HAUTE ÉCOLE D'INGÉNIEUR DU VALAIS

SUPERLAB - XF

REAL TIME PROGRAMMING

Samy Francelet
3 novembre 2021

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Ce laboratoire a servi d'introduction à la programmation temps réel sur système embarqué (ici STM32). Le but était d'implémenter un Execution Framework (XF), en architecture IDF (sans couche RTOS) pour simplifier le développement. Notre XF se doit d'être portable, avec une implémentation sur PC avec la librairie Qt, et une autre sur système embarqué STM32. Des tests communs aux deux systèmes sont fournis, ainsi qu'une structure de base du projet.

1.2 XF

Le XF (execution framework) est un framework servant à piloter plusieurs machines d'états finies en pseudo-parallèle. Pour cela, il utilise des événements qui sont générés par les machines d'états, qui passent par un Dispatcher avant d'être redistribué.

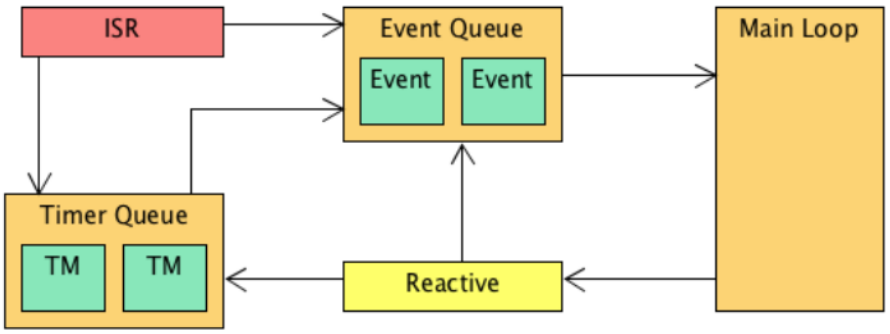


FIGURE 1.1 – Schéma de principe de notre XF en IDF (provenant du *Cours XF* [1])

Dans notre cas, le XF fonctionnera directement sur le hardware, et tournera grâce aux interruptions systèmes (architecture IDF). Ce cas est opposé d'un XF tournant par-dessus un Real Time OS (architecture OXF). L'implémentation est plus simple, mais l'efficacité à grande échelle est réduit.

1. Médard RIEDER. *Script XF 2021*

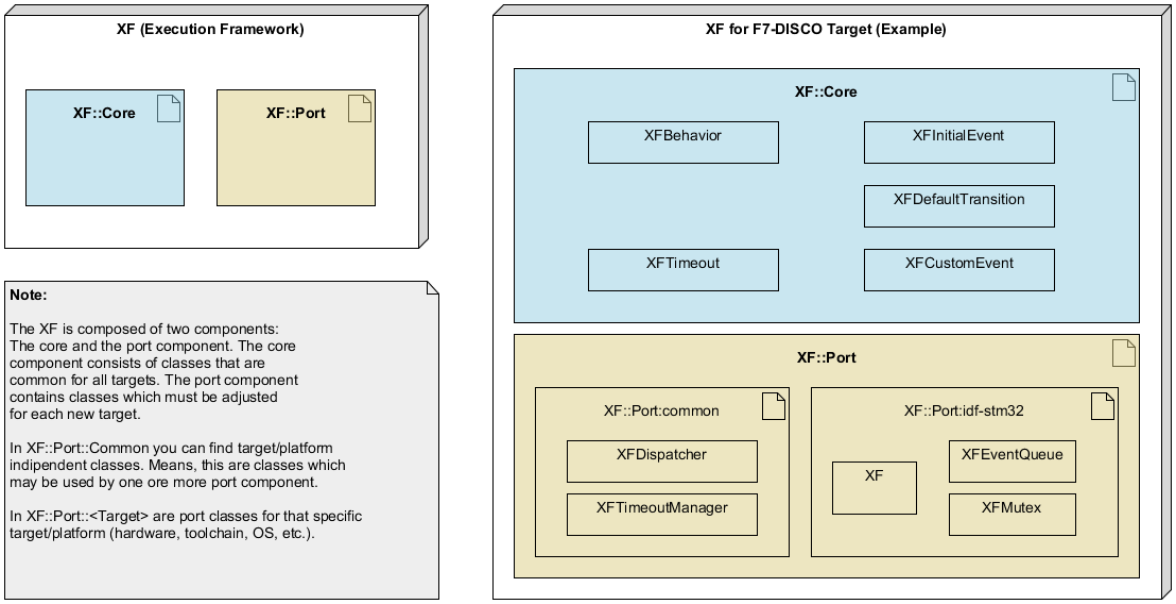


FIGURE 1.2 – Diagramme de composant de classe du XF (provenant de la documentation *Simplified XF*[2])

Notre implémentation du XF se veut portable. Il a donc était séparé en deux parties :

— XF : :Core -> Contient le cœur du XF (machines d'état, événements), cette partie restera inchangée sur tous les systèmes.

— XF : :Port -> Contient les différents ports du XF, certains de ses ports seront spécifiques à chaque système (ex : XFMutex qui bloque les interruptions sur système embarqué, et qui est un Mutex classique sur PC)

Le cœur et les ports forment ensemble un XF complet. La majeur partie du XF : :Core est déjà implémentée. Les événements restent à préciser.

2. Médard RIEDER. et Thomas STERREN. « Simplified XF 1.0.0 ». In : (). URL : `https://cyberlearn.hes-so.ch/course/view.php?id=19260`. (accessed : october 2021)

Chapitre 2

Conception

2.1 XF Core

2.1.1 Événements

La classe XFEvent est composée d'un id d'événement, et d'un type d'événement :

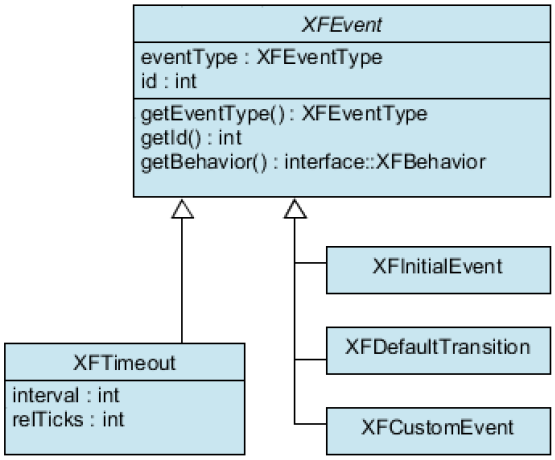


FIGURE 2.1 – Diagramme de classe du XFEvent (provenant du diagramme de classe XF de la documentation *Simplified XF* [1])

Les classes XFInitialEvent, XFDefaultTransition sont simplement des événements avec un type fixe. La classe XFCustomEvent est un alias de XFEvent. Finalement, la classe XFTimeout ajoute deux choses : une intervalle et un nombre de tick restant. En effet, le timeout est une sorte de minuterie générant un événement une fois le temps écoulé. Cette gestion des timeout sera détaillée dans la partie *"Timeout Manager"*

1. RIEDER. et STERREN., « Simplified XF 1.0.0 »

2.1.2 Machines d'états

La classe XFBehavior sert de base pour créer des machines d'états. Elle possède un lien vers le Dispatcher et a toujours un événement courant sauvegardé.

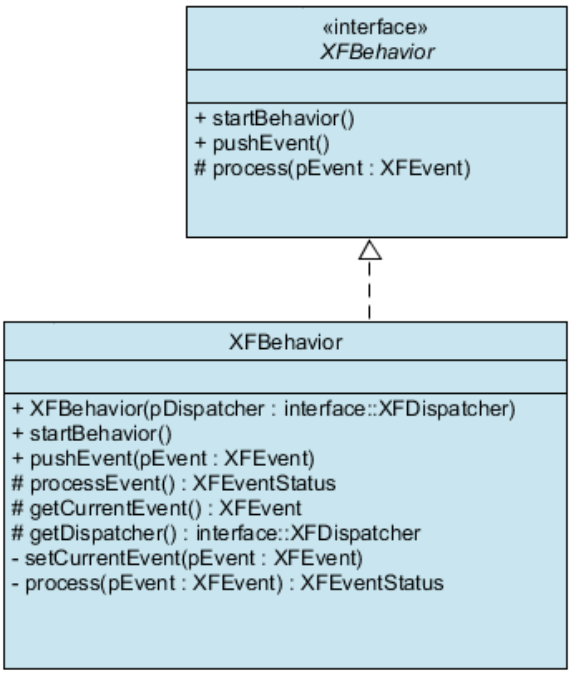


FIGURE 2.2 – Diagramme de classe du XFBehavior (provenant du diagramme de classe XF de la documentation *Simplified XF* [1])

L'utilisation d'une interface est utile pour implémenter des machines d'états personnalisées en héritant de XFBehavior. L'interface permet un polymorphisme flexible et optimal.

1. RIEDER. et STERREN., « Simplified XF 1.0.0 »

2.1.3 Interfaces

Le cœur n'a pas de lien direct sur les ports du XF, car ceci peuvent être modifié d'un système à l'autre. Pour faire en sorte que le cœur n'est jamais besoin de modification, il accède aux ports via des interfaces.

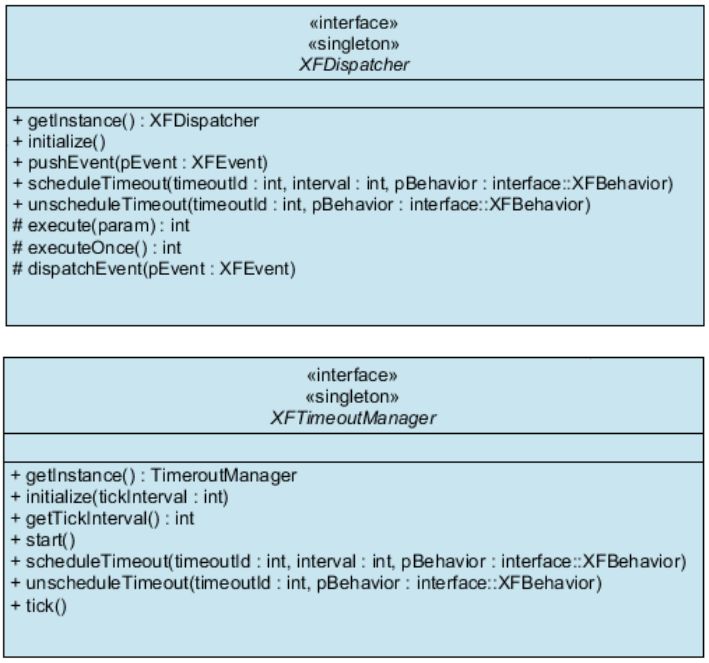


FIGURE 2.3 – Diagramme de classe des interfaces des ports(provenant du diagramme de classe XF de la documentation *Simplified XF* [1])

Ces interfaces sont celles du XFDispatcher et du XFTimeoutManager. Aucun autre port n'est accessible depuis le cœur.

1. RIEDER. et STERREN., « Simplified XF 1.0.0 »

2.2 XF Port

Tous les ports du XF se présente ainsi :

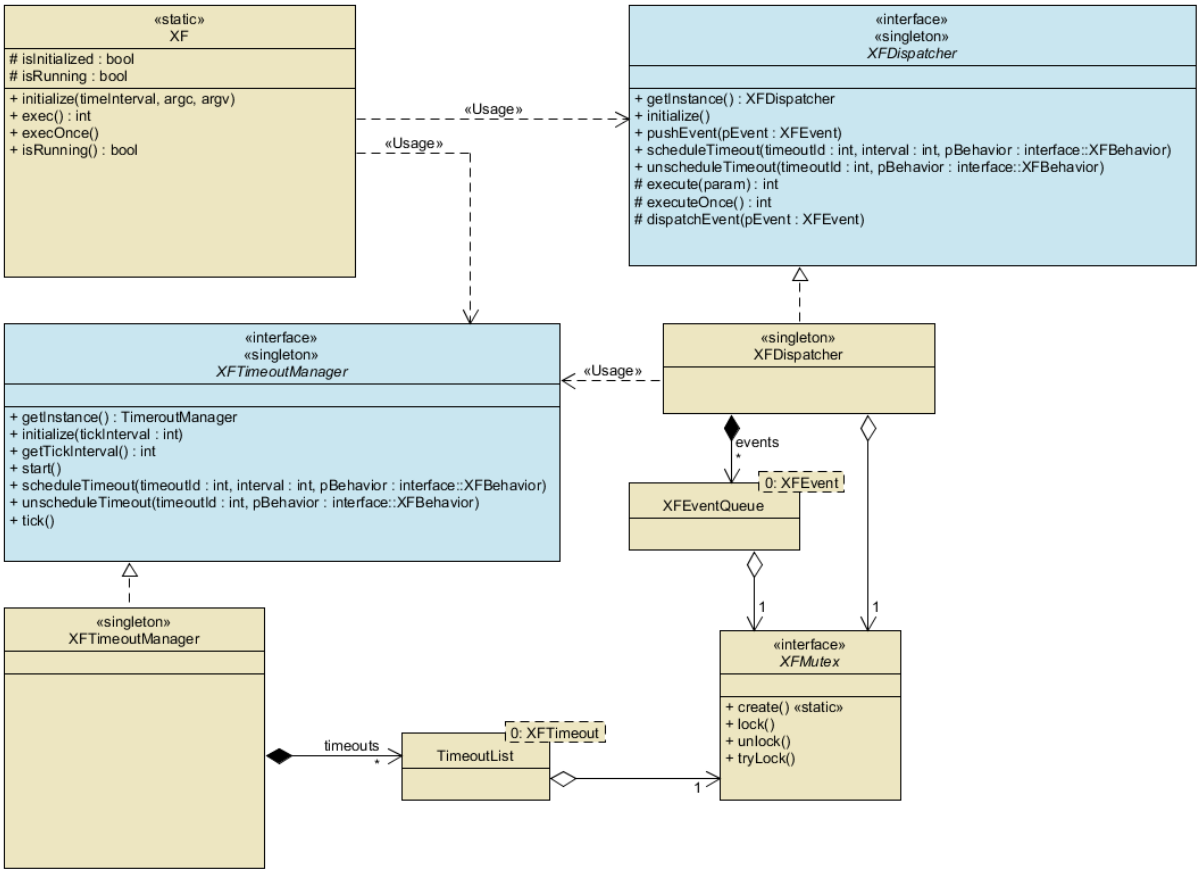


FIGURE 2.4 – Diagramme de classe des ports (provenant du digramme de classe XF de la documentation *Simplified XF* [1])

2.2.1 XF

Cette classe présente simplement la tâche "XF", elle indique si le XF est en cours d'utilisation et possède deux méthodes :

— exec() -> Tourne dans une boucle infinie appelant le Dispatcher
— execOnce() -> Plus adaptée aux systèmes embarqués, doit être appelée dans la boucle infinie du programme

2.2.2 Dispatcher

Le Dispatcher s'occupe d'une queue d'événements. Il peut recevoir des événements de 2 endroits : Le timeout manager, qui renvoie les timeouts échus, et des machines d'états directement. Son rôle est d'ensuite dispatché les événements dès que possible, dans l'ordre : premier arrivé, premier servi (comme une file d'attente).

1. RIEDER. et STERREN., « Simplified XF 1.0.0 »

2.2.3 Timeout Manager

Le Timeout Manager utilise un timer du système (QTimer sur PC, timer d'interruption sur système embarqué) pour décrémenter tous nos Timeouts (les fameuses minuteries). Une implémentation bête et simple serait d'avoir une table de tous les timeouts et de décrémenter les ticks restants à chaque "tick" de notre timer. Cependant, une telle implémentation, bien que très simple, pause un gros problèmes lorsqu'il y aura énormément de timeout. En effet, le fait de décrémenter les ticks de chaque timeout, un à un prends rapidement du temps, ce qui ralenti l'exécution et casse le principe temps réel de ce XF.
La solution plus intelligente serait de placer les timeouts dans l'ordre, et de ne décrémenter que le premier timeout :

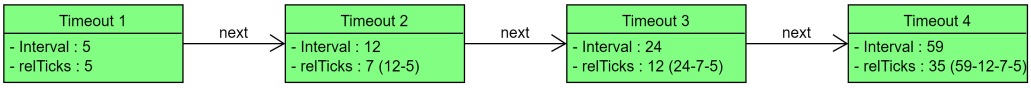


FIGURE 2.5 – Exemple de queue de timeout triée

Ainsi, quand le premier timeout sera échu, il restera 7 ticks au second, et ainsi de suite. Il faut donc un algorithme de placement intelligent :

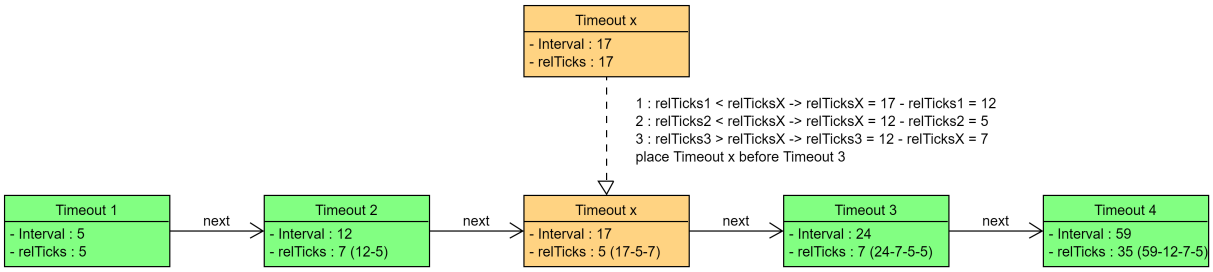


FIGURE 2.6 – Plaçage intelligent de nouveau timeout

L'algorithme procède ainsi : il compare le nouveau timeout au premier de la liste, si le ticks restants du nouveau sont plus petits que ceux du premier, on le place ici et on retire des ticks restants au premier. Sinon, on passe au prochain timeout de la liste, on retire les ticks restants du premier sur notre nouveau timeout et on continue l'opération jusqu'à la fin. Si le timeout n'a pas trouvé sa place au milieu des autres, on le place simplement à la fin, et ses ticks restants seront déjà corrects.
À chaque fois qu'un timeout est échu, il n'est pas supprimé, mais directement transmis au Dispatcher. Étant donné que timeout hérite de XFEvent, il peut donc être gérer comme un événement classique.

2.2.4 Mutex

Le Mutex est un mécanisme, dit d'Exclusion Mutuelle (MUTual EXclusion), qui permet de ne donner l'accès à un objet qu'à un seul thread à la fois.

Imaginons une salle avec beaucoup de monde qui argumente en même temps, personne ne se comprends. Maintenant, ajoutons un modérateur et un canard en plastique donnant le droit de parole. Si vous n'avez pas le canard en plastique, vous n'avez pas le droit de parler, vous pouvez seulement lever la main pour indiquer que vous voulez parler. Une fois que la personne en possession du canard en plastique a fini de parler, elle doit rendre le canard en plastique au modérateur, qui le transmettra au prochain.
On remplace canard en plastique par Mutex, et personne par thread, et le fonctionnement est le même. [1]

1. XETIUS. et IWASROBBED. « What is a mutex ? » In : (). URL : `https://stackoverflow.com/questions/34524/what-is-a-mutex#34558`. (accessed : 3 november 2021).

Chapitre 3

Résultats et tests

3.1 Description des test

5 bancs de test ont été définis pour tester différents comportement du XF. Ces 5 bancs de tests sont disponibles sous Qt, et sous STM32 avec STM32CubeIDE.

3.1.1 Test-bench 1

Le premier test instancie 2 machines d'états générant des timeouts, puis transmettant une phrase à chaque timeout : un premier transmet "Say Hello" chaque seconde, le second : "Echo" chaque 500ms.

3.1.2 Test-bench 2

Le second test instancie deux machines d'état identique, à l'exception qu'une est un objet dynamique (et donc destructible). Les deux objets vont passé un décompte de 5, avant d'être terminé. L'objet dynamique devra être détruit et son destructeur devrait afficher "Called destructor".

3.1.3 Test-bench 3

Ce test check la gestion basique des événement dans les machines d'état. Une machine d'état s'envoie périodiquement à elle même un "evRestart" pour passer d'un état à l'autre.

3.1.4 Test-bench 4

Génère plusieurs timeout, puis crée un restart, qui devra unschedule des timeouts. Si ce test n'est pas réussi, la liste des timeouts devrait se remplir, et on aurait des comportements étranges.

3.1.5 Test-bench 5

Transmets plusieurs timeout en même temps au Timeout Manager pour stress-test l'ajout intelligent de timeouts.

3.2 Discussion des résultats

Tous les tests ont été validés sur les deux systèmes. Le timing du système embarqué est compliqué à mesurer. Nous utilisons le port série pour communiquer les textes au PC, qui va stocker dans un buffer les données et les sortir quand il pourra, donnant des timings étant parfois 20ms en avance, des fois 20ms en retard. Les timings fournis en Annexe B seront donc uniquement les timings mesurer avec Qt.

Chapitre 4

Conclusion

4.1 Bilan

Tous les tests ont été validés, la portabilité fonctionne parfaitement. L'ajout de nouvelles machines d'état est simple. Le XF permet une exécution proche du temps réel grâce aux timeouts.

4.2 Voies d'amélioration

Améliorer l'aspect temps réel en implémentant le XF par-dessus un Real Time OS (Architecture OXF) plutôt que dans notre architecture IDF. Cela permettrait d'avoir plusieurs processus, avec plusieurs Dispatcher, et ainsi de permettre a des tâches complexes de s'exécuter en parallèle.

4.3 Conclusion personnelle

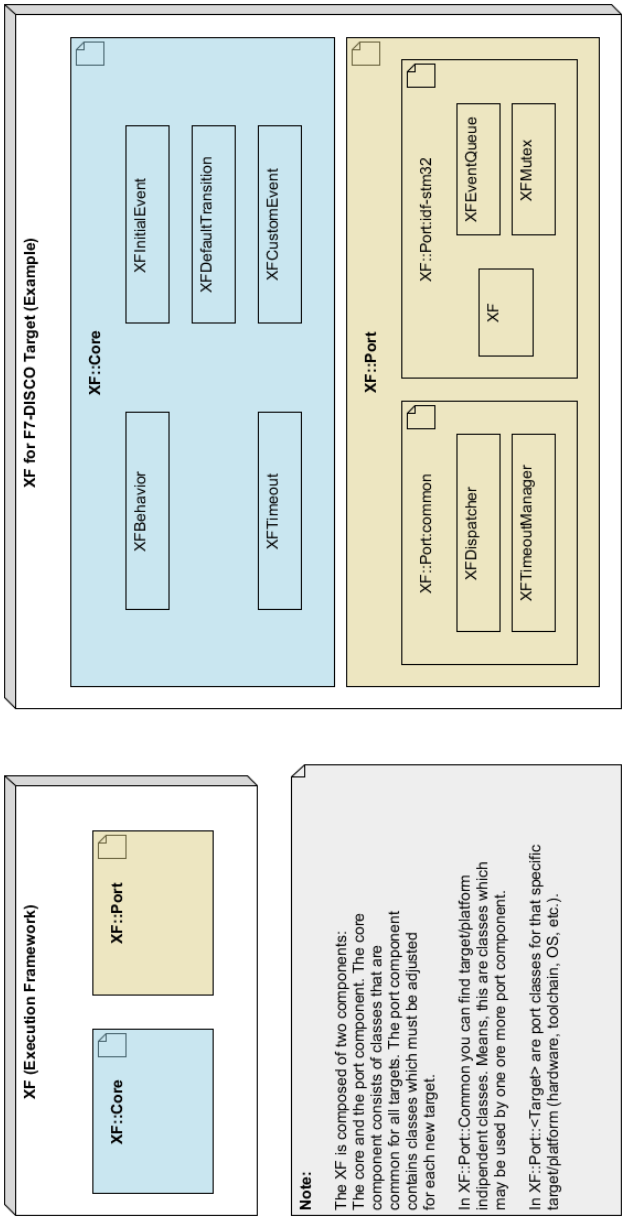
Le fait de devoir se lancer dans un projet de développement avec une structure prédéfinie et bien fournie à rajouter une difficulté au démarrage du projet. Cependant, cela m'a permis de m'habituer à arriver dans un projet en cours, lire toute la documentation et réfléchir avant de commencer à développer tête basse.

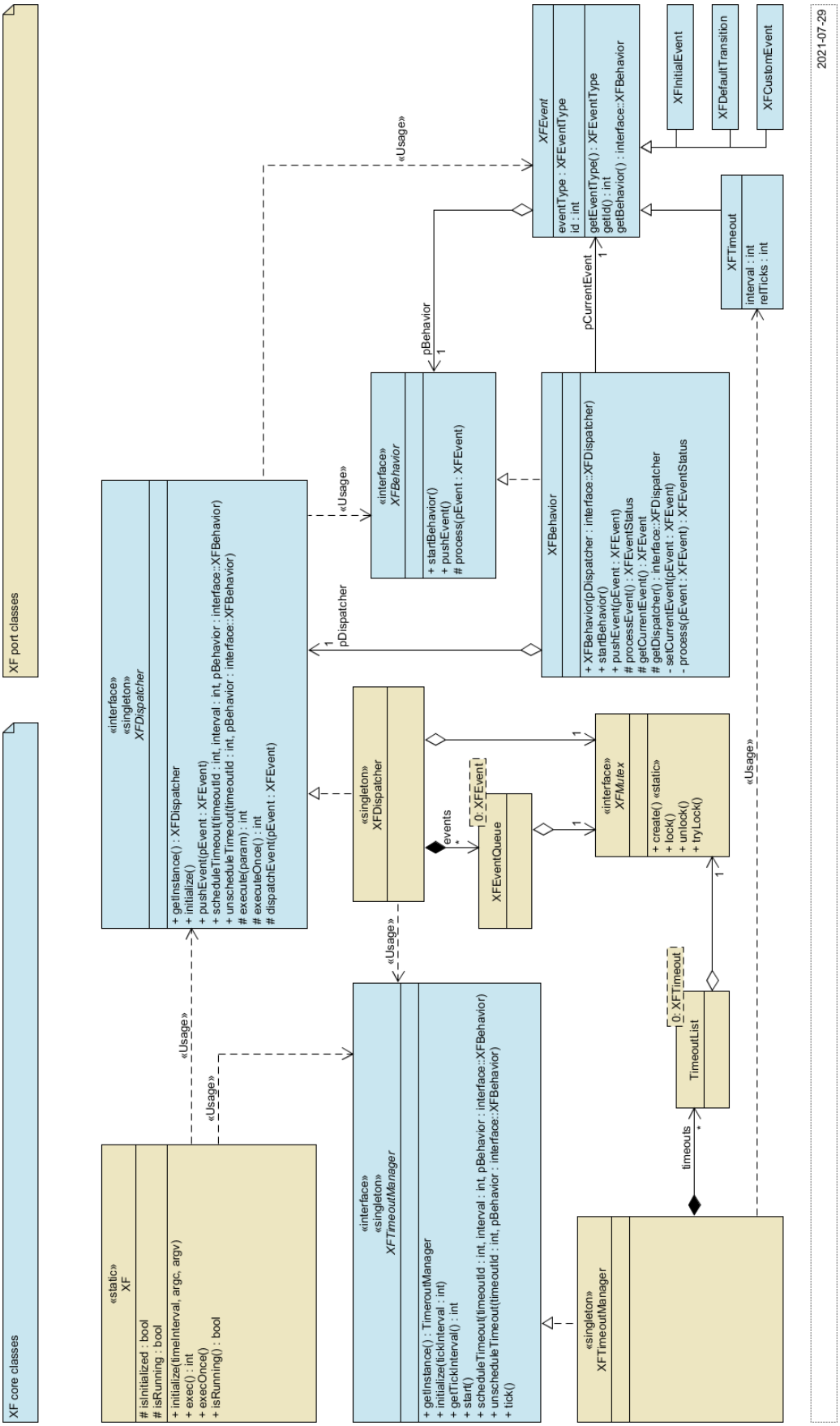
4.4 Signature

Samy Francelet

Annexe A

Diagrammes de classe




XF port classes
XF core classes
2021-07-29
«static»
XF
isInitialized : bool
isRunning : bool
+ initialize(timeInterval, argc, argv)
+ exec() : int
+ execOnce()
+ isRunning() : bool
«interface»
«singleton»
XFDispatcher
+ getInstance() : XFDispatcher
+ initialize()
+ pushEvent(pEvent : XFEvent)
+ scheduleTimeout(timeoutId : int, interval : int, pBehavior : interface::XFBehavior)
+ unscheduleTimeout(timeoutId : int, pBehavior : interface::XFBehavior)
# execute(param) : int
# executeOnce() : int
# dispatchEvent(pEvent : XFEvent)
«singleton»
XFDispatcher
events
*
0: XFEvent
XFEventQueue
1
«interface»
XFMutex
+ create() «static»
+ lock()
+ unlock()
+ tryLock()
«interface»
XFBehavior
+ startBehavior()
+ pushEvent()
# process(pEvent : XFEvent)
XFBehavior
+ XFBehavior(pDispatcher : interface::XFDispatcher)
+ startBehavior()
+ pushEvent(pEvent : XFEvent)
# processEvent() : XFEventStatus
# getCurrentEvent() : XFEvent
# getDispatcher() : interface::XFDispatcher
- setCurrentEvent(pEvent : XFEvent)
- process(pEvent : XFEvent) : XFEventStatus
XFEvent
eventType : XFEventType
id : int
getEventType() : XFEventType
getId() : int
getBehavior() : interface::XFBehavior
XFInitialEvent
XFDefaultTransition
XFCustomEvent
XFTimeout
interval : int
relTicks : int
«interface»
«singleton»
XFTimeoutManager
+ getInstance() : TimeoutManager
+ initialize(tickInterval : int)
+ getTickInterval() : int
+ start()
+ scheduleTimeout(timeoutId : int, interval : int, pBehavior : interface::XFBehavior)
+ unscheduleTimeout(timeoutId : int, pBehavior : interface::XFBehavior)
+ tick()
0: XFTimeout
TimeoutList
timeouts
«singleton»
XFTimeoutManager
«Usage»
pDispatcher
pBehavior
pCurrentEvent
1

Annexe B

Tests

B.1 Test 1

```
Starting test1...
--------------------
10:40:06.987 : Say Hello
10:40:06.989 : Echo
10:40:07.489 : Echo
10:40:07.989 : Say Hello
10:40:08.010 : Echo
10:40:08.510 : Echo
10:40:09.008 : Say Hello
10:40:09.012 : Echo
10:40:09.510 : Echo
10:40:10.010 : Say Hello
10:40:10.016 : Echo
10:40:10.509 : Echo
10:40:11.012 : Say Hello
10:40:11.016 : Echo
10:40:11.509 : Echo
10:40:12.009 : Say Hello
10:40:12.011 : Echo
10:40:12.508 : Echo
10:40:13.009 : Say Hello
10:40:13.011 : Echo
10:40:13.509 : Echo
10:40:14.009 : Say Hello
10:40:14.010 : Echo
10:40:14.508 : Echo
10:40:15.008 : Say Hello
10:40:15.010 : Echo
10:40:15.509 : Echo
10:40:16.009 : Say Hello
10:40:16.017 : Echo
10:40:16.508 : Echo
10:40:17.009 : Say Hello
10:40:17.011 : Echo
10:40:17.509 : Echo
```

B.2 Test 2

```
10:55:22.962 :  Called constructor of StateMachine02 object '1' (obj01)
10:55:22.971 :  Called constructor of StateMachine02 object '2' (obj02)
10:55:22.979 :  Starting test2...
10:55:22.981 :  ---------------------
10:55:22.983 :  obj01: counter 5
10:55:22.989 :  obj02: counter 5
10:55:23.987 :  obj01: counter 4
10:55:23.989 :  obj02: counter 4
10:55:24.986 :  obj01: counter 3
10:55:24.991 :  obj02: counter 3
10:55:25.986 :  obj01: counter 2
10:55:25.988 :  obj02: counter 2
10:55:26.987 :  obj01: counter 1
10:55:26.989 :  obj02: counter 1
10:55:27.987 :  obj01: Terminating State Machine
10:55:27.990 :  obj02: Terminating State Machine
10:55:27.993 :  obj02: Called destructor
```

B.3 Test 3

```
10:55:57.171 :  Starting test3...
10:55:57.175 :  ---------------------
10:55:57.177 :  Wait
10:55:59.179 :  Wait restart
10:55:59.179 :  Wait
10:56:01.180 :  Wait restart
10:56:01.180 :  Wait
10:56:03.180 :  Wait restart
10:56:03.180 :  Wait
10:56:05.179 :  Wait restart
10:56:05.179 :  Wait
10:56:07.180 :  Wait restart
10:56:07.180 :  Wait
10:56:09.180 :  Wait restart
10:56:09.180 :  Wait
10:56:11.180 :  Wait restart
10:56:11.180 :  Wait
10:56:13.180 :  Wait restart
10:56:13.180 :  Wait
10:56:15.180 :  Wait restart
10:56:15.180 :  Wait
10:56:17.179 :  Wait restart
10:56:17.179 :  Wait
10:56:19.180 :  Wait restart
10:56:19.180 :  Wait
10:56:21.180 :  Wait restart
10:56:21.180 :  Wait
10:56:23.180 :  Wait restart
10:56:23.180 :  Wait
10:56:25.179 :  Wait restart
10:56:25.179 :  Wait
10:56:27.180 :  Wait restart
10:56:27.180 :  Wait
10:56:29.179 :  Wait restart
10:56:29.179 :  Wait
```

B.4 Test 4

```
10:57:41.260 :  Starting test4...
10:57:41.265 :  ---------------------
10:57:41.265 :  SM04a: Wait
10:57:42.267 :  SM04b: Timeout 1
10:57:43.267 :  SM04b: Timeout 2
10:57:44.267 :  SM04b: Timeout 3
10:57:45.267 :  SM04b: Timeout 4
10:57:45.767 :  SM04a: Send restart to SM04b
10:57:45.767 :  SM04a: Wait
10:57:46.767 :  SM04b: Timeout 1
10:57:47.766 :  SM04b: Timeout 2
10:57:48.766 :  SM04b: Timeout 3
10:57:49.767 :  SM04b: Timeout 4
10:57:50.267 :  SM04a: Send restart to SM04b
10:57:50.267 :  SM04a: Wait
10:57:51.267 :  SM04b: Timeout 1
10:57:52.266 :  SM04b: Timeout 2
10:57:53.267 :  SM04b: Timeout 3
10:57:54.267 :  SM04b: Timeout 4
10:57:54.767 :  SM04a: Send restart to SM04b
10:57:54.767 :  SM04a: Wait
10:57:55.767 :  SM04b: Timeout 1
10:57:56.767 :  SM04b: Timeout 2
10:57:57.767 :  SM04b: Timeout 3
10:57:58.766 :  SM04b: Timeout 4
10:57:59.267 :  SM04a: Send restart to SM04b
10:57:59.267 :  SM04a: Wait
10:58:00.267 :  SM04b: Timeout 1
10:58:01.267 :  SM04b: Timeout 2
10:58:02.267 :  SM04b: Timeout 3
10:58:03.267 :  SM04b: Timeout 4
10:58:03.766 :  SM04a: Send restart to SM04b
```

B.5 Test 5

```
10:59:35.778 :  Starting test5...
10:59:35.783 :  ---------------------
10:59:35.784 : Tick 500ms
10:59:35.784 :  One
10:59:35.784 :   Two
10:59:35.784 :    Three
10:59:36.286 : Tick 500ms
10:59:36.785 :  One
10:59:36.785 :   Two
10:59:36.785 :    Three
10:59:36.785 : Tick 500ms
10:59:37.286 : Tick 500ms
10:59:37.786 :  One
10:59:37.786 :   Two
10:59:37.786 :    Three
10:59:37.786 : Tick 500ms
10:59:38.286 : Tick 500ms
10:59:38.786 :  One
10:59:38.786 :   Two
10:59:38.786 :    Three
10:59:38.786 : Tick 500ms
10:59:39.286 : Tick 500ms
10:59:39.786 :  One
10:59:39.786 :   Two
10:59:39.786 :    Three
10:59:39.786 : Tick 500ms
10:59:40.286 : Tick 500ms
10:59:40.785 :  One
10:59:40.785 :   Two
10:59:40.785 :    Three
10:59:40.785 : Tick 500ms
10:59:41.285 : Tick 500ms
10:59:41.786 :  One
10:59:41.786 :   Two
10:59:41.786 :    Three
10:59:41.786 : Tick 500ms
```

Bibliographie

RIEDER., Médard. *Script XF 2021*.

RIEDER., Médard et Thomas STERREN. « Simplified XF 1.0.0 ». In : (). URL : https://cyberlearn.hes-so.ch/course/view.php?id=19260. (accessed : october 2021).

XETIUS. et IWASROBBED. « What is a mutex ? » In : (). URL : https://stackoverflow.com/questions/34524/what-is-a-mutex#34558. (accessed : 3 november 2021).