisateur souhaite inscrire un coureur à une nouvelle course, on doit donc d'abord récupérer les manifestation disponibles.Pour cela on émet donc la requête SQL suivante avec l'INE en paramètre à la place du "%1":

```
SELECT `Manifestation`.Nom
FROM `Manifestation`
JOIN `Course` ON `Manifestation`.idManifestation =
`Course`.idManifestation
JOIN `Inscrit` ON `Course`.idCourse = `Inscrit`.idCourse
JOIN `Coureur` ON `Inscrit`.idCoureur = `Coureur`.idCoureur
WHERE INE = '%1' ORDER BY Date ASC;
```

  Cette requête nous permet de récupérer le nom des manifestations auxquelles le coureur est inscrit. Ensuite il ne nous reste plus qu'à récupérer le nom des autres manifestations. Elles seront ensuite triées de la plus récents aux plus anciennes. On stocke ces nom dans un QStringList puis on les retournes vers l'iHM.

- **QVector<QString> recupererListeCoursesInscription(QString nom)** a en paramètre le nom de la manifestation préalablement sélectionné par l'organisateur. Cette méthode est appelée après la méthode `recupererListeManifestationsInscription()` pour récupérer les courses disponibles de cette manifestation pour l'inscription d'un coureur. On récupére le nom des courses disponible dans cette manifestation. Pour cela on émet une requête SQL qui récupère les courses disponibles de la manifestation.

- **QStringList recupererListeCoursesGestion(QString INE)** a en paramètre l'INE d'un coureur préalablement sélectionné par l'organisateur. Cette méthode est appelée pour losque que l'organisateur clique sur un coureur du tableau et que ses informations sont transférées dans la partie gestion. On cherche donc à récupérer les noms des courses auxquelles le coureurs est déjà inscrit. Pour cela on émet la requête SQL suivante avec l'INE en paramètre à la place du "%1"

```
SELECT `Course`.idCourse
FROM `Course`
JOIN `Inscrit` ON `Course`.idCourse=`Inscrit`.idCourse
JOIN `Coureur` ON `Inscrit`.idCoureur=`Coureur`.idCoureur
WHERE INE='%1';
```

On effectue deux jointures de la table **Course** vers **Inscrit** puis vers **Coureur**. On stocke les idCourses dans un QStringList. Ensuite on effectue un boucle qui permettra pour chaque idCourse de récupérer le nom de la course associé à l'id. On stocke ensuite le nom des courses dans un QStringList qui sera retourné à l'IHM.

- **QVector<QStringList> recupererListeCoureursInscrit(QString nomCourse)** a en paramètre le nom d'une course préalablement sélectionné par l'organisateur qui souhaite inscrire un coureur à cette course. On cherche donc ici à retourner un QVector de QStringList contenant les informations : Nom, prénom et numéro de dossard des coureurs qui sont déjà inscrit à la course. Pour cela on émet une requête SQL pour récupérer l'id de la course. Ensuite on peut récupérer les noms, prénoms et numéros de dossard des inscrits de cette course. Pour cela on émet la requête SQL suivante avec l'idCourse à la place du "%1" :

```
SELECT `Coureur`.Nom, `Coureur`.Prenom, `Inscrit`.NumeroDossard
FROM `Inscrit`
JOIN `Coureur` ON Inscrit.idCoureur = `Coureur`.idCoureur
WHERE `Inscrit`.idCourse = %1;
```

Enfin on stocke dans un QStringList les informations reçues pour ensuite les retournées vers l'IHM.

- **QVector<QStringList> recupererTableBDD(QString table)** a en paramètre le nom de la table qui correspond au bouton sélectionné par l'organisateur. Ces boutons représente ce que veut modifier l'organisateur

donc **Manifestations**, **Courses** ou **Coureurs** *(cette partie du projet ne concerne que la gestion des coureurs)*. On peut donc émettre une requête SQL pour récupérer toute de la table Coureur que l'on retourne ensuite à l'IHM.

- **QString recupererCatergorieCoureur(QString idCategorie)** a en paramètre l'idCatégorie d'un coureur préalablement sélectionné par l'organisateur qu'il souhaite afficher dans la partie gestion. On récupére le nom de la catégorie du coureur avec son id avec une requête SQL. On stocke ensuite le nom de la catégorie dans un variable de type String que l'on retourne ensuite vers l'IHM.

- **QString recupererClasseCoureur(QString idClasse)** a en paramètre l'idClasse du coureur préalablement sélectionné par l'organisateur qu'il souhaite afficher dans la partie gestion. On émet une requête SQL pour récupérer le nom de la classe d'après son id qu'on retourne ensuite vers l'IHM.

- **bool verifierCreation(QStringList enregistrement)** a en paramètre un QStringList qui contient les informations préalablement entrées par l'organisateur, avant de cliquer sur Créer pour ajouter un coureur à la liste. Pour cela on émet un requête SQL contenant le nom , le prénom et l'INE du nouveau coureur. Si l'on obtient un résultat c'est que le coureur existe déjà et l'on retourne **FAUX** sinon on retourne **VRAI**.

- **void ajouterEnregistrement(QStringList enregistement)** a en paramètre un QStringList qui contient les informations préalablement entrées par l'organisateur, avant de cliquer sur Créer pour ajouter un coureur à la liste. Cette méthode est appelée après la vérification on ajoute donc les informations entrées avec une requête SQL INSERT

```
INSERT INTO `Coureur`(`idCoureur`, `idCategorie`, `idClasse`, `INE`,
`Nom`, `Prenom`, `DateNaissance`, `Sexe`)
VALUES (`idCoureur`,[value],[value],[value],[value],
[value],[value],[value]);
```

- **bool verifierModification(QStringList enregistement)** a en paramètre un QStringList qui contient les informations du coureur dont la modification. On effectue une requête SELECT pour l'INE et vérifier si il n'a pas été modifié (éviter les doublons). Ensuite on regarde pour les autres informations (nom, prenom, classe, catégorie, date de naissance) si elles sont cohérentes.On retourne **VRAI** si elle est correct sinon **FAUX**.

- **void modifierEnregistrement(QStringList enregistement)** a en paramètre un QStringList qui contient les informations du coureur dont la modification.On ajoute à la table un nouveau coureur avec la requête ALTER suivante avec l'idCoureur comme repère (ici on prend 5 pour l'exemple):

```
UPDATE `Coureur`
SET `idCategorie`=[value],`idClasse`=[value],`INE`=[value],
`Nom`=[value],`Prenom`=[value],`DateNaissance`=[value],
`Sexe`=[value]
WHERE idCoureur = 5:
```

- **bool verifierDossard(QString dossard)** a en paramètre le numéro de dossard du coureur que souhaite inscrire l'organisateur. On vérifie d'abord que le permier chiffre soit bien l'idCourse séléctionné. Ensuite que le numéro de dossard soit disponible. On retourne **VRAI** si elle est correct sinon **FAUX**.

- **void enregistrerNouvelInscrit(QStringList inscription)** a en paramètre les informations utile pour l'inscription (numéro de dossard, idCourse, INE). On récupère l'idInscrit et ajoute à la table Inscrit avec la requête SQL INSERT suivante :

```
INSERT INTO `Inscrit`(`idInscrit`, `idCoureur`, `idCourse`,
`NumeroDossard`)
VALUES (`idInscrit`,[value],[value],[value]);
```

Diagramme de séquence

Afficher les coureurs

Dans ce scénario, l'organisateur lance le logiciel **Gestion-Cross**, puis décide d'afficher l'ensemble des coureurs enregistré avec leurs informations.

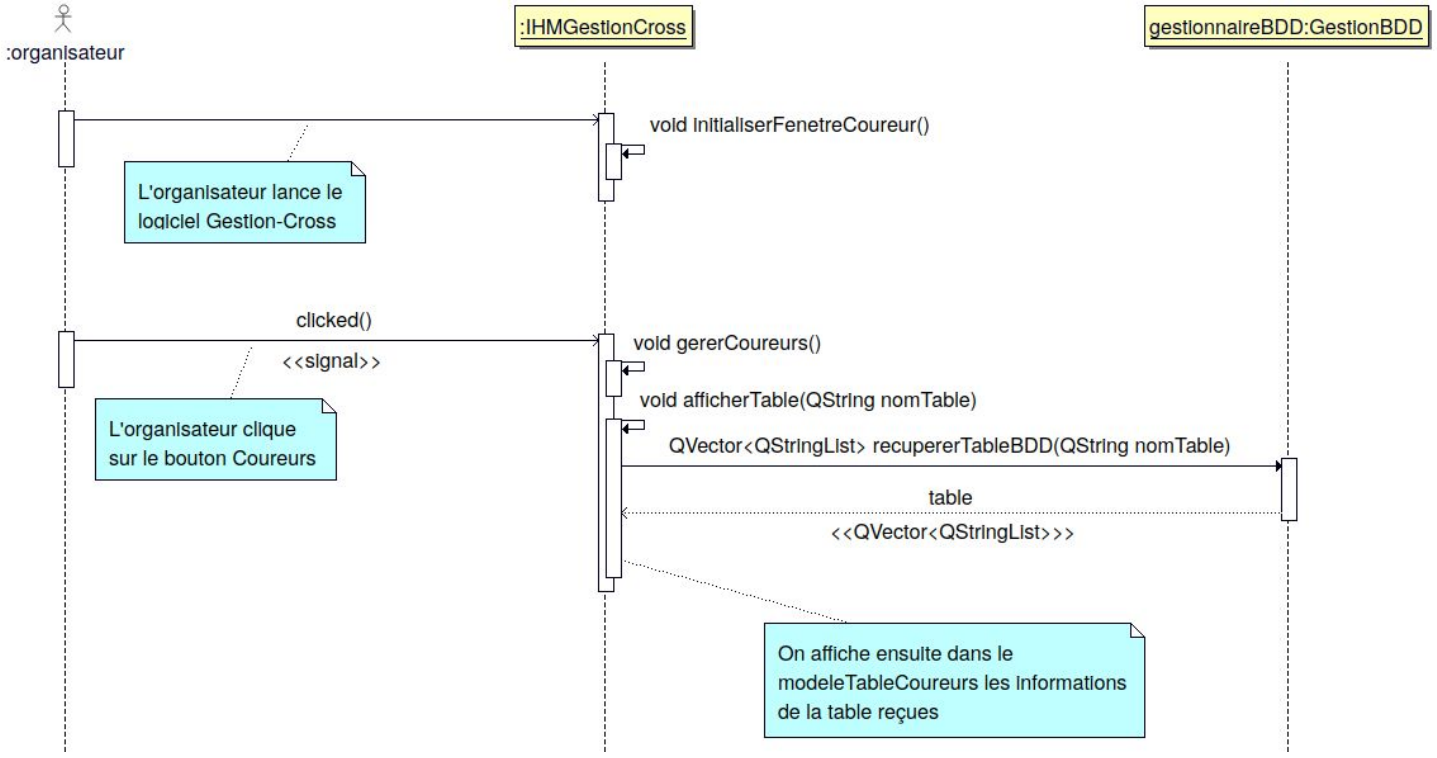


Diagramme de séquence "afficherCoureurs"

Sélectionner un coureur puis l'inscrire à une course

Dans ce scénario, l'organisateur a déjà lancé le logiciel et affiché l'ensemble des coureurs. L'organisateur veut dons sélectionner un coureur spécifique puis l'inscrire à une course d'une manifestation disponible.

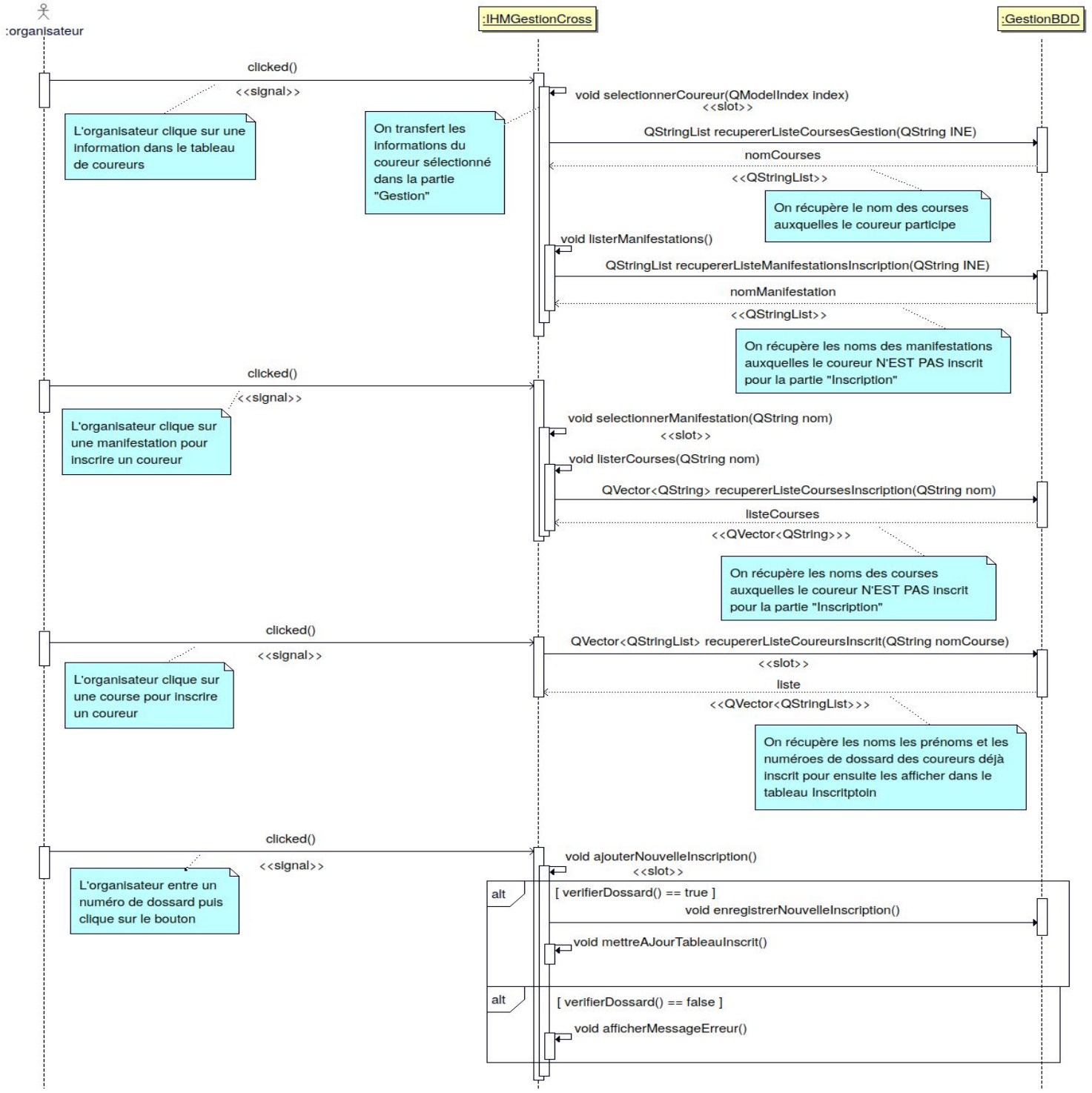


Diagramme de séquence "selectionnerCoureurPuisInscrire"

Créer un coureur

Dans ce scénario, l'organisateur a lancé le logiciel et souhaite créer un nouveau coureur.

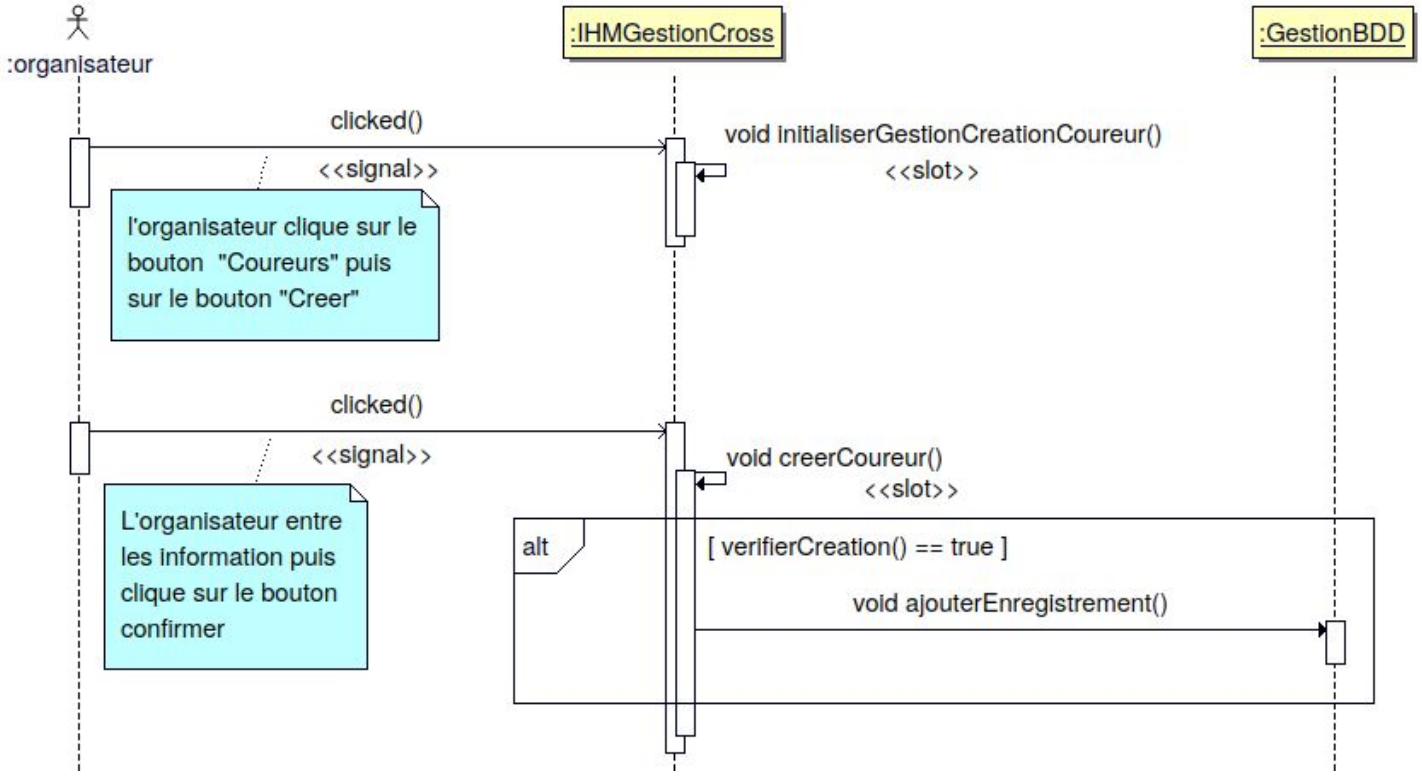


Diagramme de séquence "creerCoureur"

Modifier un Coureur

Dans ce scénario, l'organisateur a déjà lancé le logiciel et affiché l'ensemble des coureurs. L'organisateur souhaite modifier les informations d'un coureur déjà existant.

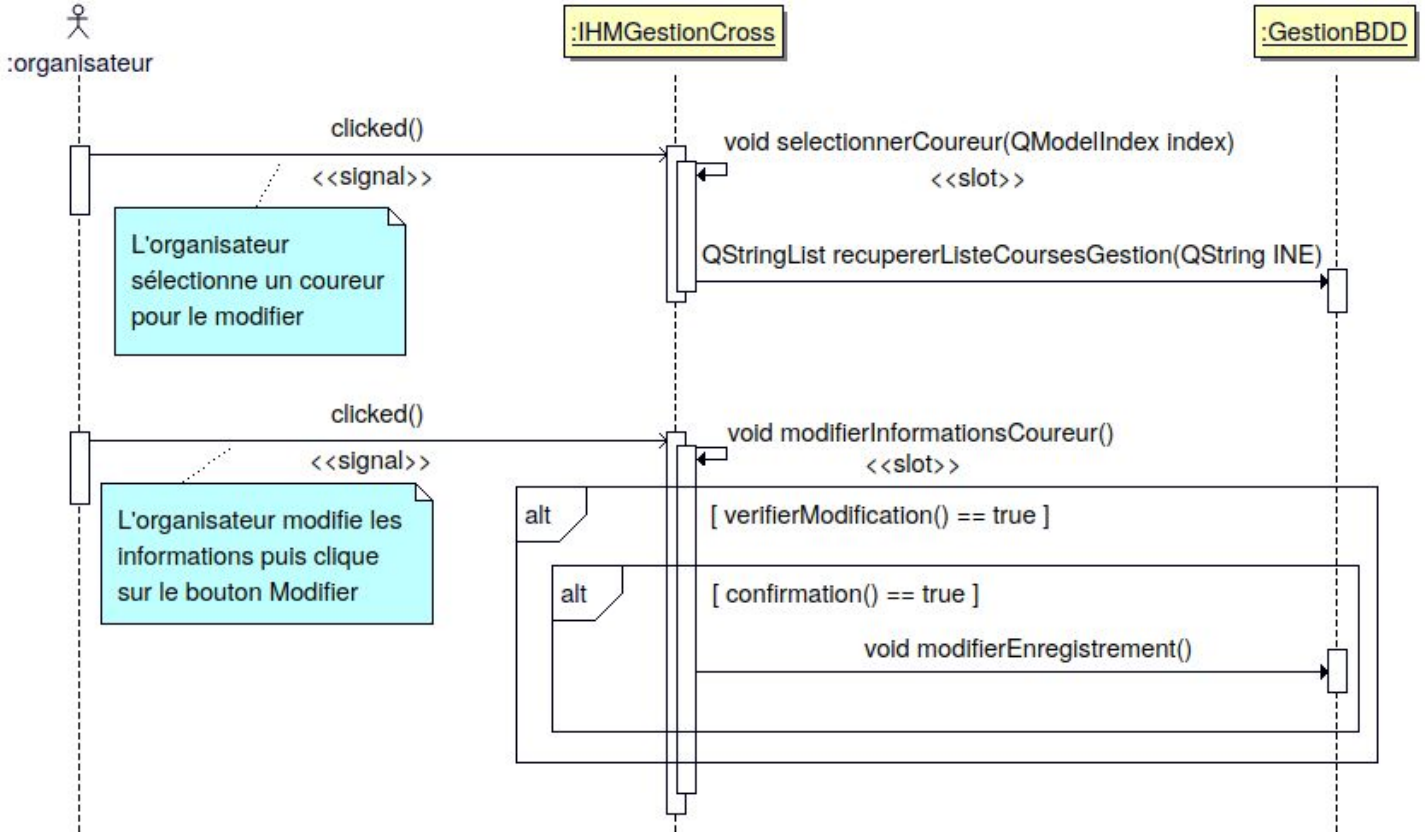


Diagramme de séquence "modifierCoureur"

Supprimer un coureur

Dans ce scénario, l'organisateur a déjà lancé le logiciel et affiché l'ensemble des coureurs. L'organisateur souhaite supprimer un coureur déjà existant.

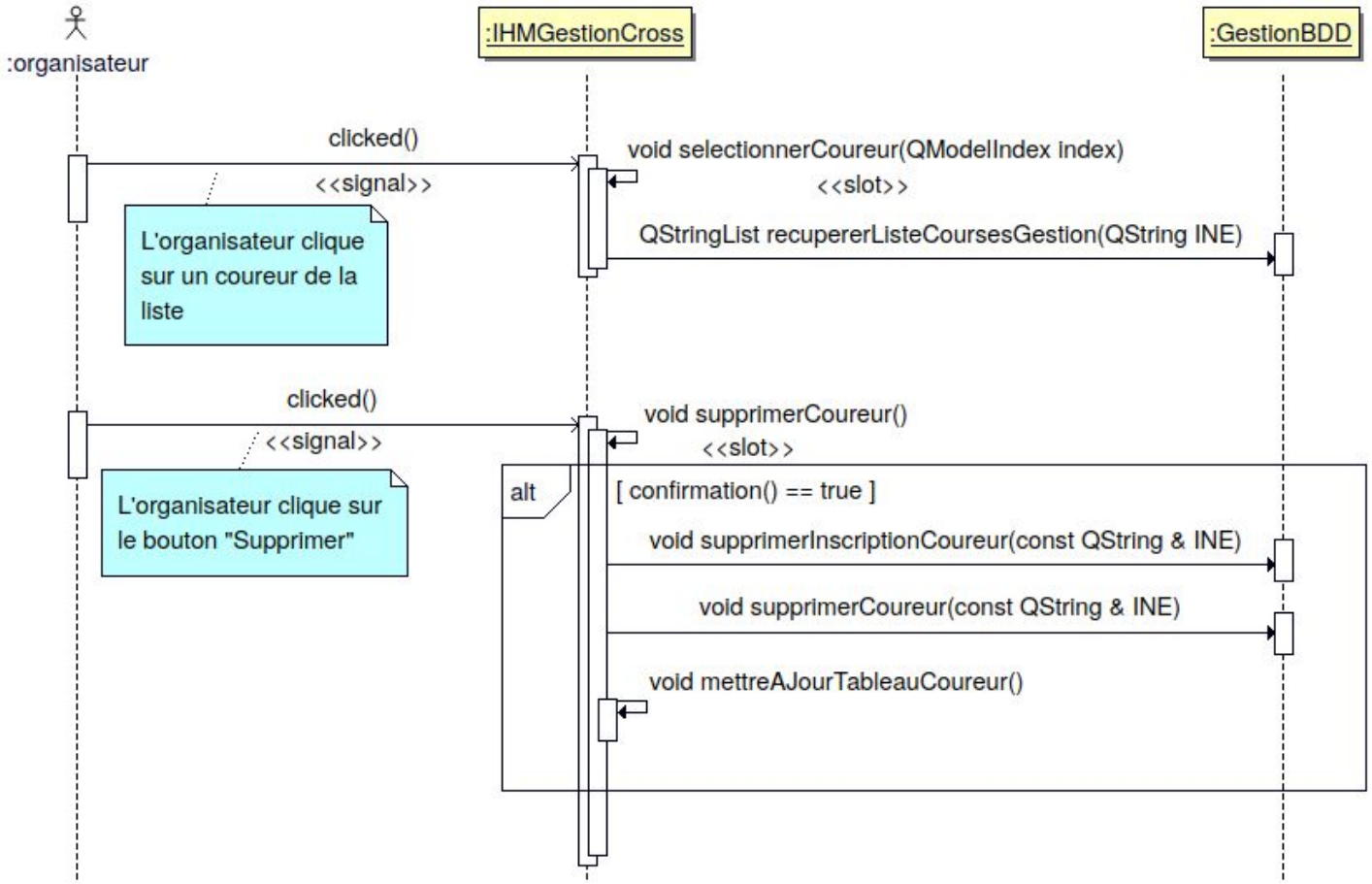


Diagramme de séquence "supprimerCoureur"

Test de validation

Désignation	Démarche à suivre	Résultat attendu	Oui / Non	Remarque
Créer un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton Coureur - Cliquer sur “Nouveau” - Entrer les informations puis cliquer sur créer et confirmer	Un coureur est ajouté à la liste des inscrit, des coureurs et dans la table coureur de la base de données.	oui	
Modifier un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur puis modifier les valeurs - Cliquer sur modifier et confirmer	Les informations du coureur ont été modifiés dans la liste des coureurs et dans la table coureur de la base de données	oui	
Supprimer un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur - Cliquer sur le bouton “Supprimer” et confirmer	Le coureur a été supprimé de la liste des coureurs et dans la table coureur de la base de données	oui	
Entrer des informations erronées	- Dans le logiciel “Gestion-Cross” cliquer, sur le bouton coureur - Cliquer sur modifier puis entrez des informations erronées	Un message d’erreur apparaît et la partie fausse devient rouge Les informations ne sont pas ajoutées à la base de données	non	
Inscrire un coureur à une course	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur - Sélectionner une manifestation puis une course -Entrer un numéro de dossard valide puis cliquez sur le bouton	Le coureur avec son numéro de dossard est ajouté dans la liste des inscrits La table Inscrit de la base de données est mise à jour avec un nouvel enregistrement	oui	

	“Inscrire”			
Afficher tous les coureurs	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur	Le tableau de coureurs se remplit de tous les coureurs enregistré dans la base de données	oui	
Afficher les manifestations disponibles pour un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur	La liste des manifestation disponible se met à jour et montre les manifestations disponible pour ce coureur	oui	
Afficher les courses disponibles pour un coureur	- Dans le logiciel “Gestion-Cross”, cliquer sur le bouton coureur - Sélectionner un coureur - Sélectionner une manifestation disponible	La liste des courses disponible se met à jour et montre les courses disponible pour ce coureur	oui	