

P4

Sacha COLBERT, Xavier FERBER, Dylan BEROUD, Nicolas MAS

21 Novembre 2023

Contents

1	Contextualisation :	2
2	Problématique :	2
3	Contraintes :	2
3.1	Exemple de code.txt incohérent.	2
3.2	Différentes fonctionnalités à implémenter.	2
4	Plan d'action :	2
4.1	Analyser les ressources.	2
4.2	Déterminer les parties réutilisables du code initial.	2
4.3	Comprendre l'objectif de la simulation et les différentes fonctionnalités à intégrer.	2
4.4	Décomposer logiquement le programme.	2
4.5	Concevoir une simulation cohérente.	2
5	Solution :	3
6	Code :	6
6.1	Envoyer un message :	6
6.2	Recevoir un message :	6
6.3	Gestion des évènements :	7
7	Conclusion :	9

1 Contextualisation :

Nous travaillons sur une simulation de communications multiplexées, le code actuel n'est pas fonctionnel car il emploie une technique qui n'est pas crédible, la station FM devrait connaître tous les postes radio FM qui sont en écoute de sa diffusion pour pouvoir les contacter.

La simulation devra lire les messages depuis le début de la diffusion sans être forcément en écoute dès le début.

L'utilisateur doit également avoir la possibilité de s'abonner à plusieurs canaux. Cette partie de code est déjà fonctionnelle dans le code fourni.

2 Problématique :

Comment recréer la simulation de manière réaliste et optimisée ?

3 Contraintes :

3.1 Exemple de code.txt incohérent.

3.2 Différentes fonctionnalités à implémenter.

4 Plan d'action :

4.1 Analyser les ressources.

4.2 Déterminer les parties réutilisables du code initial.


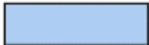






4.3 Comprendre l'objectif de la simulation et les différentes fonctionnalités à intégrer.

4.4 Décomposer logiquement le programme.

4.5 Concevoir une simulation cohérente.

5 Solution :

Légende

Relatif à la classe Récepteur	
Relatif à la classe Émetteur	
Relatif à la fonction Recevoir de Récepteur	
Relatif à la fonction AddEvent de Émetteur	
Relatif au message émit	
Relatif à la liste "events" de fonctions Recevoir de Récepteur stockée dans Émetteur	
Relatif à la fonction Emettre de Émetteur	
Relatif à la fonction AddRecept de Récepteur	

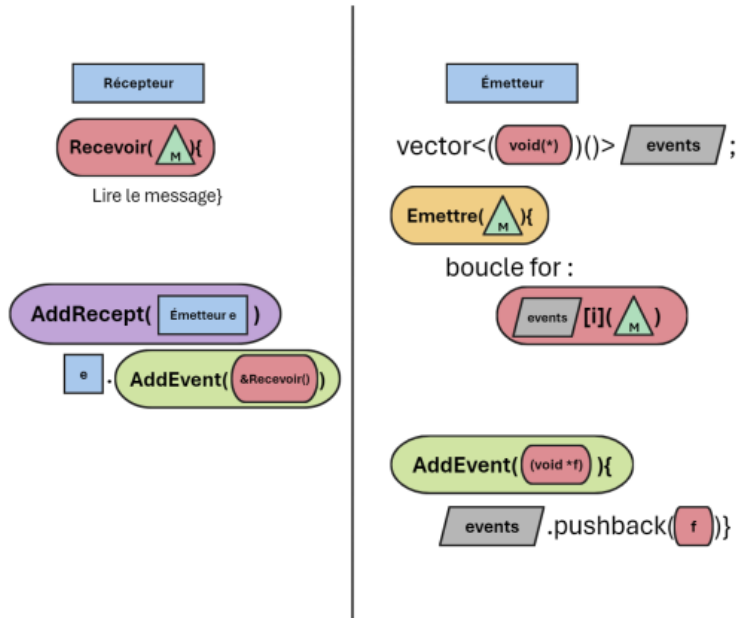
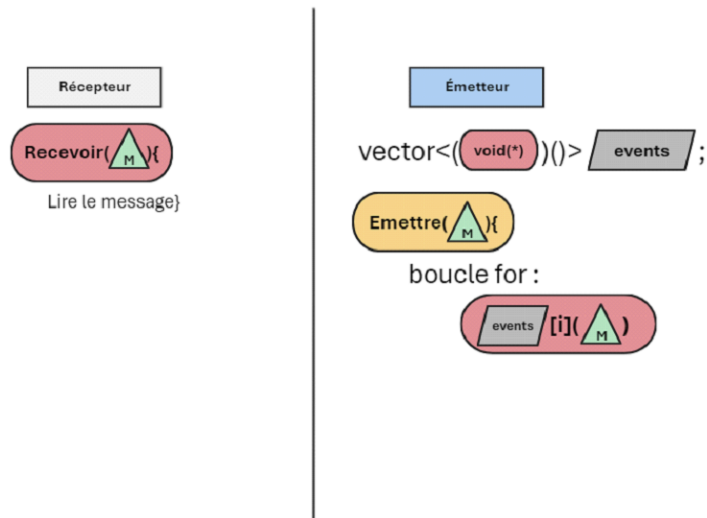


Figure 1: Voici un schéma de la méthode que nous comptons utiliser pour résoudre le problème posé :

Commençons par expliquer cette partie, qui est l'essentiel du fonctionnement du programme :



Lors de la conception, nous avons deux options :
Nous devons faire interagir émetteur et récepteurs. Pour cela, nous pouvons créer une troisième classe stockage, et insérer chaque émission de l'émetteur dans le stockage pour le récupérer par la suite dans les récepteurs.
Nous n'avons pas trouvé cette solution réaliste, et elle minimise les interactions entre les classes. Ainsi, nous avons choisi une approche différente. Nous créons dans émetteurs une liste de fonctions Recevoir de Récepteur, et lorsque l'on décide d'émettre un nouveau message.

6 Code :

6.1 Envoyer un message :

```
1  /// @brief Fonction qui permet d'émettre un message.
2  /// @param m Le message à envoyer.
3  void Emetteur::SendMessage(const str& m) const
4  {
5      // On crypte d'abord le message
6      str newMessage = this->name + " : " + m;
7      newMessage = Cryptage::Encode(newMessage);
8
9      // On appelle les fonctions qui écoutent
10     this->onMessageLaunch.CallEvents(newMessage);
11 }
```

6.2 Recevoir un message :

```
1  /// @brief Fonction appelée lorsque l'on reçoit un message d'une certain station
2  /// @param m Le message reçu.
3  void Recepteur::ReceiveMessage(const str &m)
4  {
5      // On décode le message reçu.
6      str decodedMessage = m;
7      decodedMessage = Cryptage::Decode(decodedMessage);
8      str newMessage = this->name + " a reçu le message suivant :\n" + decodedMessage;
9
10     // On affiche le message récupérer.
11     cout << newMessage << endl;
12
13     // On garde le message en mémoire
14     this->previousMessages.push_back(decodedMessage);
15 }
```

6.3 Gestion des événements :

```
1  /// @brief On s'abonne à l'évènement.  
2  /// @param newCallback La fonction qui sera appelée par l'évènement  
3  void EventHandler::AddEvent(EventCallback* newCallback)  
4  {  
5      // On ajoute la fonction à l'évènement  
6      this->callbacks.push_back(newCallback);  
7  }
```

```
1  /// @brief Fonction qui appelle toutes les fonctions abonnées.  
2  /// @param m Le message à envoyer.  
3  void EventHandler::CallEvents(const str& m) const  
4  {  
5      for (const auto& callback : this->callbacks)  
6      {  
7          (*callback)(m);  
8      }  
9  }
```

```

1  /// @brief On supprime un évènement déjà ajouté.
2  /// @param callback la fonction à supprimer.
3  void EventHandler::DeleteEvent(EventCallback* callback)
4  {
5      try
6      {
7          for(unsigned int i = 0; i < this->callbacks.size(); i++)
8          {
9              // Si on l'a trouvé, on le supprime
10             if(this->callbacks[i]->target<void*>() == callback->target<void*>())
11             {
12                 this->callbacks.erase(this->callbacks.begin() + i);
13                 return;
14             }
15         }
16         throw EventCouldntDelete();
17     }
18     catch(const EventCouldntDelete& e)
19     {
20         e.what();
21     }
22 }
23

```

```

1  #pragma once
2
3  // On importe le système d'évènement
4  #include <functional>
5  #include <vector>
6  #include "String.hpp"
7  typedef std::function<void(const str&)> EventCallback;
8
9  class EventHandler
10 {
11     private:
12         std::vector<EventCallback*> callbacks;
13         EventCallback* ConvertFrom(void*)(const str&) const;
14     public:
15         void AddEvent(EventCallback*);
16         void AddEvent(void*)(const str&);
17         void DeleteEvent(EventCallback*);
18         void DeleteEvent(void*)(const str&);
19
20         void CallEvents(const str&) const;
21         void Clear();
22
23         ~EventHandler();
24 };

```

```

1  /// @brief Cette fonction supprime tous les évènements ajoutées à l'évènement handler.
2  void EventHandler::Clear()
3  {
4      for(unsigned int i = 0; i < this->callbacks.size(); i++)
5      {
6          delete this->callbacks[i];
7      }
8      this->callbacks.clear();
9  }
10
11 /// @brief Destructeur de l'évènement handler.
12 EventHandler::~EventHandler()
13 {
14     this->Clear();
15 }

```


7 Conclusion :

Notre code fonctionne et les fonctionnalités tel que la programmation asynchrone et l'abonnement sont correctement intégrées, à l'avenir nous pourrions améliorer la gestion des événements pour l'utiliser avec des fonctions d'arguments quelconques.

