Jérémie Rafinesque

jeremie.rafinesque@imt-atlantique.net

Guillaume Chiquet

guillaume.chiquet@imt-atlantique.net

Arnaud Rohé

arnaud.rohé@imt-atlantique.net

Pierre Dugast

pierre.dugast@imt-atlantique.net

SIT213 Fil Rouge

Rapport Etape 2

Version 1,0 - 22/09/2019

Formation d'ingénieur

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

SOMMAIRE

INDEX DES ILLUSTRATIONS

I. INTRODUCTION

Tous les quatre étudiants au sein de l'IMT Atlantique, nous allons vous présenter notre deuxième étape du projet SIT213 au sein de ce livrable. Ce projet va nous permettre d'associer apprentissages du langage Java et simulation de signaux et briques de transmission.

L'objectif final de ce projet consiste à transmettre un message d'un point d'entrée à un point de sortie via un canal de transmission. Nous utiliserons lors de ce projet les connaissances acquises dans les modules SIT 211 (modélisation et validation des logiciels) et SIT 212 (simulation des signaux et briques de transmission).

Ce livrable présente l'ensemble des évolutions à chaque séance de notre projet.

II. LA RÉALISATION GLOBALE ATTENDUE

III. ÉTAPE 1

III.1 Objectif

- ✔ Réaliser une transmission élémentaire « back to back » ;
- ✔ Développer des composants dans le langage de programmation Java ;
- ✔ Mettre en œuvre ces composants à travers une classe de test sous la forme d'une chaîne de transmission.

III.2 Cahier des charges

Pour cette première itération du projet, le but est de réaliser une transmission élémentaire « back-to-back ». Elle est caractérisée par un transmetteur logique « parfait » ; on simule donc un canal de transmission parfait qui n'introduit pas de bruit parasite. On s'attend donc à recevoir un message identique à celui émis. Par conséquent le taux d'erreurs binaire du système doit valoir 0.

Voici l'illustration de la chaîne de transmission à l'étape 1 :

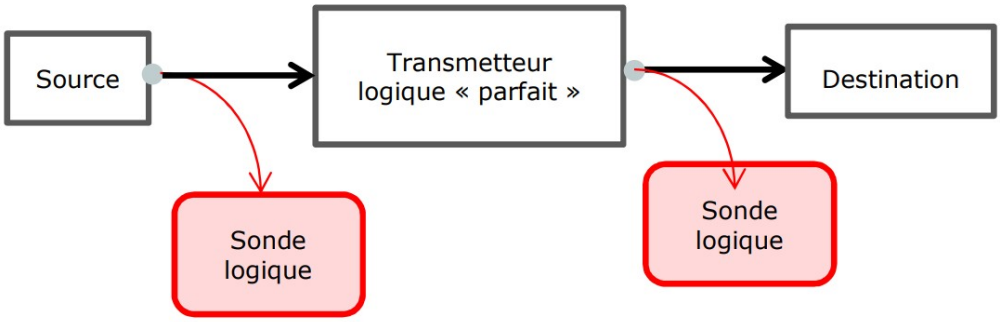

Illustration 1: Simulation Destination Finale

III.3 Développement Logiciel

III.3.1 Réalisation

- Développement de la classe « SourceAléatoire » afin de générer une chaîne Booleenne d'une longueur donnée de bits aléatoires, dans un objet Information ;
- Développement de la classe « SourceFixe » afin de générer une chaîne Booleenne à partir d'une chaîne de caractère donnée composée de '0' et de '1', dans un objet Information ;
- Développement de la classe « TransmetteurParfait » permettant de transmettre simplement un objet Information ;
- Développement de la classe « DestinationFinale » permettant de recevoir simplement un objet Information.

Modification sur l'existant :

- Implémentation de la méthode calculTauxErreurBinaire() de la classe Simulateur permettant de calculer le taux d'erreur binaire en comparant les objets Information ;
- Implémentation de la méthode execute() de la classe Simulateur, afin de rajouter l'option d'affichage des sondes ;
- Implémentation du constructeur Simulateur() de la classe Simulateur, afin de gérer les différentes méthodes de création de donnée binaire, et d'assembler les composant suivant le cahier des charges ci-dessus.

III.3.2 Résultats

Comme escompté, nous recevons bien un message identique à celui émis. Par conséquent le taux d'erreurs binaire du système vaut bien 0. Ces résultats sont tout à fait logiques et conformes à ce qui était attendu.

- Pour illustrer voici un exemple d'exécution avec l'option d'exécution **-mess 1234 -s ;** c'est-à-dire la génération d'un message aléatoire de longueur 1234 :

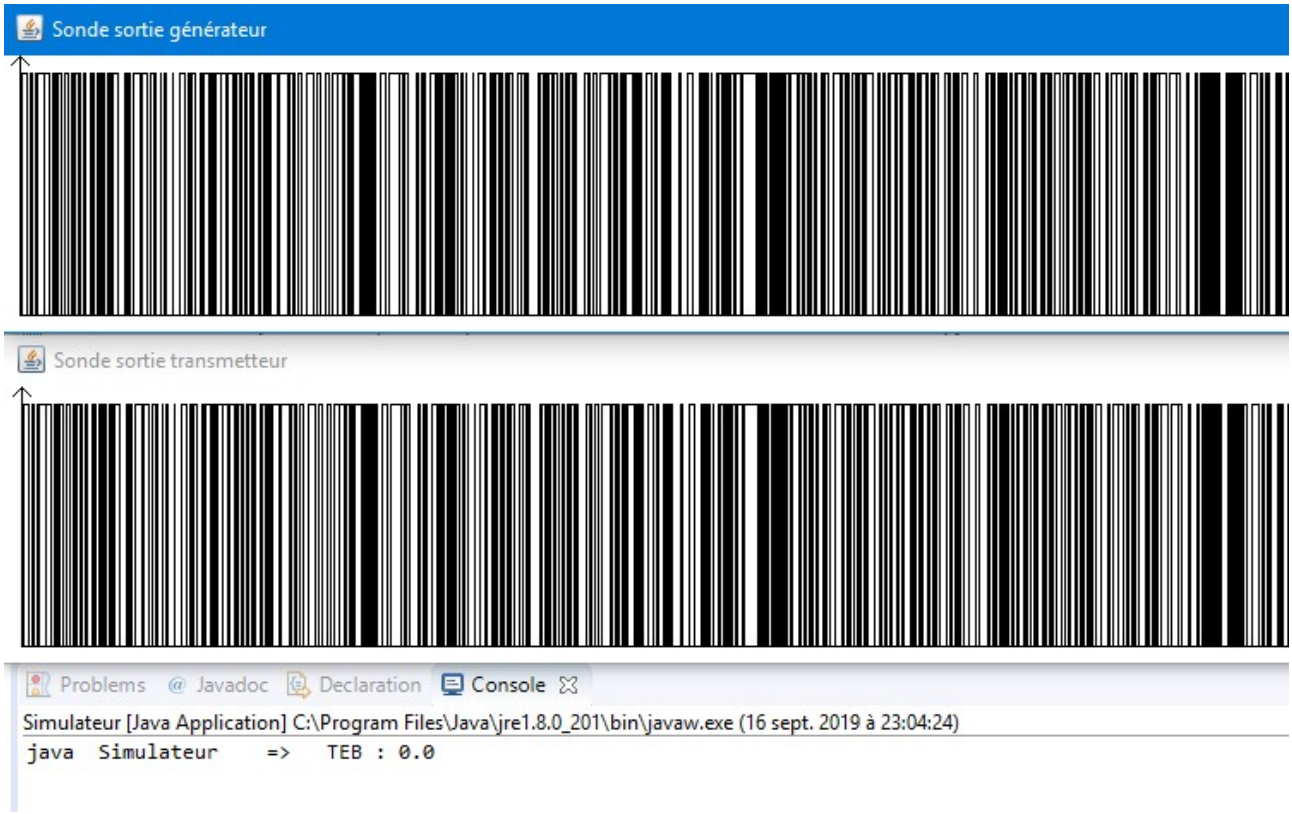

Illustration 2: Résultat transmission logique parfaite, message aléatoire

- Voici un deuxième exemple d'exécution avec l'option **-mess 11001101 -s** ; c'est à dire l'envoi d'une chaîne fixe 11001101.

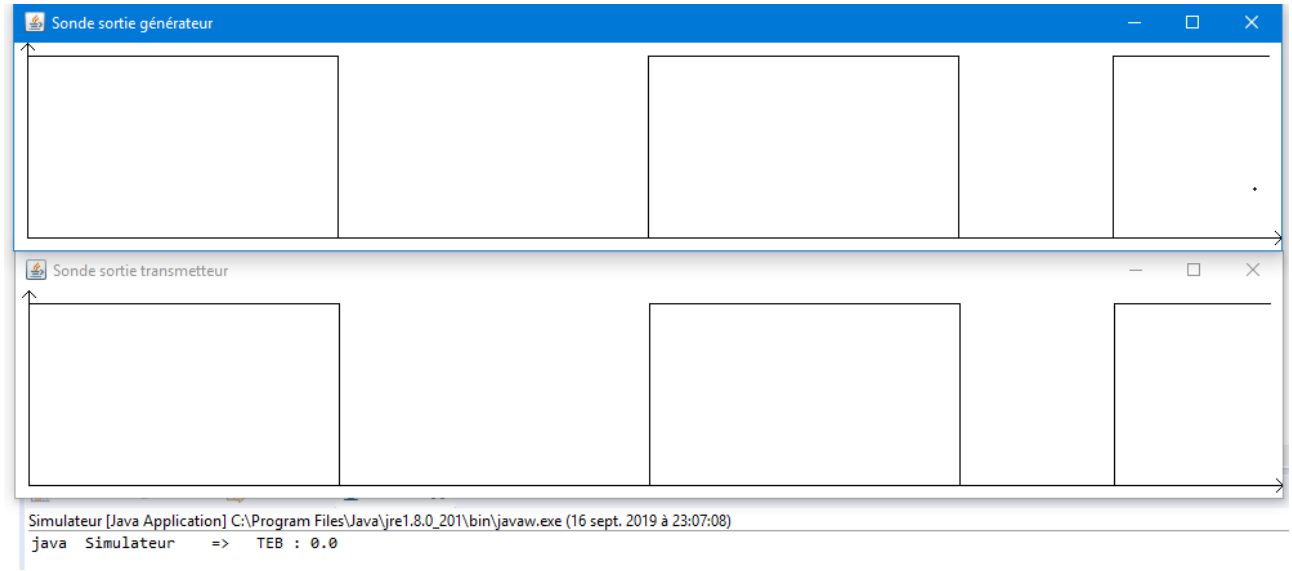

Illustration 3: Résultat transmission logique parfaite, message fixe

III.3.3 Observations

- Les tests du code ont été effectué sans classe tests, directement en manipulant le code : choix de l'efficacité, du fait du peu de classe et de leur faible complexité. La compilation a été testée sous éclipse et sous linux par un script bash ;
- Pas de gestion d'exception ajouté ;
- Les options -s et -mess sont utilisables sans problème ;
- Pas de remarque concernant les résultats obtenu par la simulation.

IV. ÉTAPE 2

IV.1 Objectif

Lors de cette deuxième étape nous travaillons en groupe de 4 avant différents objectifs :

- Réaliser une transmission non bruitée d'un signal analogique ;
- Développer des composants dans le langage de programmation Java :
  - Émetteur (NRZ / NRZT / RZ) ;
  - Récepteur (NRZ / NRZT / RZ) ;
  - Un transmetteur parfait analogique ;
- Mettre en œuvre ces composants à travers une classe de test sous la forme d'une chaîne de transmission.

IV.2 Cahier des charges

Dans cette étape nous devons mettre en œuvre un convertisseur numérique analogique et un convertisseur analogique numérique. L'objectif étant de permettre la transmission non bruité d'un signal analogique. Nous créerons ensuite un transmetteur analogique parfait capable de transmettre le signal analogique, nous vérifions cette transmission à l'aide de sondes analogiques.

Voici le modèle que nous devons construire :

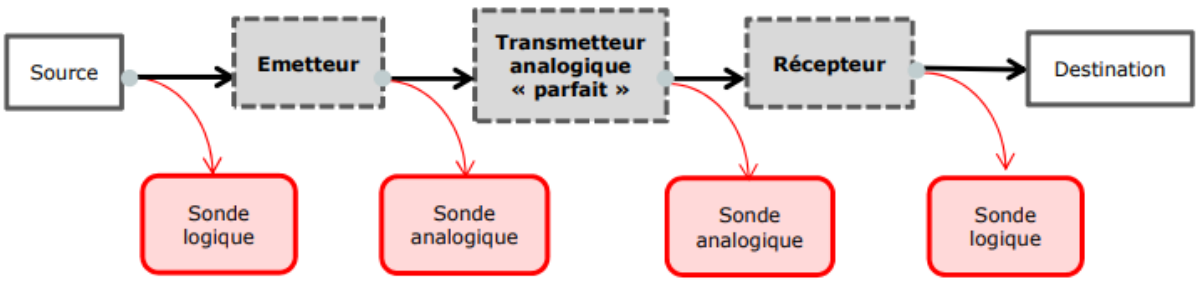

Illustration 4: Modélisation de la chaîne de transmission à l'étape 2

IV.3 Développement Logiciel

IV.3.1 Réalisation

- **Convertisseur Numérique Analogique et Analogique Numérique**

L'objectif étant de transmettre le signal numérique sur le transmetteur parfait analogique, nous allons devoir effectuer une conversion :

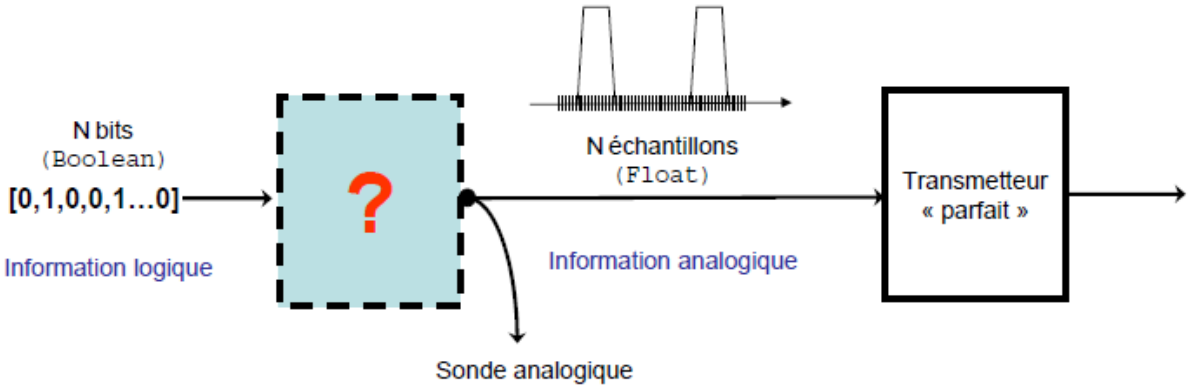

Illustration 5: Conversion Numérique - Analogique

Pour se faire nous avons crée différentes classes, (cf diagramme de classes en annexe) :

- ConvertisseurAnalogiqueNumérique ;
- EmetteurNrz ;
- EmetteurNrzt ;
- EmetteurRz ;
- RecepteurNrz ;
- RecepteurNrzt ;
- RecepteurRz.

Comme ce schéma nous le précise, nous allons avoir en entrée une information logique / numérique à transmettre sous forme d'information analogique.

Pour se faire notre signal binaire sera échantillonné puis convertie en float.

Nous allons utiliser lors de cette séance trois formes de conversion numérique analogique :

- NRZ (Non Return to Zero), forme d'onde rectangulaire ;
- NRZT (Non Return to Zero), forme d'onde trapézoïdale ;
- RZ (Return to Zero), forme d'onde impulsionnelle.

Voici les différentes règles :

- **Codade NRZ :**
  - True = Amplitude max
  - False = Amplitude min

- **Codage NRZT :**
  - True = Amplitudde max
  - False = Amplitude min
  - Le temps de montée de l'amplitude min vers l'amplitude max est de un tiers du temps du bit et le tmeps de descente est aussi d'un tiers. Nous serons donc à un tiers du temps bit à AmplMax ou AmplMin.

(A) CODAGE NRZ

- **EmetteurNrz :**

L'émetteur Nrz reçoit un objet Information Boolean et le transforme en objet Information Float, ses objectifs :

- Récupérer l'information reçue Boolean ;
- Créer un objet Information contenant une liste de Float : chaque True devient un packet de Float de valeur amplitudeMax et de taille nbEchantillon, chaque False devient un packet de Float de valeur amplitudeMin et de taille nbEchantillon.

- **RécepteurNrz :**

Le récepteur Nrz reçoit un objet Information Float et le transforme en objet Information Boolean, ses objectifs :

- Récupérer l'information reçue Float ;
- Créer un objet Information contenant une liste de Float : chaque paquet de taille nbEchantillon dont les bits sont à amplitudemax devient un True, chaque paquet de taille nbEchantillon dont les bits sont à amplitudeMin devient un False.

(B) CODAGE NRTZ

(C) CODAGE RZ

Chaque bit est divisé en 3. De cette manière, le premier et le dernier tiers prennent l'amplitude minimale alors que sur le tiers central on a un maximum au milieu du temps du bit égal à son amplitude maximum. Ce codage est celui qui sera utilisé par défaut pour le signal analogique.

Nous avons eu à construire deux classes pour réaliser ce codage :

- **EmetteurRz :**

Pour se faire nous avons un constructeur qui contient un nombre d'échantillon ainsi que les deux amplitudes (max et min).

Au sein de cette classe et de la méthode émettre nous avons différentes variables :

nbMesure qui permet de définir le nombre de mesure en fonction du nombre d'échantillon et du nombre d'éléments contenues dans l'information reçue. Cette variable va nous permettre de définir un tableau d'informations analogiques.

Nous avons ensuite une variable tiers qui va nous permettre de deviser le tiers le nombre d'échantillons afin d'affecter les valeurs d'amplitudes max et min. Nous avons ensuite une variable tiers qui stock le nombre d'échantillons (divisé par 3) afin d'affecter les valeurs d'amplitudes max et min.

Au sein de cette méthode nous parcourons l'information reçue grâce à la variable i puis nous utilisons la variable j que nous avons initialisé à 0 pour parcourir l'échantillon et ses trois tiers. De cette manière nous allons pouvoir définir le premier tiers comme j inférieur à 1/3 comme valeur d'amplitude minimale, le second comme égal à l'amplitude maximale (si j est inférieur à 2/3) et le troisième comme amplitude minimale dans le cas ou j est inférieur à 2/3.

Enfin nous définissons cette information comme l'information à envoyer et connectons vers les destinations.

- **RecepteurRz :**

Nous souhaitons ici convertir l'information analogique reçue, pour se faire nous allons définir dans la classe RecepteurRz un constructeur, il contient les mêmes paramètres qu'émetteur Rz. Ce qui est logique puisque ce récepteur décode l'information transmise par l'émetteur Rz.

Définition de la méthode émettre :

- Émettre récupère l'information reçue sous forme de tableau de float ;
- L'information reçue est ensuite regroupée par paquet (de la taille d'un échantillon, correspondant à un bit) ;
- Chaque paquet est analysé de la manière suivante :
  - Si dans le paquet il y a une valeur correspondant à l'amplitude Max alors le paquet correspond à un bit à 1 ;
  - Sinon le paquet est un bit à 0.
- En parallèle est remplie une liste de Boolean qui sera emballée dans un objet Information puis envoyé comme Information Émise.

V. NOTES

Nous n'avons pas finalisé l'étape 2 suite à des difficultés rencontrées dans l'élaboration du système, nous n'avons donc pas de simulations à insérer dans ce rapport pour le moment.

VI. ANNEXE

VI.1 Diagramme de Classes étape 2

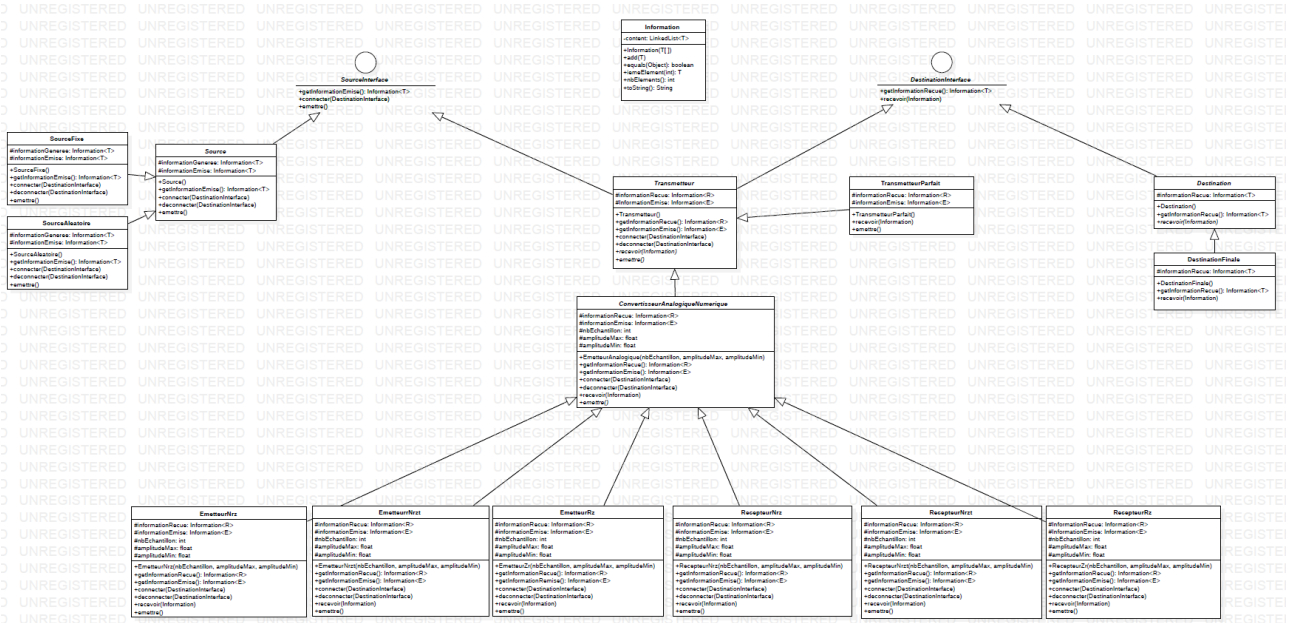

Illustration 6: Diagramme de classes