

|  |
| --- |
| SIT213 Etape 2  IMT Atlantique |
| Chaire M@D - Maintien à domicile | IMT Atlantique |
| 21 septembre  Créé par :  LE DUC Elouan MAQUIN Philippe LE GRUIEC Clément LE JEUNE Matthieu FRAIGNAC Guillaume |

Table des matières

[Intentions du projet 3](#_Toc50887207)

[2ème étape du projet 4](#_Toc50887208)

[ Objectifs 4](#_Toc50887209)

[ Analyse des actions à mener 5](#_Toc50887210)

[ Tests et validations du programme 7](#_Toc50887211)

[ Résultats attendus 7](#_Toc50887212)

[ Tests avec signal de type NRZ 8](#_Toc50887213)

[ Tests avec signal de type NRZT 9](#_Toc50887214)

[ Tests avec signal de type RZ 10](#_Toc50887215)

[Conclusion 11](#_Toc50887216)

[Table des illustrations 12](#_Toc50887217)

# Objectifs généraux

Il s’agit de réaliser, par équipe de 4 ou 5 élèves, une maquette logicielle (en Java) simulant un système de transmission numérique élémentaire. On intégrera donc dans la chaîne un bloc de modulation numérique.

Le système sera assemblé suivant une bibliothèque de modules comportant des ports d’entrée, des ports de sortie et des paramètres physiques. Ces derniers pourront être déterminés à partir des activités du module SIT 212. Le système global sera mis au point progressivement sur 5 séances au cours desquelles les modules seront raffinés, complétés, validés et connectés selon un schéma de transmission de type « point-à-point ».

Outre la qualité technique de la réalisation, on insistera sur les points suivants :

1. La qualité de documentation de la maquette logicielle (notamment la Javadoc).
2. Les efforts de validation des résultats de simulation produits par la maquette.
3. La maîtrise du processus de travail : gestion des versions successives de la maquette logicielle et du dossier technique afférent, synergie de l'équipe, démarche qualité. Concernant ce tout dernier critère, le respect des exigences de mise en forme du livrable sera primordial.

# 1ère étape du projet

## Cible

Transmission élémentaire "back-to-back". On introduira le premier modèle de transmission schématisé sur la figure 1. Il vérifie les propriétés suivantes :

* La source émet une séquence booléenne soit fixée, soit aléatoire.
* Le transmetteur logique parfait se contente, à la réception d’un signal, de l’émettre tel quel vers les destinations qui lui sont connectées.
* La destination se contente de recevoir le signal du composant sur lequel elle est connectée.
* Des sondes logiques permettent de visualiser les signaux émis par la source et le transmetteur parfait.
* L’application principale calcule le taux d’erreur binaire (TEB) du système.



Figure 1 Modélisation de la chaîne de transmission à l’étape 1.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

L’option *-mess m* précise le message ou la longueur du message à émettre :

* Si *m* est une suite de 0 et de 1 de longueur au moins égale à 7, m est le message à émettre.
* Si m comporte au plus 6 chiffres décimaux et correspond à la représentation en base 10 d'un entier, cet entier est la longueur du message que le simulateur doit générer et transmettre.
* Par défaut le simulateur doit générer et transmettre un message de longueur 100.

L’option *-s* indique l’utilisation des sondes. Par défaut le simulateur n’utilise pas de sondes

## Analyse des actions à mener

Pour guider le développement de notre simulateur, un diagramme de classe (figure 2) nous a été fourni. Cela permet d’avoir une vue globale sur la structure à mettre en œuvre ainsi que sur les parties spécifiques à développer dans le cadre de l’étape 1 du projet.



Figure 2 Diagramme de Classe du projet simulateur

De manière général chaque classe correspond à une fonctionnalité spécifique de notre chaine de transmission. Des classes mères abstraites et des interfaces ont déjà été développées au préalable par l’équipe enseignante. Nous pouvons ainsi avoir une base commune sur le projet. Les sous-blocs restent à développer en prenant en compte l’existant et les objectifs.

D’après le diagramme de classe nous avons 4 blocs à implémenter, cela correspond aux éléments en jaune vif sur la figure 2. Pour nous donner des indications sur les méthodes à écrire et sur les interactions entre les différents éléments du programme nous disposons également d’un diagramme de séquence. Les nouveaux blocs seront orchestrés depuis la classe *Simulateur*, le « main ».

Les conditions et indications décrites dans la cible peuvent s’apparenter à un cahier des charges. Ceux sont des éléments à prendre impérativement en compte dans le développement de l’ensemble des fonctionnalités.

## Programmation et connexion des blocs

Dans cette partie nous allons parler brièvement des éléments qui ont été programmés. Nous nous concentrerons sur la classe S*imulateur* afin de ne pas alourdir le compte-rendu. Les codes sources des classes développées sont disponibles dans l’archive fournit avec ce document. Une JavaDocs est également disponible afin d’aider à la compréhension des méthodes implémentées.

### Mise en place des liaisons entre les blocs

Le constructeur de la classe *Simulateur* est programmé selon les indications que nous avons eues. C’est-à-dire que l’ensemble des blocs de la chaine de transmission doivent être instanciés et connectés ensemble. Des « sondes » permettent de pouvoir visualiser graphiquement l’entrée ou la sortie d’un bloc. C’est une fonctionnalité qui sera très utile pour les tests à réaliser.

Le code permettant de mettre en œuvre la chaine selon le schéma de la figure 1.

public Simulateur(String[] args) throws ArgumentsException {

        //Analyse des arguments

        analyseArguments(args);

        //Instanciations des differents blocs de traitement

        if (messageAleatoire) {

            source=new SourceAleatoire(nbBitsMess);

        } else {

            source=new SourceFixe(messageString);

}

        transmetteurLogique = new TransmetteurParfait();

        destination = new DestinationFinale();

        //Instanciations des differentes sondes

        SondeLogique viewSrc = new SondeLogique("ViewSrc", 720);

        SondeLogique viewTransmit = new SondeLogique("ViewTransmit", 720);

        //connexion des blocs ensembles

        source.connecter(transmetteurLogique);

        if(affichage) source.connecter(viewSrc);

        transmetteurLogique.connecter(destination);

        if(affichage) transmetteurLogique.connecter(viewTransmit);

}

### Calcul du TEB

Le programme doit calculer le TEB (Taux d’Erreur Binaire) du système. Pour calculer ce taux nous utilisons la formule suivante :

Pour l’étape 1 ce TEB est optionnel puisque nous avons affaire à un transmetteur parfait. Si tout à bien été fait nous devons avoir un résultat final à 0.0 quel que soit le signal en entrée du système.

En Java le TEB est implémenté de la manière suivante :

public float calculTauxErreurBinaire() {

        int nbErr=0;

        float TEB=0.0f;

        for (int i = 0; i < destination.getInformationRecue().nbElements(); i++) {

            if(destination.getInformationRecue().iemeElement(i)

!=source.getInformationEmise().iemeElement(i)) nbErr++;

        }

        TEB=(nbErr\*1.0f)/(source.getInformationEmise().nbElements());

        return TEB;

}

## Réalisation des tests

## Tests et validations du programme

### Vérification du TEB

Dans le cas d’une transmission parfaite le TEB attendu pour tout signal est à 0. Ce test est validé et fonctionne peu importe le signal, qu’il ait été généré de manière aléatoire ou qu’il ait été fixé.







Par acquis de conscience j’introduis volontairement 33 erreurs dans le signal par défaut afin de vérifier le calcul. Je m’attends alors à voir un TEB à 0.33. Et c’est bien le cas :



### Vérification des signaux entrée / sortie

Cas avec un signal par défaut (longueur 100, aléatoire) :



Figure 3 Signal aléatoire par défaut 1

Comme on pouvait s’y attendre le signal est identique en entrée et en sortie. Le TEB à 0 le confirme.

Je réalise un second lancer afin de vérifier que le signal est bien aléatoire :



Figure 4 Signal aléatoire par défaut 2

Les 2 signaux sont une nouvelle fois identique (TEB à 0), de plus il y a bien un message différent entre le lancer 1 et le lancer 2. Ces tests sont donc concluants.

Cas avec un signal aléatoire de longueur fixée :

Je lance le Simulateur en indiquant de générer un signal de longueur 10 :



Le message généré doit donc faire une longueur de 10 bits.



1 2 3 4 5 6 7 8 9 10

Figure 5 signal aléatoire de longueur fixée à 10

Le résultat obtenu est bien celui attendu. Le test est concluant.

Cas avec un signal fixé :

Je décide de mettre en paramètre le message fixe suivant : 10101010



Le message transmis doit donc correspondre à ce message, une alternation de niveaux logiques haut et bas graphiquement.



Figure 6 signal fixé sur 01010101

Le résultat obtenu est bien celui attendu. Le test est concluant.

# Capitalisation du travail réalisé



C’est un projet qui est réalisé en groupe. Dans l’intérêt de chacun et pour disposer d’un système de versionnage nous utilisons les outils git et un dépôt GitHub.

Cette partie du programme a été déposée et versionnée dans un répertoire GitHub commun à tous les membres de nos groupes finaux. Ainsi nous pourront mettre en commun nos aboutissements sur cette première étape.

# Conclusion

Cette première étape nous a permis d’appréhender le projet en réalisant une chaine de transmission simple dites « Back to Back ».

Au travers d’une analyse, puis en passant par le développement et pour finir sur une batterie de tests, nous sommes arrivés à mettre en place un système de communication. Il met en évidence les éléments importants d’une chaine de transmission comme la source, le transmetteur et le récepteur (appelé destination ici).

Dans l’étape 2 nous devrons implémenter un système de communication analogique non bruitée. L’approche sera certainement différente, il faudra de nouveau effectuer les étapes d’analyse avant de commencer à programmer les nouveaux blocs.

L’approche des télécommunications par la programmation est très intéressante. C’est une approche pratique qui permet de bien comprendre l’utilité et l’organisation des système de communication. De plus c’est un projet pluridisciplinaire qui fait appel à diverses connaissances et qui permet bien sûr d’en développer des nouvelles.

# Intentions du projet

Il s’agit de réaliser, par équipe de 4 ou 5 élèves, une maquette logicielle (en Java) simulant un système de transmission numérique élémentaire. On intégrera donc dans la chaîne un bloc de modulation numérique.

Le système sera assemblé suivant une bibliothèque de modules comportant des ports d’entrée, des ports de sortie et des paramètres physiques. Ces derniers pourront être déterminés à partir des activités du module SIT 212. Le système global sera mis au point progressivement sur 5 séances au cours desquelles les modules seront raffinés, complétés, validés et connectés selon un schéma de transmission de type « point-à-point ».

Outre la qualité technique de la réalisation, on insistera sur les points suivants :

1. La qualité de documentation de la maquette logicielle (notamment la Javadoc).
2. Les efforts de validation des résultats de simulation produits par la maquette.
3. La maîtrise du processus de travail : gestion des versions successives de la maquette logicielle et du dossier technique afférent, synergie de l'équipe, démarche qualité. Concernant ce tout dernier critère, le respect des exigences de mise en forme du livrable sera primordial.

# 2ème étape du projet

## Objectifs

Dans cette phase du projet nous allons travailler sur une transmission non bruitée d’un signal analogique. On prendra en compte la nature analogique du canal de transmission en faisant évoluer la chaîne de transmission par l’adjonction de deux étages (logique → analogique et analogique → logique), comme indiqué sur le schéma de la figure 1.



Figure Modélisation de la chaîne de transmission à l’étape 2.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

L’option *-mess m* précise le message ou la longueur du message à émettre :

* Si *m* est une suite de 0 et de 1 de longueur au moins égale à 7, m est le message à émettre.
* Si m comporte au plus 6 chiffres décimaux et correspond à la représentation en base 10 d'un entier, cet entier est la longueur du message que le simulateur doit générer et transmettre.
* Par défaut le simulateur doit générer et transmettre un message de longueur 100.

L’option *-s* indique l’utilisation des sondes. Par défaut le simulateur n’utilise pas de sondes

L’option *-form f* précise la forme d’onde. Le paramètre f peut prendre les valeurs suivantes :

* NRZ : forme d'onde rectangulaire
* NRZT : forme d'onde trapézoïdale
* RZ : forme d'onde impulsionnelle

L’option *-nbEch ne* en transmission analogique, précise le nombre d’échantillons par bit.

L’option *-ampl min max* en transmission analogique précise l’amplitude min et max du signal

## Analyse des actions à mener

Nous allons devoir programmer un simulateur de signaux analogiques sur une machine. La nature du signal ne s’y prête pas. C’est pourquoi nous allons essayer de faire au mieux en suréchantillonnant des signaux logiques. Il faudra transformer une liste de Boolean en une liste de Float. Les True et les False prendront des valeurs différentes en fonctions de la forme et des amplitudes choisies. Cela correspondrait au schéma de la figure 2 ci-dessous.



Figure Schéma de transformation de Boolean en Float

Nous avons 3 formes possibles à prendre en compte :

* NRZ

C’est le type de forme la plus simple. Chaque bit correspond à une valeur, il suffit donc d’insérer dans l’information X valeurs correspondant au bit à convertir. Par exemple si on tombe sur un 1 (True), on mettra 30 Float de valeur 5.0 dans l’information, et pour le cas d’un 0, 30 Float de valeur -5.0. Cela ne sera pas forcément très réaliste car nous aurons un signal analogique très carré en sortie. Cela correspond au schéma de la figure 1.

* NRZT

Une forme dérivée du NRZ et plus réaliste car on intègre une pente sur les fronts montants et descendants. Chaque symbole sera en tiers de période afin de prendre en compte la phase montante, stabilisée et descendante. Quand 2 symboles identiques se suivent il ne faudra pas réaliser l’ensemble des phases et garder le signal sur son seuil (Amax ou Amin).

Figure NRZT avec 10110

* RZ

C’est une forme qui est particulièrement lisible. Lorsque l’on a un 1 (True), elle maintiendra la valeur Amax sur un tiers de sa période seulement et reviendra à 0 sur les deux autres tiers. La valeur max se situe sur le tier central. Pour un 0 (False), elle reste à 0. Avec cette forme, même si 2 bits de mêmes valeurs se suivent, les valeurs retourneront à 0 sur 2 tiers de période.

Figure RZ avec 10110

D’après le schéma de la figure 1 nous devons programmer 3 nouveaux blocs : un émetteur, un transmetteur analogique parfait et un récepteur. Les sources et destinations sont les mêmes qu’à l’étape une. Quand on y regarde de plus près, l’émetteur et le récepteur sont des transmetteurs qui encodent ou décodent nos informations. Ils ont tous deux, une entrée et une sortie, l’une pour des Float, l’autre pour des Boolean. Nous pourrons donc faire hériter ces 3 nouveaux blocs de la classe Transmetteur et bénéficier des fonctionnalités déjà programmées. Pour ce qui concerne les sondes, elles ont déjà été programmées par les enseignants.

Nous devons également programmer la gestion des paramètres du logiciel afin de pouvoir configurer notre simulateur facilement sans avoir à retourner dans le code source. Certaines fonctions avaient déjà été réalisées par les enseignants, d’autres restent à programmer.

Nous devons implémenter les paramètres suivants : *-form f, nbEch ne, -ampl min max, -seed v.*

Pour chacune de ces fonctionnalités nous devons détecter les paramètres du programme et s’assurer de leur(s) conformité(s) en utilisant des Regex par exemple.

La fonctionnalité -seed pour paramétrer une graine dans le cadre des générations aléatoires avait été omis dans l’étape 1. Nous l’avons donc ajouté.

## Tests et validations du programme

À ce stade du projet nous n’avons pas encore automatisé les tests mais nous pouvons déjà en réaliser grâce aux données tel que le TEB et l’affichage des sondes.

### Résultats attendus

Nous sommes dans le cas d’une chaine de transmission avec un transmetteur analogique parfait. Le signal émis doit être identique au signal reçu qu’il soit converti sous une forme NRZ, NRZT ou encore RZ.

Sur les sondes les signaux sources et destinations sont identiques, les signaux en sorti de l’émetteur et du transmetteur sont également identiques. Dans tous les cas le TEB doit être à 0 à la fin de la chaine de transmission si tout se passe bien. Les formes d’ondes choisies doivent être utilisées selon leur spécifications.

Pour les tests nous utiliserons un signal avec les paramètres suivants :

* Amplitude max : 5V
* Amplitude min : -5V
* Nb échantillon : 60
* Seed : 40
* Longueur (mess) : 20

Le signal aléatoire généré sera égal à 1101 0011 0100 1001 1111

Ce message est intéressant, il permet de tester différents cas de figure, par exemple deux 1 qui se suive, le passage d’un 0 à un 1 etc…

### Tests avec signal de type NRZ





Figure Tests du signal de type NRZ

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique s’est réalisée comme prévu, 5V pour un bit à 1 et -5v pour un bit à 0. Le test est validé.

### Tests avec signal de type NRZT





Figure Tests du signal de type NRZT

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique s’est réalisée comme prévu, 5V pour un bit à 1 et -5v pour un bit à 0. Que cela soit un 0 ou un 1 qui est transmis, il y a des pentes. Lorsque deux symboles identiques se suivent, le niveau est maintenu, il n’y a pas de retour à l’amplitude min. Le test est validé.

### Tests avec signal de type RZ





Figure Tests du signal de type RZ

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique a réagi comme prévu, 5V pour un bit à 1 et 0V pour un 0. Pour un 0 il n’y a pas de changement de niveau, pour un 1 nous avons bien 2 tiers du symbole à 0 et 1 tiers du symbole à 5V. Quand deux 1 se suivent il y a quand même un retour à 0. Le test est validé.

# Conclusion

Nous avons ainsi pu simuler une conversion numérique à analogique et inversement, analogique à numérique. Les tests se sont passés comme prévus, les résultats sont donc satisfaisants.

Dans l’étape 3 nous devrons implémenter une transmission non-idéale avec canal bruité de type « gaussien ». Il faudra programmer un ou plusieurs nouveaux blocs afin d’y arriver. Le travail réalisé lors de cette seconde étape sera une base importante pour la suite du projet.

# Table des illustrations

[Figure 1 Modélisation de la chaîne de transmission à l’étape 2. 4](#_Toc50887442)

[Figure 2 Schéma de transformation de Boolean en Float 5](#_Toc50887443)

[Figure 3 NRZT avec 10110 5](https://retis-my.sharepoint.com/personal/clegruiec_retis_fr/Documents/IMT/IMT-SIT213/LEGRUIEC-LEDUC-FRAIGNAC-MAQUIN-LEJEUNE-2-Rapport.docx#_Toc50887444)

[Figure 4 RZ avec 10110 6](https://retis-my.sharepoint.com/personal/clegruiec_retis_fr/Documents/IMT/IMT-SIT213/LEGRUIEC-LEDUC-FRAIGNAC-MAQUIN-LEJEUNE-2-Rapport.docx#_Toc50887445)

[Figure 5 Tests du signal de type NRZ 8](#_Toc50887446)

[Figure 6 Tests du signal de type NRZT 9](#_Toc50887447)

[Figure 7 Tests du signal de type RZ 10](#_Toc50887448)