IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

12/09/2024

BODIN Noé
COLIN Guillaume
DOUANT Antoine
LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 1

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Introduction

L'atelier SIT213 vise à nous aider à comprendre et à simuler les concepts de la transmission numérique. On se concentre sur toutes les étapes d'une chaîne de communication, depuis la création d'un signal à la source jusqu'à sa réception à destination, en passant par les divers éléments comme les transmetteurs et les modulateurs.

Pour cette première étape du projet, notre objectif est de poser les bases de la simulation de la transmission numérique en mettant en place les composants essentiels de la chaîne de communication. On va donc travailler avec :

- La **`SourceFixe`**, qui est responsable de la génération du message à transmettre.
- La **`SourceAléatoire`**, qui génère un message aléatoire.
- Le **`TransmetteurParfait`**, qui transmet l'information sans la modifier.
- La **`DestinationFinale`**, qui reçoit l'information transmise.

Le rapport décrit les actions effectuées pour instancier ces composants, les connecter et mesurer le taux d'erreur binaire (TEB).

1. Conception des composants

1.1. `SourceFixe`

- **But** : La SourceFixe génère un message binaire de type `Boolean[]`, en fonction d'un String (0 ou 1).
- **Fonctionnalité** : La méthode principale `emettre()` transmet le message au composant suivant de la chaîne de communication via la méthode `connecter(DestinationInterface<T>)`.

1.1. `SourceAléatoire`

- **But** : La SourceAléatoire génère un message binaire de type `Boolean[]`, en fonction d'une taille et, si indiqué, d'une graine.
- **Fonctionnalité** : La méthode principale `emettre()` transmet le messageau composant suivant de la chaîne de communication via la méthode `connecter(DestinationInterface<T>)`.

1.2. `TransmetteurParfait`

- **But** : Le TransmetteurParfait prend le message émis par la source et le transmet à la destination sans dégradation.
- **Fonctionnalité** : Le transmetteur reçoit une information via `recevoir(Information<T>)` et la retransmet avec la méthode `emettre()`.

1.3. `DestinationFinale`

- **But** : La DestinationFinale reçoit l'information transmise par le transmetteur et la stocke pour évaluation. Elle est capable de recevoir et de traiter les informations de type `Boolean`.
- **Fonctionnalité** : Elle stocke l'information reçue avec la méthode `recevoir(Information<T>)`, qui peut être consultée via `getInformationRecue()`.

2. Connexion des composants

2.1. Connexion de SourceFixe au TransmetteurParfait

La SourceFixe est connectée au TransmetteurParfait en utilisant la méthode `connecter(DestinationInterface<T>)`. Cela permet à la source de transmettre son message au transmetteur.

2.2. Connexion du TransmetteurParfait à DestinationFinale

Le TransmetteurParfait est ensuite connecté à la DestinationFinale. Cette connexion permet de faire passer l'information reçue par le transmetteur jusqu'à la destination.

3. Simulation de l'émission et de la réception

3.1. Émission depuis SourceFixe

Après la connexion des composants, la méthode `emettre()` de la SourceFixe est appelée. Cette méthode génère un message aléatoire de longueur définie (par exemple 100 bits) et le transmet au TransmetteurParfait.

3.2. Réception de l'Information par DestinationFinale

Le TransmetteurParfait transmet ensuite cette information à la DestinationFinale. La destination utilise la méthode `recevoir(Information<T>)` pour enregistrer le message reçu.

4. Calcul du taux d'erreur binaire (TEB)

Une fois que la source a émis et que la destination a reçu le message, le taux d'erreur binaire (TEB) peut être calculé. Le TEB est calculé en comparant le message émis par la source et le message reçu par la destination.

4.1. Algorithme du Calcul du TEB

Le calcul du TEB est basé sur la proportion de bits erronés dans le message reçu par rapport au message original. Le TEB est donné par la formule suivante :

TEB=Nombre de bits erronés / Nombre total de bits émis

Pour cela, les méthodes `getInformationEmise()` de la source et `getInformationRecue()` de la destination sont appelés et les messages sont comparés bit par bit.

5. Tests

Nous utilisons JUnit (version 4) pour effectuer les tests unitaires.
De la manière suivante :

- Les fichiers .jar sont situés dans le répertoire `lib`
- compilation des tests via le script `compile`
- Exécution des tests via le script `runTests`, utilisant la classe AllTests

Chaque classe et ses méthodes sont testées.

L'objectif principal des tests est de s'assurer que chaque composant du système fonctionne correctement de manière isolée, ainsi que lorsqu'il est intégré avec d'autres composants. Cela inclut la génération de sources aléatoires ou fixes, la transmission des informations via des transmetteurs parfaits, et la réception correcte des informations par les destinations. De plus, le simulateur dans son ensemble est également testé pour vérifier le flux complet de la transmission.

5.1. Organisation des tests

Une suite de tests (`AllTests`) a été configurée pour regrouper et exécuter automatiquement tous les tests des différentes classes. Voici les classes de test incluses :

1. **SourceAleatoireTest** : Teste la génération de messages aléatoires par la classe `SourceAleatoire`.
2. **SourceFixeTest** : Valide la génération et la transmission de messages fixes.
3. **TransmetteurParfaitTest** : Vérifie le comportement du transmetteur parfait, qui doit transmettre l'information sans aucune altération.
4. **DestinationFinaleTest** : Contrôle la réception de l'information par la destination et vérifie l'intégrité des données reçues.
5. **InformationTest** : Teste la structure de la classe `Information`, qui gère les informations échangées entre les différents composants.
6. **SimulateurTest** : Évalue le fonctionnement global du simulateur, en simulant une chaîne de transmission complète et en calculant le taux d'erreur binaire.

5.2. Implémentation des tests

Les tests sont implémentés en utilisant le framework JUnit. La suite de tests est définie à l'aide de l'annotation `@RunWith(Suite.class)` pour exécuter tous les tests de manière centralisée. Nous utilisons la classe `AllTests` pour lancer tous les tests.

Les tests ont pour objectifs de valider les entrées et sorties de chaque méthodes en fonction des attentes.

5.3. Résultats des tests

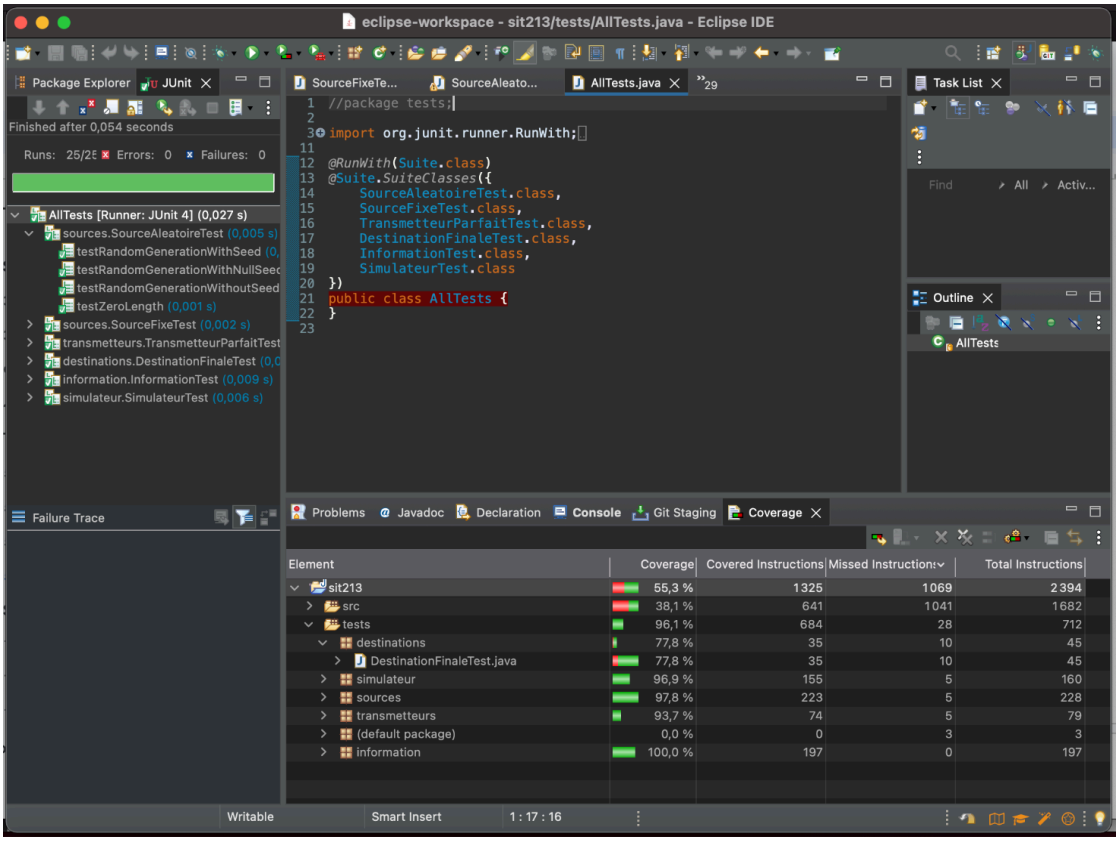
L'exécution des tests a produit les résultats suivants :

- Tous les tests ont passé avec succès, validant le comportement attendu des composants, après quelques changements dans les tests et les sources, notamment grâce à des getters.
- Le simulateur a correctement simulé la transmission d'un message à travers un canal parfait sans erreurs.
- Les classes comme `SourceFixe`, `SourceAleatoire`, `TransmetteurParfait` et `DestinationFinale` ont fonctionné comme prévu, permettant une transmission fidèle des messages avec un taux d'erreur binaire nul dans les scénarios testés.

5.4. Couverture de code avec Emma

Nous avons utilisé l'outil Emma pour mesurer la couverture de code des tests. Voici les résultats globaux et détaillés :

- **Couverture totale** : 96,1 %
- **Couverture des destinations** : 77,8 %
- **Couverture du simulateur** : 96,9 %
- **Couverture des sources** : 97,8 %
- **Couverture des transmetteurs** : 93,7 %
- **Couverture des informations** : 100 %



Ces résultats montrent que l'implémentation a été bien testée, avec une couverture très élevée dans les principaux composants du simulateur. La couverture plus faible pour les destinations (77,8 %) indique qu'il reste des scénarios spécifiques à vérifier, mais globalement, le système est largement testé et fonctionne conformément aux attentes. En effet, certains éléments spécifiques ne sont pas exécutés. Ce qui sera à corriger dans les prochaines itérations. Il manque principalement les tests sur la gestion des arguments et les visualisations, ces classes/méthodes nous ont été fournis. Dans nos classes, les sections non testées sont les méthodes main de debug.

6. Simulation d'une transmission par un canal parfait

Dans cette section, on va simuler la transmission d'un message numérique à travers un canal parfait, c'est-à-dire un canal sans bruit ni interférence. L'idée est de voir comment se comporte une transmission où les informations sont reçues exactement comme elles ont été envoyées. Ça nous permettra de mieux comprendre comment fonctionnent les composants clés de la chaîne de transmission..

6.1. Composants impliqués

La simulation s'articule autour des composants suivants :

1. SourceFixe : Génère un message binaire fixe composé d'une suite de bits (0 et 1). Ce message sera transmis sans aucune altération par la chaîne de transmission.

2. TransmetteurParfait : Ce composant représente un canal de communication parfait, c'est-à-dire qu'il n'introduit ni erreur, ni dégradation du signal au cours de la transmission. Le signal reçu par le transmetteur est donc identique au signal émis.

3. DestinationFinale : Reçoit l'information transmise par le canal et la stocke pour l'analyse. Elle permet de vérifier si l'information reçue correspond exactement à l'information émise par la source.

6.2. Étapes de la simulation

1. Initialisation des composants : Les trois composants principaux sont instanciés :
   - La source génère une séquence binaire fixe.
   - Le transmetteur parfait est initialisé pour transmettre sans erreurs.
   - La destination est prête à recevoir l'information.

2. Connexion des composants : À cette étape, les composants sont reliés entre eux via la méthode connecter(DestinationInterface). La source est connectée au transmetteur, et le transmetteur est à son tour connecté à la destination.

3. Émission du signal : La méthode emettre() est invoquée sur la SourceFixe. Celle-ci génère le message binaire et le transmet au TransmetteurParfait, qui le relaie ensuite à la DestinationFinale sans altération.

4. Réception et vérification : À la fin de la transmission, la DestinationFinale stocke l'information reçue. Le message émis par la source est comparé au message reçu pour vérifier qu'aucune erreur n'a été introduite.

5. Calcul du Taux d'Erreur Binaire (TEB) : Le TEB) est calculé pour quantifier les éventuelles erreurs de transmission. Dans le cas d'un canal parfait, le TEB doit être égal à zéro, signifiant que tous les bits ont été correctement transmis sans erreurs.

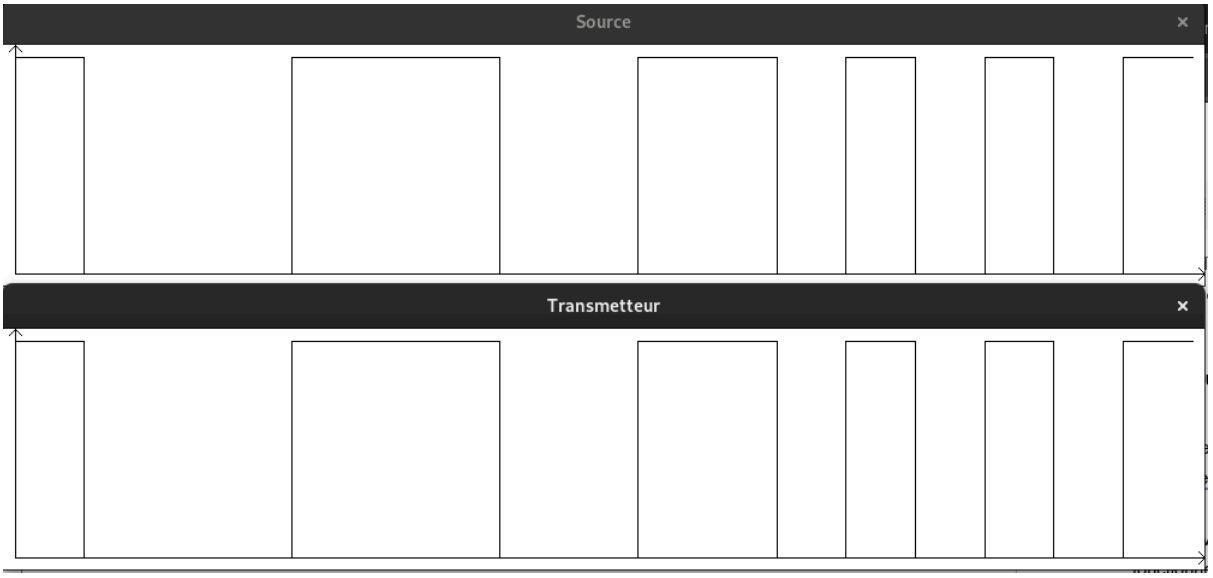
6.3. Résultats attendus

Dans une transmission via un canal parfait, l'information reçue par la destination doit être exactement identique à celle émise par la source. Cela signifie qu'aucune erreur ne doit être observée et que le TEB sera de 0 %. Ce cas constitue une référence pour les simulations futures avec des canaux bruités ou des systèmes de codage, car il permet de valider le bon fonctionnement de la chaîne de transmission dans des conditions idéales.

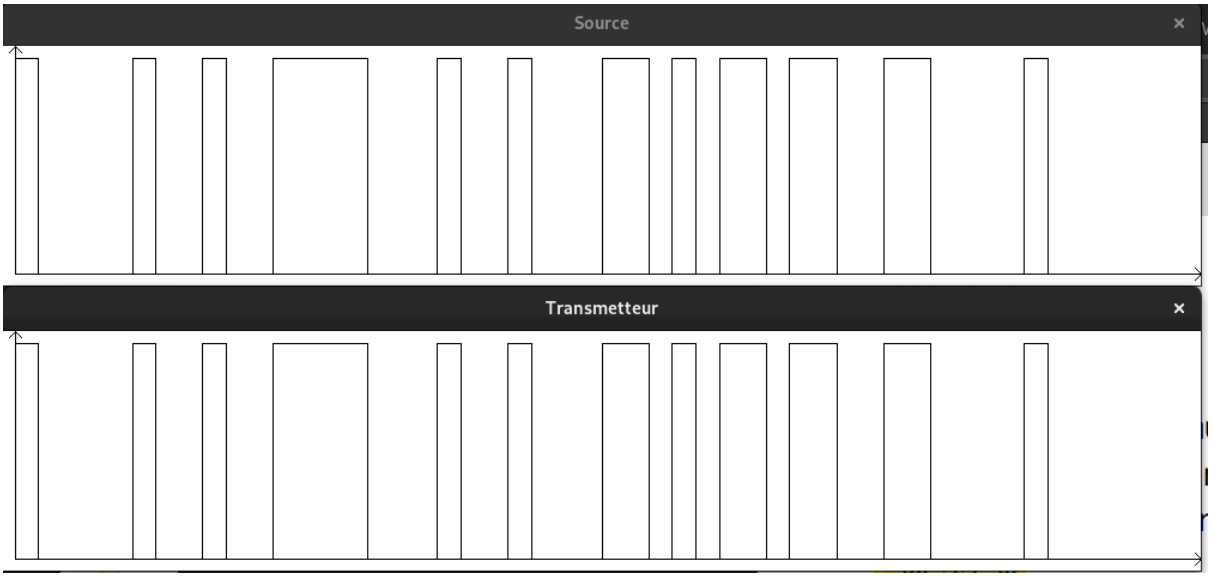
6.3. Résultats obtenus

En exécutant le simulateur avec une source fixe et avec l'affichage des sondes, on retrouve les résultats attendus (même message à la source et transmetteur/destination) correspondant au message passe en paramètres, l'affichage fonctionne avec le paramètre, et on peut utiliser un message aléatoire si indiqué.

```
~/sit213 main ❯ ./simulateur -s -mess "10001110011010101"
java Simulateur -s  -mess  10001110011010101    =>   TEB : 0.0
```




```
~/sit213 main ❯ ./simulateur -s -mess 50 -seed 10
java Simulateur -s  -mess  50  -seed  10    =>   TEB : 0.0
```



Conclusion

À la fin de cette première étape, nous avons :

- Instancier les composants essentiels de la chaîne de transmission (Source, Transmetteur, Destination).
- Connecter les composants de manière logique pour permettre la transmission de l'information.
- Simuler l'émission, la transmission, et la réception de l'information.
- Mis en place le calcul du taux d'erreur binaire (TEB), qui permet d'évaluer la performance de la transmission.
- Effectuer les tests et observer leurs couvertures.

La prochaine étape consistera à introduire les éléments de bruitage et de codage pour améliorer la robustesse de la transmission et à simuler différentes conditions de communication.