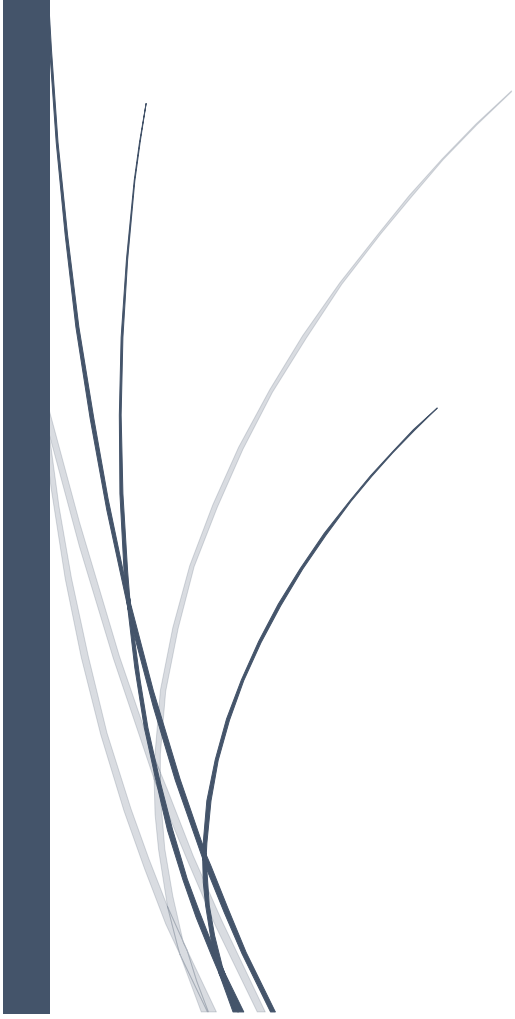
03/11/2021

XF
Programmation temps réel



Métral Sébastien
HES-SO VALAIS-WALLIS

1 TABLE DES MATIERES

2 INTRODUCTION

2.1 CONTEXTE

Le but de ce laboratoire est de mettre en pratique les notions théoriques vues concernant le XF. Nous avons eu un petit aperçu de ce qu'était le XF lors du SummerSchool de cette année. Dans ce laboratoire, le XF sera néanmoins plus complexe que celui abordé en SummerSchool. Les fichiers «.h » nous ont été fournis, il faudra donc compléter les fichiers « .c/.cpp » correspondants. Nous allons tout d'abord réaliser ce laboratoire sur la platefome QT avant de le réaliser sur le système embarqué (Stm 32). Nous avons ensuite 5 tests à disposition afin de valider la fonctionnalité de notre programme.

2.2 XF

Le XF est une petite forme d'un OS (Operating System).

Il contient plusieurs éléments :

- Une queue d'événements
- Un timer manger
- Un dispatcher d'événements
- Ainsi que des mécanismes de protection (dans notre cas Mutex)

Source : Script XF Medard Rieder/Sterren Thomas page 4

Le XF est séparé en deux parties distinctes :

- XF Core : contient des classes qui ne changeront pas selon la plateforme.
- XF Port : contient des classes qui seront, elles, adaptées en fonction de la plateforme

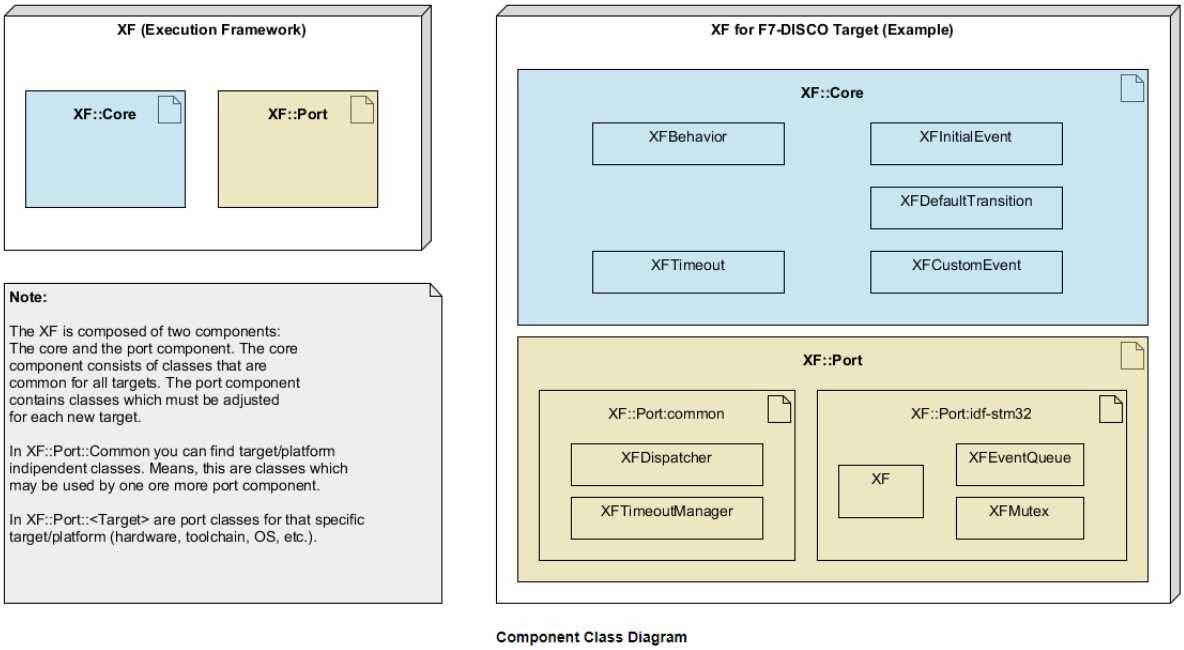


Figure 1 XF core/port class Diagram

Nous pouvons voir sur la figure 1, ci-dessus, que la partie core contient tout ce qui concerne les événements (InitialEvent, Timeout, DefaultTransition, CustomEvent) ainsi que la Behaviour (Machine d'état).

La partie port, quant à elle, est séparée en deux sous-parties. La partie port commun, elle contient les classes qui seront communes aux projets QT et Stm32 (Dispatcher, TimeoutManager), et le port Stm32, lui, est spécifique au STm32 (XF, eventQueue, Mutex).

Source : Html document Simplified XF Medard Rieder/Sterren Thomas Component class Diagram

3 CONCEPTION

Dans cette partie, nous allons détailler chaque élément composant notre XF. Sur toute cette partie conception, les classes en jaune appartiennent à la partie port et les classes en bleu à la partie core du XF.

Toutes les figures appartenant à cette partie proviennent de : *Source : Html document Simplified XF Medard Rieder/Sterren Thomas XF Class Diagram*

3.1 EVENT

Tout d'abord, nous allons parler de la partie concernant le XF Event (événements).

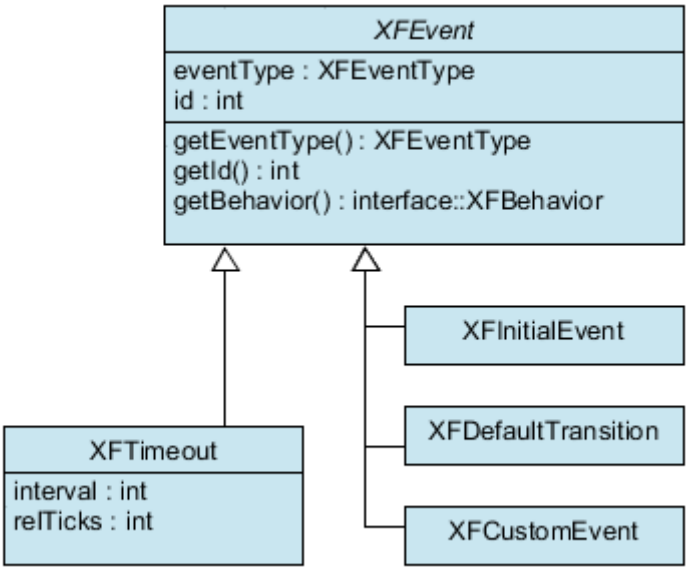


Figure 2 XF Event class Diagram

Sur la figure 2, ci-dessus, nous pouvons remarquer que nous avons une classe de base XF Event, ainsi que plusieurs autres classes qui héritent de cette classe de bases (XF timeout, XF Initial Event, XF DefaultTransition, XF CustomEvent).

XF timeout va permettre de créer des événements de manière retardée (delay).

XF InitialEvent va permettre de créer des événements de type Initial qui auront pour but de faire le premier changement d'état de notre Behaviour (machine d'état).

XF DefaultTransition permet de générer un événement par défaut.

XF CustomEvent permet de générer un événement personnalisé.

3.2 EventQueue

Cette classe XF eventQueue va mettre dans une queue (liste) tous les événements créés. Le dispatcher va pop le premier élément de la queue dès qu'il peut (en fonction des threads). Si la queue est vide, il ne va rien faire.

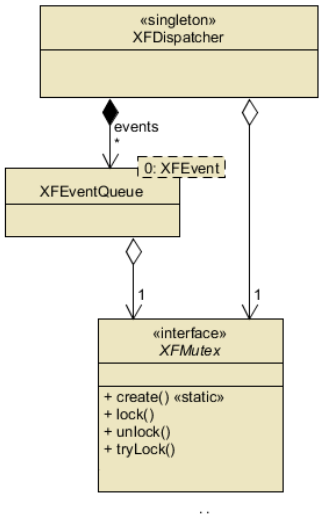


Figure 3 XF EventQueue class Diagram

Cette queue d'événements peut être protégée par un mutex, afin d'éviter que deux threads n'accèdent à cette queue en même temps.

3.3 Mutex (protection mecanism)

La partie Mutex va servir à protéger l'accès à nos listes/queues, plus précisément à la liste de timeouts (XF TimeoutManager) et à la queue d'événement events (XF Dispatcher).

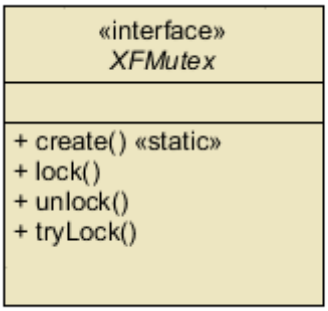


Figure 4 XF Mutex class Diagram

Ce système de protections est utilisé afin d'éviter que plusieurs threads (sur QT) ne puissent accéder en même temps à une de nos listes. En système embarqué, il permet de déclencher/enclencher nos interruptions afin d'empêcher que celles-ci accèdent en même temps que notre programme principal à une des listes.

3.4 TIMEOUTMANAGER

Cette classe est une classe singleton, cela signifie qu'elle ne possède qu'un seul objet.

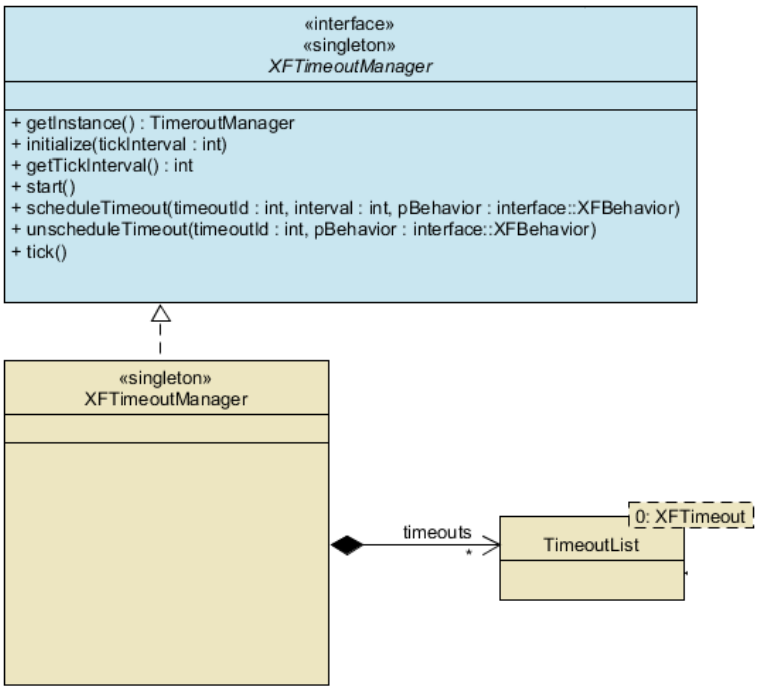


Figure 5 XF Timeout Manager class Diagram

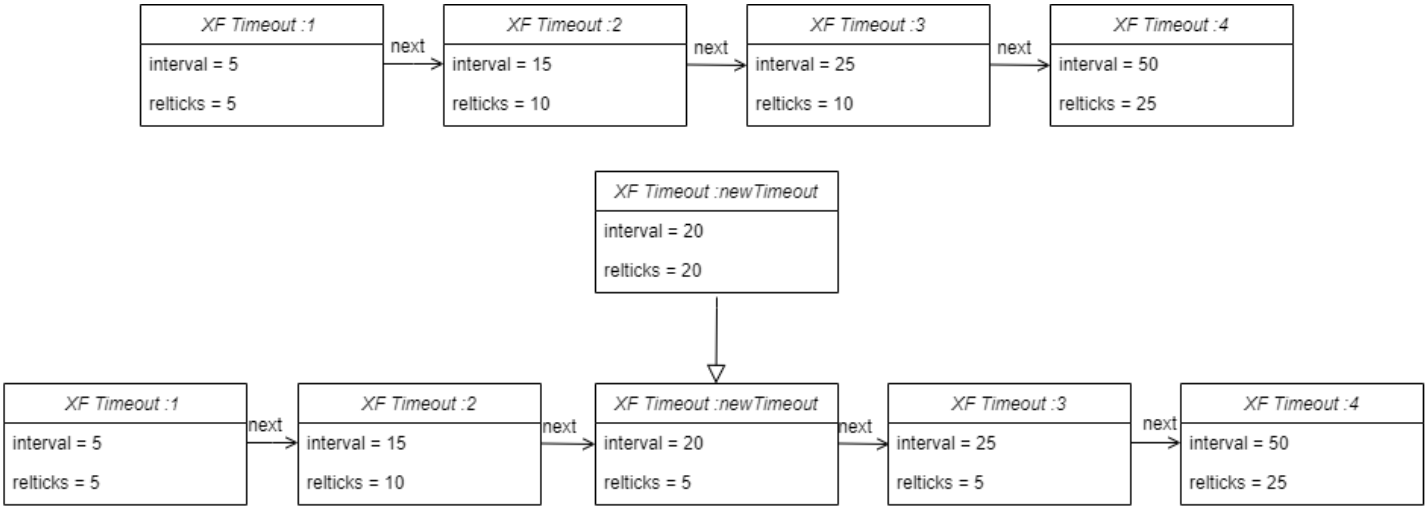
Cette classe possède comme attribut une liste de XF Timeout nommé timeouts.

Cette classe va nous permettre de gérer cette liste avec plusieurs méthodes :

- getInstance() : permet de retourner le seul objet de la classe
- Initialize(…) : permet d'initialiser le temps de tick de notre timer
- Start() : permet de créer notre timer
- ScheduleTimeout(…) : va permettre de créer un XF timeout (voir figure 2) et de l'ajouter à la liste timeouts avec le bon retard (delay). Une fois ce delay effectué, un objet XF Event sera créé et ajouté à la liste d'événements.
- UnSchedeleTimeout(…) : va permettre de supprimer un XF Timeout de la liste timeouts
- Tick() : méthode qui est appelée, à chaque tick du timer. Cette méthode va permettre de décrémenter l'attribut relTicks du premier objet XF Timeout présent dans notre liste et va ensuite push un XF event dans la queue d'événements du dispatcher, si le relTicks d'un des XF Timeouts présents dans la liste est inférieur ou égal à 0.
- getTickInterval() : retourne l'intervalle des ticks de notre timer.

3.4.1 Algorithme ScheduleTimeout

Nous allons voir maintenant un petit algorithme pour ajouter le XF Timeout créé à notre liste. De cette façon, il nous suffira dans la méthode tick de ne décrémenter que le premier XT Timeout de la liste.



Pour ajouter un nouveau Timeout dans la liste:
1. On compare les relTick du newTimeout avec les relTicks du timeout1: newTimeout.relTicks - 1.relTicks
==> si le résultat est négatif on place notre newTimer à l'emplacement 1
==>si le résultat est positif on ne va pas l'insérer la et on fait newTimeout.relticks-=1.relitcks.
2. On reproduit le processus jusqu'à avoir un réultat positif et ainsi pouvoir insérer notre timer.
3. Une fois le nouveau timer inséré, il faut update les relTicks du timer suivant, ici en l'occurrence le timer 3. Il faut lui soustraire les relticks restant du nouveau timer.

Figure 6 ScheduleTimeout method algorithm

3.5 DISPATCHER

La classe XF Dispatcher est également une classe singleton, comme la classe XF TimeoutManager vu précédemment.

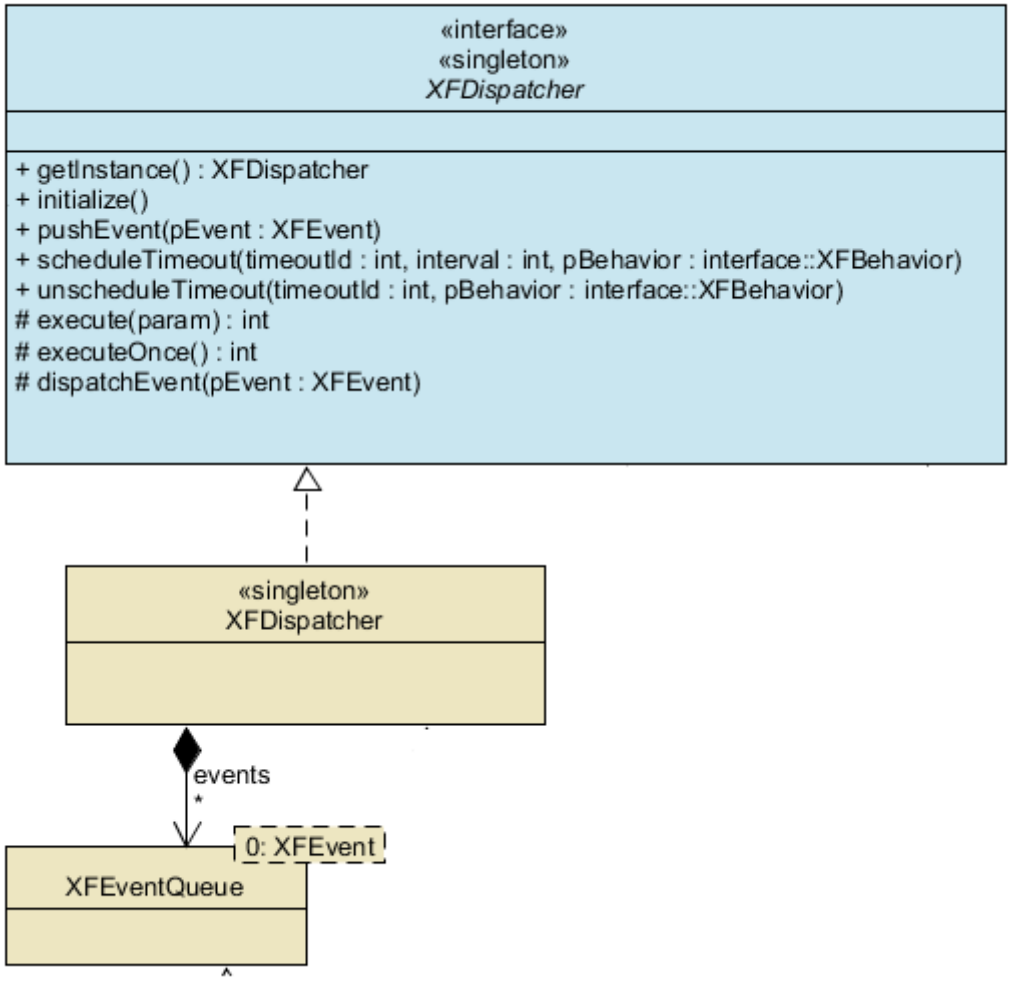


Figure 7 XF Dispatcher class Diagram

Cette classe possède une queue d'événements (voir figure 2). Elle va push des événements dans cette queue, puis les pops afin de les distribuer à la machine d'état qui correspond à l'événements pop. La machine d'état va ensuite process cet événement.

3.6 BEHAVIOUR

Chaque objet de cette classe est une machine d'état de notre programme. Chaque objet peut donc communiquer avec l'objet de la classe XF dispatcher (singleton) grâce au pointeur pDispatcher. Ceci va nous permettre de retourner au dispatcher le statut de l'événement qui vient d'être consommé.

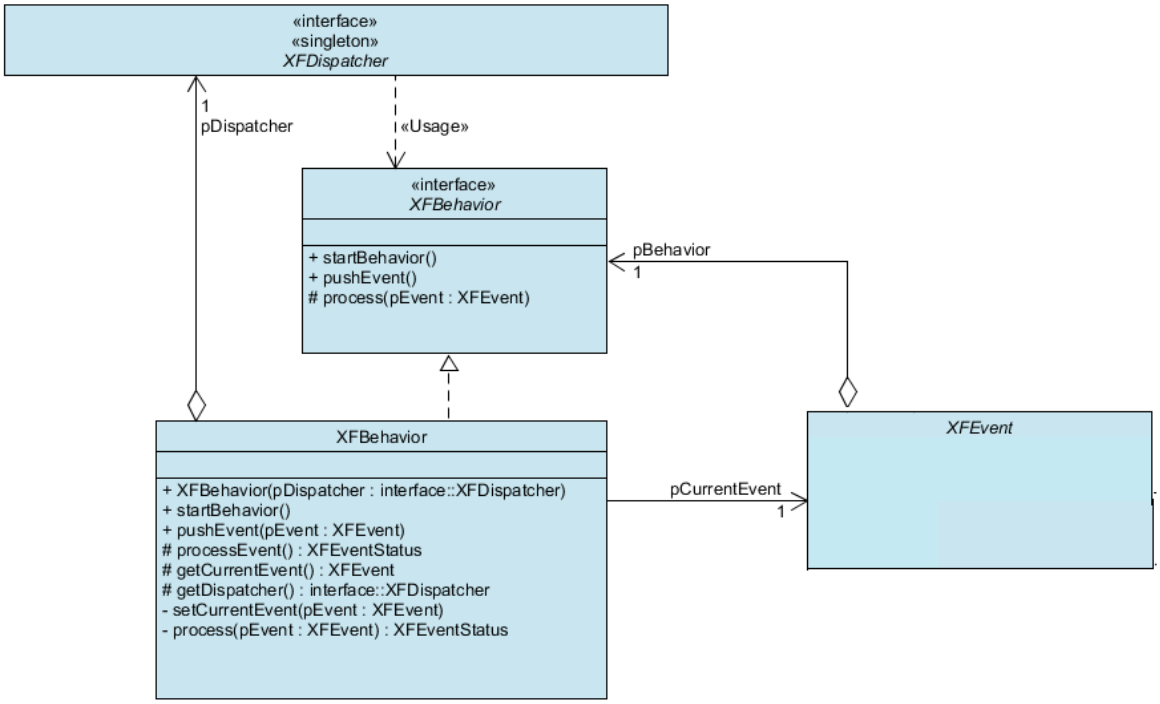


Figure 8 XF Behaviour class Diagram

Sur la figure ci-dessus, nous observons également que la machine d'état possède un attribut pCurrentEvent qui va lui permettre de savoir quel est l'événement qu'elle doit traiter.

Nous voyons également que chaque objet de la classe XF Event a un attribut pointeur sur la machine d'état Interface Xf Behaviour, celui-ci va lui permettre de savoir à quelle machine d'état l'objet appartient.

3.7 XF

Cette classe va permettre de démarrer notre tâche XF, elle va ainsi initialiser les classes XF Dispatcher et XF timeoumanager.

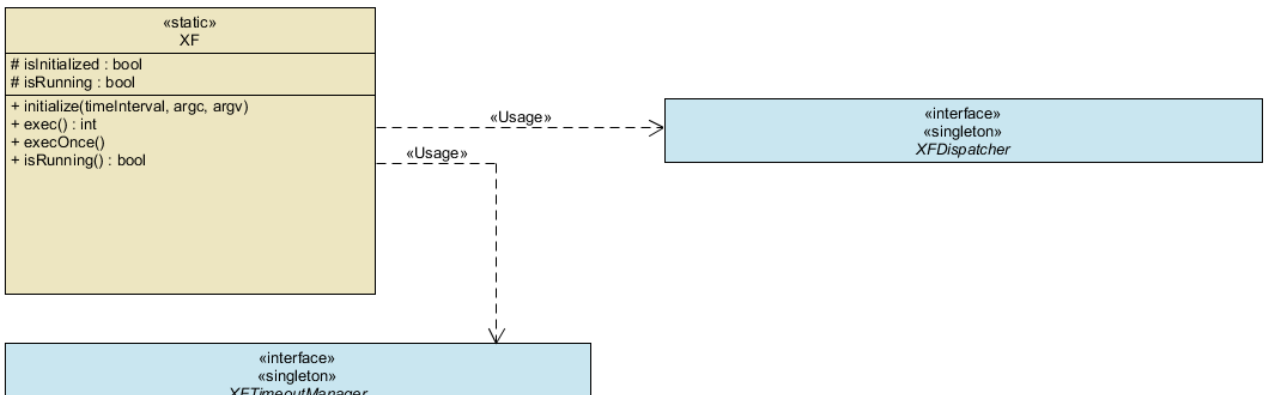


Figure 9 XF class Diagram

4 RÉSULTATS ET TESTS

4.1 DESCRIPTION

Maintenant que nous avons vu comment fonctionne notre XF, il va falloir l'implémenter puis le tester. Pour ce faire nous allons effectuer 5 tests afin de vérifier le bon fonctionnement de notre système.

Tous les tests ont été effectués sur les deux plateformes QT et Stm 32.

4.2 TEST 1

Ce test a été, pour ma part, le plus long à réaliser, car nous devons implémenter toutes les classes de bases (Dispatcher, TimemoutManager, behaviour).

Dans ce test, nous avons deux machines d'état :

- La première va écrire un message Hello chaque 1s.
- La deuxième va écrire un message Echo chaque 500 ms.

```
Starting test1...
---------------------
15:07:28.747 Say Hello
15:07:28.747 Echo
15:07:29.248 Echo
15:07:29.748 Say Hello
15:07:29.748 Echo
15:07:30.249 Echo
15:07:30.748 Say Hello
15:07:30.748 Echo
15:07:31.248 Echo
15:07:31.749 Say Hello
15:07:31.749 Echo
15:07:32.249 Echo
```

Figure 10 Test 1 QTCreator

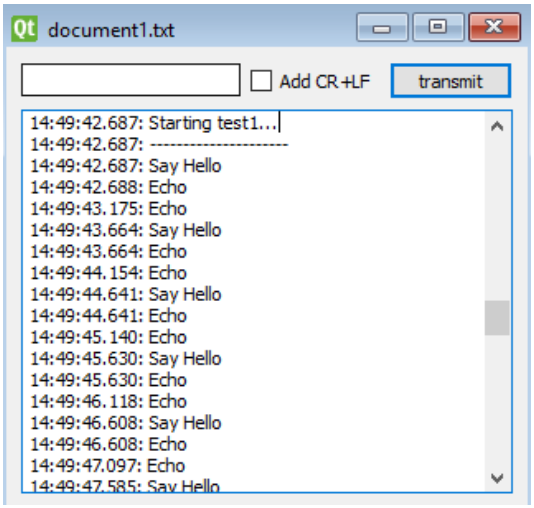


Figure 11 Test 1 STM32

Nous constatons sur les deux figures ci-dessus que le temps sur QT est parfait mais que celui sur STM32 n'est pas correct. Ceci est dû au logiciel tracelog qui va prendre du temps pour écrire les messages reçus. Ce problème va donc se répercuter sur tous les tests.

4.3 TEST 2

Dans ce test, nous avons deux machines d'états : une dynamique et une autre statique. Le but de ce test est de détruire la machine d'état statique à la fin du programme.

```
Called constructor of StateMachine02 object '1' (obj01)
Called constructor of StateMachine02 object '2' (obj02)
Starting test2...
---------------------
obj01: counter 5
obj02: counter 5
obj01: counter 4
obj02: counter 4
obj01: counter 3
obj02: counter 3
obj01: counter 2
obj02: counter 2
obj01: counter 1
obj02: counter 1
obj01: Terminating State Machine
obj02: Terminating State Machine
obj02: Called destructor
```

Figure 12 Test 2 QTCreator

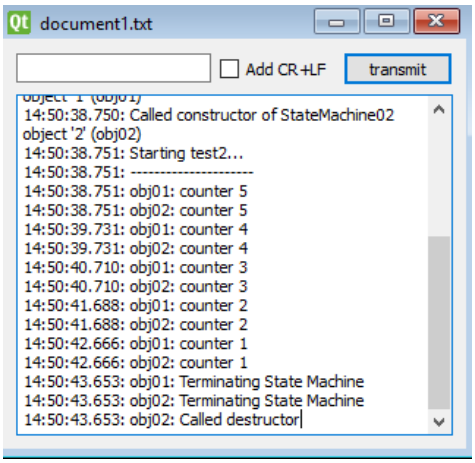


Figure 13 Test 2 STM32

4.4 TEST 3

Une machine d'état va s'envoyer un événement «evRestart » afin de pouvoir changer d'état (redémarrer la machine d'état).

```
Starting test3...
---------------------
Wait
Wait restart
Wait
Wait restart
Wait
Wait restart
Wait
Wait restart
Wait
```

Figure 15 Test 3 QTCreator

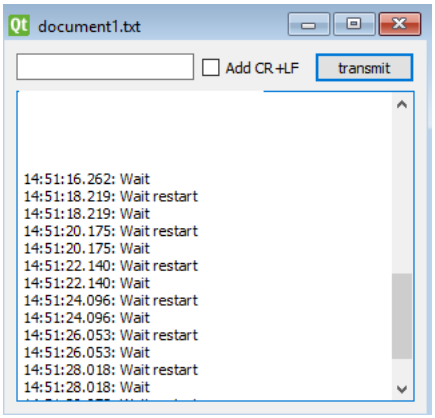


Figure 14 Test 3 STM 32

4.5 TEST 4

Dans ce test il a fallu implémenter la méthode unscheduleTimeout() afin de pouvoir supprimer des timeouts de la liste des XF Timeout.

```
Starting test4...
---------------------
SM04a: Wait
SM04b: Timeout 1
SM04b: Timeout 2
SM04b: Timeout 3
SM04b: Timeout 4
SM04a: Send restart to SM04b
SM04a: Wait
SM04b: Timeout 1
SM04b: Timeout 2
SM04b: Timeout 3
SM04b: Timeout 4
SM04a: Send restart to SM04b
SM04a: Wait
SM04b: Timeout 1
SM04b: Timeout 2
SM04b: Timeout 3
SM04b: Timeout 4
SM04a: Send restart to SM04b
SM04a: Wait
```

Figure 17 Test 4 QTCreator

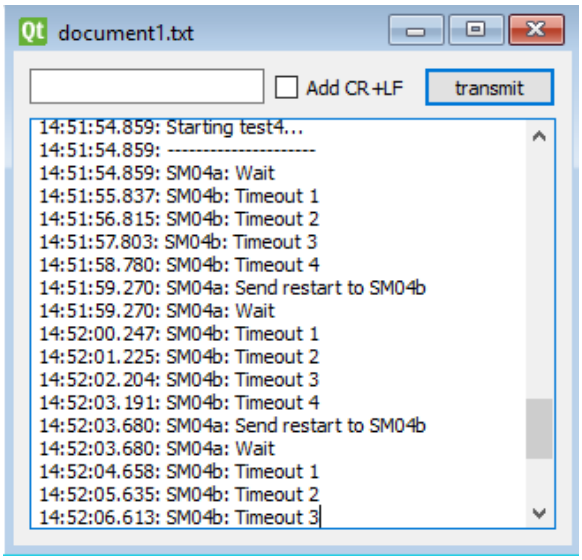


Figure 16 test 4 STM32

4.6 TEST 5

Ce test va permettre de vérifier si nos événements timer sont traités correctement et dans le bon ordre. Ce test va générer plusieurs événements en même temps. Le but est de voir si ces événements sont ajoutés dans le bon ordre dans la liste.

```
Starting test5...
---------------------
15:10:54.070 Tick 500ms
15:10:54.071  One
15:10:54.071   Two
15:10:54.071    Three
15:10:54.572 Tick 500ms
15:10:55.072  One
15:10:55.072   Two
15:10:55.072    Three
15:10:55.072 Tick 500ms
15:10:55.572 Tick 500ms
15:10:56.072  One
15:10:56.072   Two
15:10:56.072    Three
15:10:56.072 Tick 500ms
15:10:56.572 Tick 500ms
15:10:57.072  One
15:10:57.072   Two
15:10:57.072    Three
15:10:57.072 Tick 500ms
15:10:57.572 Tick 500ms
```

Figure 19 Test 5 QTCreator

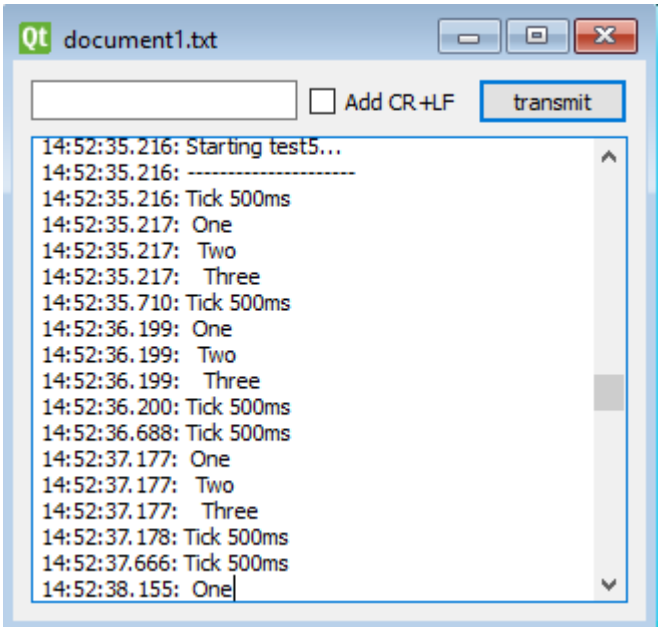


Figure 18 Test 5 STM32

5 CONCLUSION

5.1 BILAN

Tous les tests sont fonctionnels sur les deux plateformes (QTCreator et STM32), ce qui nous montre que le XF a bien été implémenté et est portable d'une plateforme à une autre.

5.2 CONCLUSION

Pour ma part, ce projet s'est bien passé. Il a été très enrichissant et a permis de bien mettre en pratique la théorie vue concernant le XF.

Il m'a été compliqué au début de bien comprendre le diagramme de classe. Une fois cette étape passé tout s'est dérouler de manière assez fluide. Une fois le premier test réussi avec l'algorithme de tri optimisé, les modifications à effectuer pour réussir les autres tests étaient relativement simples et ne m'ont pris que peu de temps.

Nous avons d'abord programmé le XF sur la plateforme QT avant de passer sur la plateforme STM32, ceci m'a demandé un peu d'adaptation car comme expliqué en introduction, la partie port du XF était à réimplémenter car nous changions de plateforme.

En ce qui concerne l'imprécision du temps sur les affichages tracelog du STM32, j'ai tout d'abord cru qu'il s'agissait d'une erreur de programmation de ma part. Après en avoir parlé avec mes camarades et les professeurs, nous avons remarqué que le problème était nouveau et relativement répandu dans la classe, ce qui nous a permis de déduire que ceci est dû au temps d'écriture des messages par tracelog.

5.3 SIGNATURE

Réchy, le 03.11.2021

Sébastien Métral

6 ANNEXES

6.1 COMPONENT CLASS DIAGRAM

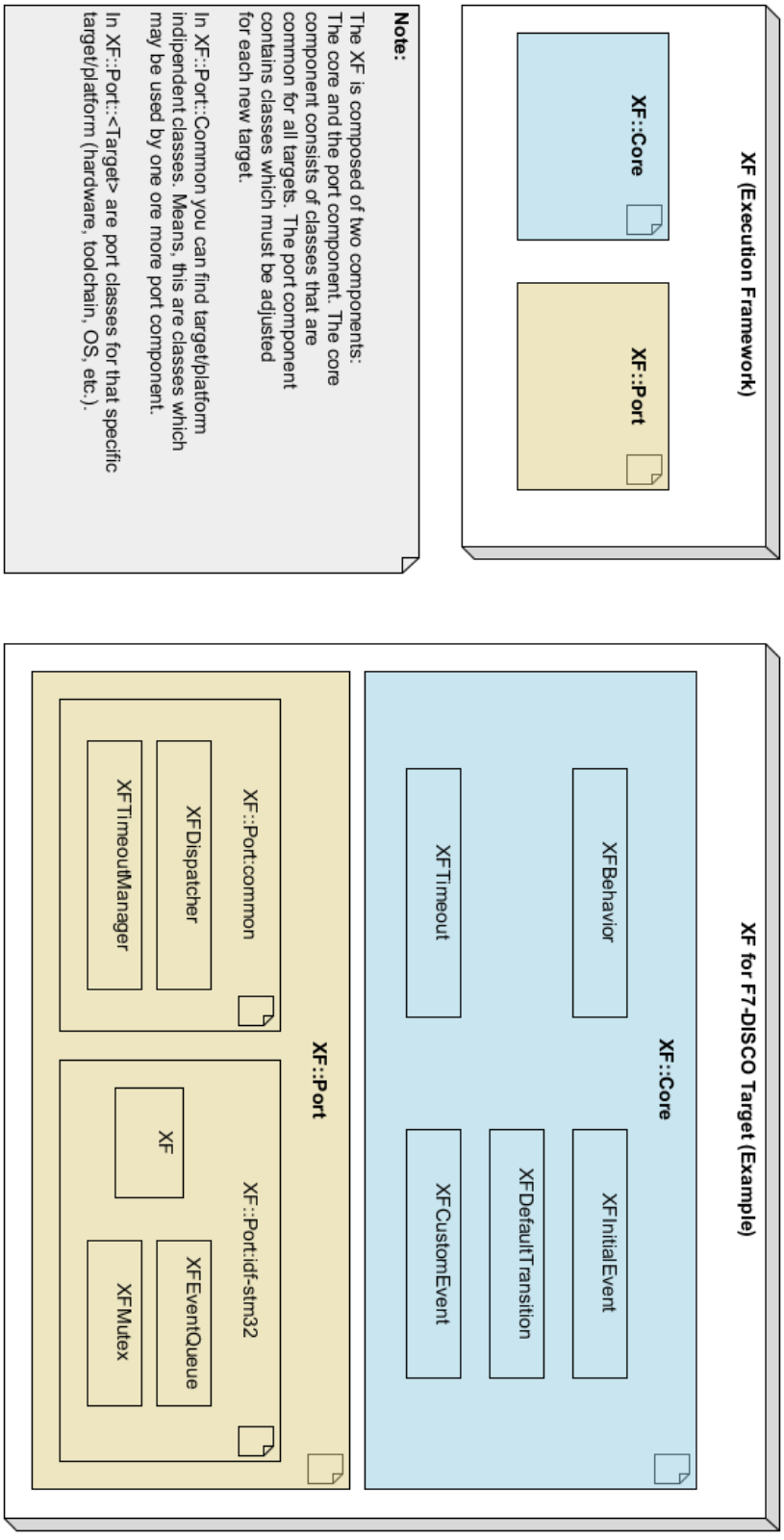


Figure 20 Annexe 1 Component Class Diagram

6.2 XF Class Diagram

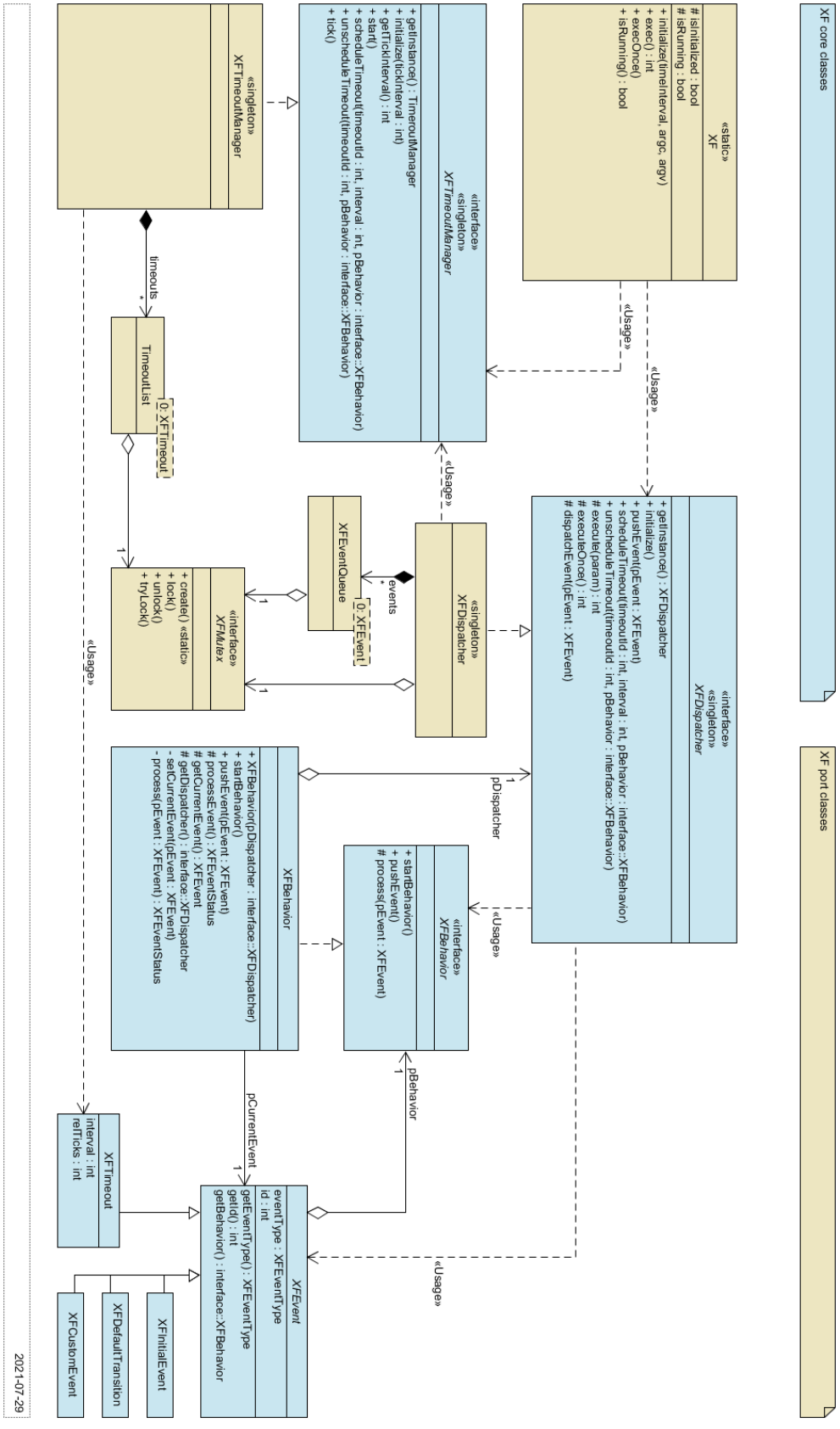


Figure 21 Annexe 2 XF Class Diagram

6.3 SCHEDULETIMEOUT METHOD ALGORITHM

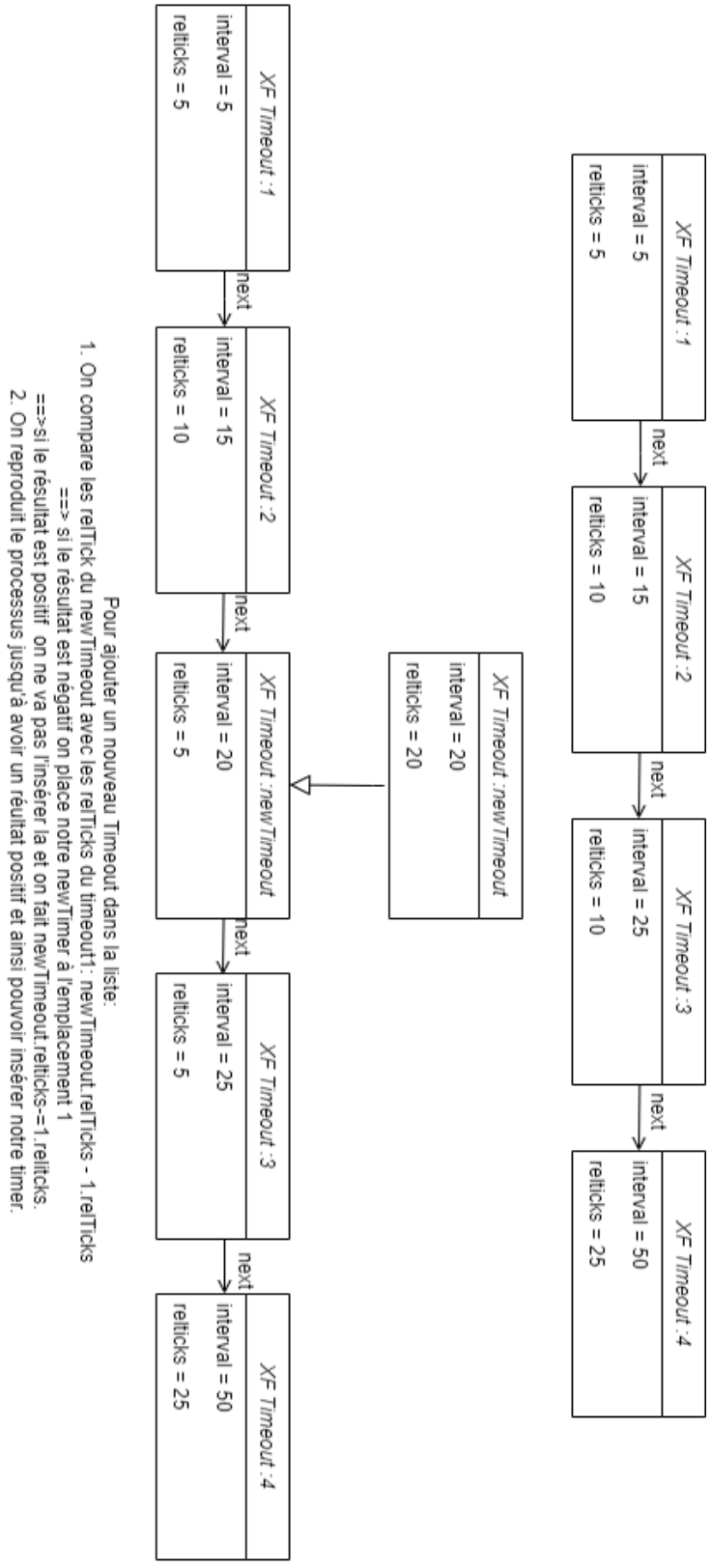


Figure 22 Annexe 3 Algorithme ScheduleTimeout()