République Algérienne Démocratique
Et Populaire
Ministère de l'enseignement supérieur et de la
Recherche scientifique

Université Mouloud Mammeri de Tizi-Ouzou
FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE

DEPARTEMENT INFORMATIQUE

Mémoire de Master

Spécialité : Informatique

Option : Réseaux, Mobilité et Systèmes Embarqués

Thème :

« Application répartie pour un réseau de Capteur sans fil »

Réalisé par **:**

- Sahi Louiza
- Hamouche Louiza

Proposé et dirigé par **:**

HADAOUI Rebiha

Soutenance : 11\07\2017.

Remerciements

C'est avec plaisir que nous réservons ces quelques lignes en signe de gratitude et de profonde reconnaissance à l'égard de tous ceux qui ont aidés à la réalisation de notre projet de fin d'études.

Nous tenons à exprimer nos sincères remerciements à notre promotrice Mme. HADAOUI pour l'intérêt et l'aide qu'elle nous a donné.

Que les membres de jury trouvent nos profondes gratitudes pour l'honneur qu'ils nous font en assistant à notre soutenance.

Nous remercions aussi tous ceux, et celles qui ont contribué de près ou de loin pour l'accomplissement de ce modeste travail.

Dédicace :

Je dédie ce modeste travail :

À mes très chers parents, qui m'ont beaucoup aidé et soutenu durant ma vie et surtout dans mes études. Je pris dieu le tout puissant de les protéger du mal et les récompenser de toutes les peines et sacrifices données aux quels je ne rendrai jamais assez.

À mes chères frères adorés :Abderrezak, Mokrane, Massyl.

À mes chères sœurs adorées : Hanane, Fahima.

À mon cher Jugurtha qui n'a pas cessé de m'encourager et de me soutenir tout au long de ce travail.

À tout mes amis (es) et ceux qui m'ont aidé à réalisé ce projet.

À ma binôme, celle qui a contribué à ce travail avec beaucoup de sérieux.

Louiza

Dédicaces

En signe de reconnaissance et de respect, je dédie ce modeste travail :

Aux êtres les plus chers à mon cœur, mes parents, qui m'ont aidé et soutenu tout au long de mon parcours scolaire et universitaire.

À ma chère sœur Amel, son mari Mohand ainsi que leur petite fille Céline tout en leurs souhaitant beaucoup de réussite.

À ma chère sœur Hassiba.

À mon cher Samir qui n'a pas cessé de m'encourager et de me soutenir tout au long de ce travail.

À tous mes oncles et mes tantes et leurs familles.

À tous mes amis (es).

Sans oublier, celle qui a contribué à ce travail avec beaucoup de sérieux,

Je cite : Louiza et toute sa famille.

Louiza

Aujourd'hui, les progrès réalisés dans les domaines de la micro-électronique, et des technologies de communication sans fil permettent de créer de petits systèmes communicants appelés capteurs. Ces derniers sont équipés d'une unité de mesure, d'une unité de calcul, de mémoires et d'une radio pour communiquer, donc ce sont de véritables systèmes embarqués.

Le déploiement de plusieurs capteurs, en vue de collecter et transmettre des données environnementales vers un ou plusieurs points de collecte appelée(s) puits, et de manière autonome, forme un réseau de capteurs sans fil.

Les RCSF ont suscité beaucoup d'engouement dans la recherche scientifique en raison notamment des nouveaux problèmes de routage sous forte contrainte de durée de vie du réseau et de faibles capacités des nœuds. Malgré la miniaturisation et la réduction du coût de fabrication, ces capteurs sont généralement dotés de ressources limitées en termes de puissance de calcul, de capacité de traitement et de stockage des données et d'énergie. Ces contraintes matérielles ont influencé une grande partie des problématiques de recherche du domaine.

En raison des contraintes de ressources limitées citées au dessus, un capteur ne peut pas réaliser un calcul complexe car les processeurs des réseaux de capteurs utilisent souvent des microcontrôleurs de faibles fréquences.

L'objectif principal de notre travail, est de tirer profit des ressources inutilisées des nœuds du réseau (mémoire, temps processeur, etc.) et d'explorer la possibilité d'utiliser un réseau de capteur sans fil dans le but de réaliser un calcul complexe. Un processus nécessitant un calcul considérable est divisé en de nombreuses petites tâches, chacune répartie sur les différents nœuds du réseau.

Pour cela, nous allons implémenter un nouveau module d'application qui sera créé dans un environnement de simulation OMNeT++ avec la plate forme Castalia.

Pour mener à bien notre travail, nous l'avons organisé en quatre chapitres selon un plan méthodologique suivant :

Dans le premier chapitre, nous présenterons les réseaux de capteurs sans fil : leurs architectures de communication et leurs applications. Nous discuterons également sur les principaux facteurs et contraintes qui influencent la conception des réseaux de capteurs sans fil.

Dans le deuxième chapitre, nous présenterons dans la première partie les concepts de base liés à la simulation des réseaux. Un comparatif entre les différents simulateurs réseaux enrichira le contenu du chapitre. Dans la deuxième partie, nous parlerons d'une façon détaillée sur le simulateur réseau que nous avons conçu qui est OMNeT++.

Dans le troisième chapitre, nous décrirons le simulateur Castalia qui est une plate forme pour les réseaux de capteurs sans fil basée sur OMNeT++.

Dans le dernier chapitre, nous présenterons la conception de notre solution pour le calcul distribué dans les réseaux de capteurs sans fil.

Enfin, nous terminerons ce mémoire par une conclusion générale sur le thème abordé tout en présentant un certain nombre de perspectives pour montrer comment on peut améliorer ce travail.

Introduction

L'essor des technologies sans fil offre aujourd'hui de nouvelles perspectives dans le domaine des télécommunications. En comparaison avec l'environnement filaire, l'environnement sans fil permet aux utilisateurs une souplesse d'accès et une facilité de manipulation des informations à travers des unités de calcul mobiles (PC portable, PDA, capteur…).

Avec les avancées techniques en terme de performances et de miniaturisation, réalisées dans les microsystèmes électromécaniques (MEMS: microcontrôleur, etc.) et les communications sans fil, une nouvelle variante de réseaux ad hoc s'est créée afin d'offrir des solutions économiquement intéressantes pour la surveillance à distance et le traitement des données dans les environnements complexes et distribués : Les réseaux de capteurs sans fil.

Ce chapitre est scindé en deux parties. Dans la première partie, nous donnerons une vue d'ensemble sur les réseaux sans fil ainsi que leurs principales spécificités. Dans la seconde partie, nous présenterons les RCSF, leurs caractéristiques, leurs situation par rapport aux restes de réseaux sans fil, leurs contrainte ainsi que leurs domaines d'applications.

I. Les réseaux sans fil

1. Définition :

Un réseau sans fil comme son nom l'indique, un réseau dans lequel au moins deux périphérique (Ordinateurs, PDA, imprimante, routeur, etc.) peuvent communiquer sans liaison filaire [17].

Les réseaux sans fil ont recours à des ondes radioélectriques (radio et infrarouges) en lieu et place des câbles habituels. Il existe plusieurs technologies se distinguant d'une part par la fréquence d'émission utilisée ainsi que le débit et la portée des transmissions.

2. Avantage des réseaux sans fil :

Les réseaux sans fil procurent plusieurs avantages, notamment [17] :

- L'usage facile dans les endroits à câblage difficile
- La réduction du temps de déploiement et d'installation
- La réduction des coûts d'entretien
- L'augmentation de la connectivité

- La réduction de l'encombrement
- La portabilité, le nomadisme et même la mobilité

3. Caractéristiques des réseaux sans fil :

La principale caractéristique des réseaux sans fil est qu'ils sont libres de tout câblage ou autre matériel encombrant. Voici un ensemble de caractéristiques propres aux réseaux sans fil [1] [2] :

- **Débit** : Suivant la technologie ainsi que le type du média utilisé, le débit peut constituer un grand inconvénient, par exemple : Les réseaux de capteurs fonctionnent avec un débit de quelques Kbps tandis que le Wifi peut atteindre 100 Mbps.
- **Fiabilité** : Les signaux transmis via les ondes radio subissent des perturbations (Erreurs de transmission, coupures …) dues à l'environnement.
- **Sécurité** : La diffusion des messages dans l'air rend les réseaux sans fil sujets à écoute (piratage des messages).
- **Energie** : En cas de mobilité ou de fonctionnement autonome, les réseaux sans fil sont confrontés au problème d'énergie ; une gestion des ressources énergétiques est nécessaire pour augmenter la durée de vie d'un réseau.
- **Mobilité** : La mobilité est l'un des atouts majeurs des réseaux sans fil.

4. Classification des réseaux sans fil :

Le choix de classement des réseaux sans l'étendue (WPAN, WIFI, WIMAX, GSM, GPRS, UMTS) ou bien le mode de communication (avec ou sans infrastructure) [3].

4.1. Selon l'étendue :

On distingue habituellement plusieurs catégories de réseaux sans fil, selon le périmètre géographique offrant une connectivité (appelé *zone de couverture*).

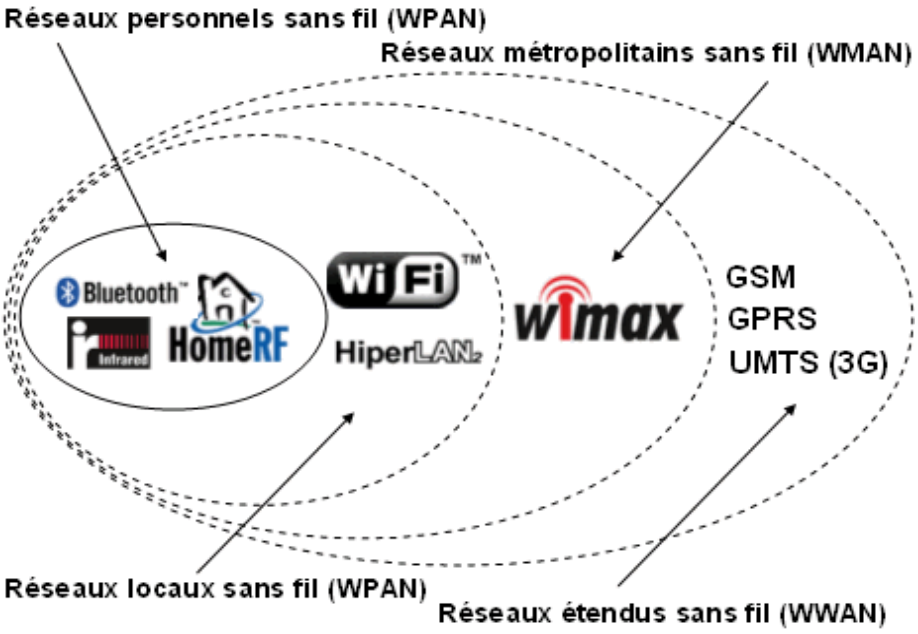


Figure I.1 : Les catégories des réseaux sans fil.

- **Le réseau personnel sans fil (WPAN) :**

Il concerne les réseaux sans fil d'une faible portée : de l'ordre de quelques dizaines de mètres. Ce type de réseau sert généralement à relier des périphériques (imprimante, téléphone portable, appareils domestiques, PDA…). Il existe plusieurs technologies utilisées pour les WPAN [3] :

- La technologie **Bluetooth**
- La technologie **ZigBee**
- Les liaisons **infrarouges**

- **Le réseau local sans fil (WLAN) :**

C'est un réseau permettant de couvrir une portée d'environ une centaine de mètres. Il permet de relier entre-eux les terminaux présents dans la zone de couverture. Il existe plusieurs technologies, la plus connue est le Wifi [3] :

- **Heperlan**
- **Home RF**
- **802.11 de IEEE**

- Les réseaux **Wifi :** Le Wifi est une technologie permettant de créer des réseaux informatiques sans fil (Wireless). Il s'agit d'une norme de l'IEEE baptisée 802.11 qui définie une architecture cellulaire.

➢ **Le réseau métropolitain sans fil (WMAN) :**

Il est connu aussi sous le nom de Boucle Locale Radio (BLR). Il convient de rappeler que la BLR permet, en plaçant une antenne parabolique sur le toit d'un bâtiment, de transmettre par voie hertézienne de la voix et données à haut débit pour l'accès à l'internet et la téléphonie. Il existe plusieurs types de réseaux WMAN dont le plus connu est :

- **Les réseaux Wimax** : ils émanent de la norme IEEE 802.16 et ont pour but de développer des liaisons hertziennes concurrentes aux techniques xDSL terrestres et offrent un débit utile de 1 à 10 Mbit/s dans la bande 10-66 GHz pour une portée de 4 à 10 kilomètres, ce qui destine principalement cette technologie aux opérateurs de télécommunication [3].

➢ **Le réseau étendu sans fil (WWAN) :**

Il est connu sous le nom de réseau cellulaire mobile et il est le plus répandu de tous puisque tous les téléphones mobiles sont connectés à un réseau étendu sans fil. Les principales technologies sont les suivantes [3]:

- **GSM**
- **GPRS**
- **UMTS**

4.2. Selon le mode de communication :

4.2.1. Avec infrastructure :

Dans ce genre de réseaux, la communication entre les différents éléments est faite à travers une station de base ayant des points d'accès, ces derniers servent à connecter les différentes unités mobiles pour constituer le réseau. [2] [3].

Tout message doit impérativement passer par une station de base avant d'atteindre sa destination finale. Ce type de réseaux est le plus utilisé, principalement dans les réseaux cellulaires de téléphonie mobiles.

Figure I.2 : Réseaux sans fil avec infrastructure.

4.2.2. Sans infrastructure :

Dans les réseaux sans infrastructure, les membres du réseau font deux fonctions différentes, ils servent en même temps de client et de point d'accès.

Une communication entre deux éléments du réseau peut passer par un ou plusieurs autres éléments suivant la topologie du réseau. Ce type de réseaux est adapté le plus aux réseaux ad hoc et autres communications instantanées (Par exemple : réseau d'ordinateurs portables …) [2].

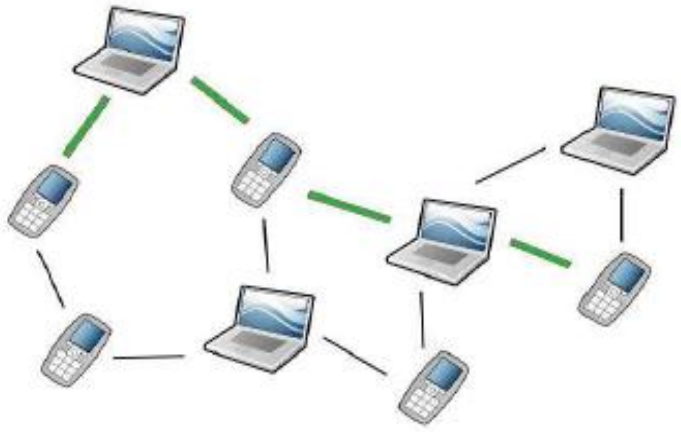


Figure I.3 : Réseaux sans fil sans infrastructure.

5. Les MANETs :

5.1. Définition :

Un réseau ad hoc est ensemble de machines reliées par une technologie sans fil, capables de s'organiser sans l'aide d'une infrastructure. Les communications dans les MANETs se font comme suit :

Un nœud ne peut communiquer qu'avec son voisin direct ; pour pouvoir atteindre des éloignés (destination), un message devra passer par un ou plusieurs nœuds Intermédiaires qui se chargeront de son acheminement. Chaque nœud se comporte donc comme client et routeur en même temps, ce qui rend ces réseaux autonomes [19].

5.2. Caractéristiques :

- **Auto-organisation :** Les nœuds s'adaptent de façon automatique à tout changement de topologie et s'intègrent facilement à tout nouveau réseau d'où les termes : ad hoc, spontané ou encore instantané.
- **Autonome :** Le double rôle des terminaux (client, routeur) les rend indépendants de toute infrastructure, ainsi les termes routeur et client sont confondus dans ce genre de réseaux.
- **Contraintes énergétiques :** L'alimentation basée sur les batteries ainsi que le double rôle des terminaux font que l'énergie est grandement affectée.
- **Sécurité :** L'utilisation des ondes radio qui sont fortement sujettes à écoute, rend ces réseaux vulnérables aux intrusions.
- **Sans fil :** La mobilité des nœuds exige des communications sans fil. Les nœuds s'échangent les messages via les ondes hertziennes [19].

5.3. Domaines d'application :

Grâce aux multiples avantages qu'offrent les réseaux ad hoc, leurs domaines d'utilisation sont très variés [4].

- Les premières applications ont été introduites dans un but militaire, elles permettaient la surveillance et la localisation des véhicules et des soldats dans un champ de bataille.
- A l'intérieur d'une entreprise, un réseau ad hoc est plus adapté pour relier les différents services et machines.

- Dans les bibliothèques et les institutions publiques, un point d'accès à internet permet à n'importe qui possédant un terminal muni d'une technologie sans fil, de se connecter à internet via le point d'accès.
- Les applications les plus récentes des réseaux ad hoc sont la surveillance environnementale (météo, pollution, catastrophes naturelles … etc.), grâce à la technologie nouvelle des réseaux de capteurs.

II. Les réseaux de capteurs sans fil

1. Définition d'un réseau de capteur sans fil :

Les réseaux de capteurs sans fil (RCSF) sont une nouvelle technologie basée sur les MANETs, ils sont constitués d'un ensemble de nœuds-capteurs déployés à travers une surface géographique plus ou moins grande. Chaque capteur effectue la tâche de rassembler les données de son environnement (suivant le domaine d'application).Dès qu'un événement précis se produit, les capteurs concernés (situés près de l'événement) transmettent aussitôt les données vers la station de base appelée aussi puits, afin de les traiter et déduire la source [4].
De nouvelles applications permettent de consulter les données d'un réseau de capteurs sur internet grâce aux communications par satellite.

2. Définition d'un capteur :

Un capteur sans fil est un petit dispositif électronique capable de mesurer une valeur physique environnementale (température, lumière, pression, humidité, vibration, etc.), et de la communiquer à un centre de contrôle via une station de base (puits, **sink**) [4].

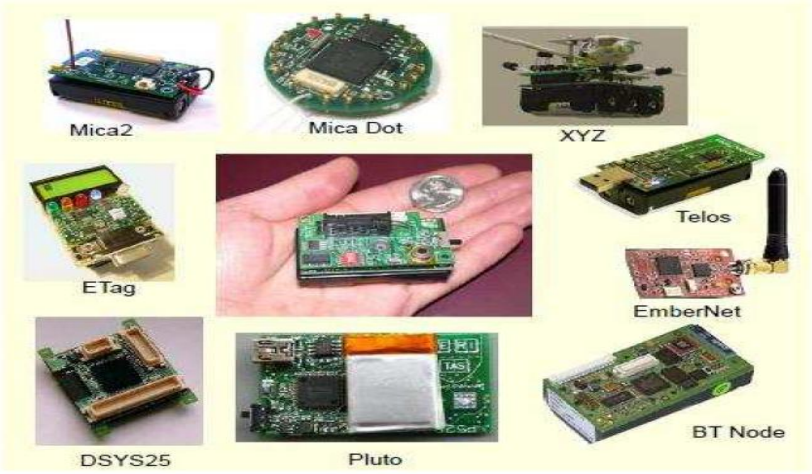


Figure I.4 : Exemples de capteurs.

2.1. Caractéristiques d'un capteur :

- **Sensibilité**: Elle est définie par le rapport entre le signal physique en entrée et le signal de sortie.
- **Précision**: S'exprime en fonction de la grandeur physique mesurée (la plus grande erreur prévue) entre le réel perçu et les signaux électriques obtenus en sortie du capteur.
- **Rapidité**: le temps de réaction d'un capteur entre la variation de la grandeur physique mesurée et l'instant de capter ce changement.
- **Fidélité**: Pour la même entrée, le capteur génère la même sortie
- **Bruit:** produit par un capteur et qui limite les performances d'exécution du système [5].

2.2. Architecture et anatomie d'un capteur :

Un capteur est composé de différentes unités, nécessaires à son fonctionnement [6] :

- **Unité de captage** : Rassemble habituellement deux sous-unités, le capteur et un convertisseur analogique/numérique qui permet de transformer les signaux capturés sous forme numérique pour ensuite les transmettre à l'unité de traitement.
- **Unité de traitement** : Composée d'une petite mémoire et d'un processeur à faible puissance, elle assure les fonctions suivantes : Exécuter les protocoles de routage pour collaborer avec les autres nœuds et participer dans le routage, elle peut aussi effectuer de petits calculs allégeant ainsi la tâche du nœud puits.
- **Unité de transmission** : Une antenne émetteur/récepteur permet de relier le nœud aux différents membres du réseau, elle est la plus coûteuse en termes de consommation d'énergie car un nœud doit être tout le temps actif pour pouvoir détecter et router les messages des autres nœuds.

En plus de ces trois unités communes à plusieurs types de capteurs, il existe d'autres unités optionnelles :

- **Unité de contrôle d'énergie** : Généralement, les capteurs fonctionnent à l'aide de batteries AA (petite taille). La durée de vie du capteur en-dépend. D'où la nécessité de bien répartir l'énergie entre les différents modules, et éviter les dépenses inutiles.

- **Unité de mobilité** : Un petit moteur est ajouté au capteur pour permettre la mobilité de ce dernier. L'unité de mobilité est nécessaire dans certaines applications exigeant un déplacement constant des capteurs.

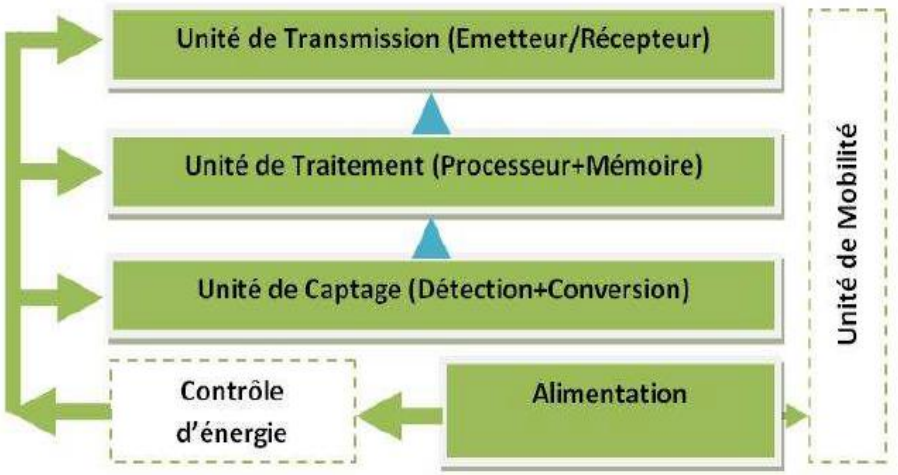


Figure I.5 : Schéma synoptique d'un capteur.

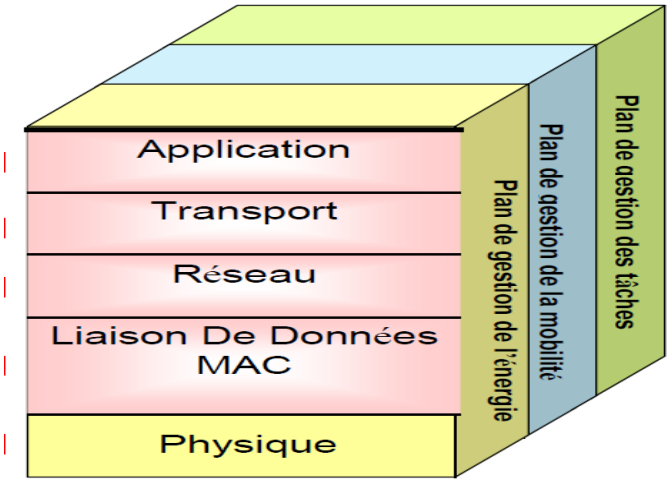


Figure I.6 : La pile protocolaire dans les RCSF.

2.3. Logiciel :

De nombreux produits logiciels destinés aux réseaux de capteurs existent, TinyOS est le plus utilisé d'entre eux.

TinyOS (petit système d'exploitation : en anglais) est un système open-source écrit en NesC, il est développé à l'université américaine Berkeley spécialement pour les réseaux de capteurs, il respecte une architecture basée sur une association de composants, ce qui réduit considérablement la taille de son code.

En s'appuyant sur un fonctionnement événementiel, TinyOS gère la consommation énergétique du capteur et lui permet de s'adapter aux changements de topologie et autres événements aléatoires [7].

3. Les principales caractéristiques des RCSF :

Les facteurs de conception du RCSF (caractéristiques) font l'objet d'études de plusieurs chercheurs. Cependant, aucune de ces études n'intègre une vue totale de tous les facteurs conduisant à la conception d'un réseau de senseurs et de nœuds de senseurs. Ces facteurs sont importants parce qu'ils vont servir comme guide à la conception d'un protocole ou d'algorithmes pour les réseaux de senseurs [12].

- **Energie limitée**: les RCSF étant constitué de micro senseur tirant leur autonomie d'énergie à partir des petites batteries inchangeables. De ce fait, une des contrainte les plus importantes pour les capteurs, le besoin de consommer de basse quantités d'énergie. Les protocoles conçue pour les réseaux de capteurs doivent s'intéresser essentiellement à la conservation de l'énergie .Ils doivent implémenter des mécanismes qui évitent les échanges inutiles entres les nœuds afin de prolonger la durée de vie du réseau.
- **Déploiement dense et aléatoire** : les RCSF se compose d'un nombre très important de nœuds (des centaines voir des milliers). Ce déploiement dense a pour but de permettre une meilleure granularité de surveillance et augmenter la tolérance aux pannes. Ces capteurs sont dispersés par voie aérienne ou par un robot ou un être humain.
- **Topologie dynamique :** la vulnérabilité physique des capteurs est à l'origine des changements très fréquents de la topologie d'un réseau du capteur. Les capteurs sont détruits lors de leurs éjections d'un avion dans la zone ou après avoir épuisés leurs batteries.
- **Auto organisation du réseau** : Compte tenu de l'instabilité dans les RCSF et de la non intervention d'une entité administrative, une auto organisation s'avère nécessaire. En effet, chaque nœud doit être capable de localiser ses voisins et établir des routes pour l'acheminement de l'information au sein du réseau.

- **Tolérance aux fautes** : Quelques nœuds senseurs peuvent tomber en panne ou sont bloqués à cause de l'absence d'énergie, ou ont subi un dommage physique ou sont soumis à l'interférence environnementale. La défaillance de nœuds senseurs n'affecte pas la tâche globale du réseau senseur. C'est la fiabilité ou la tolérance aux fautes. La tolérance aux fautes est la possibilité de soutenir les fonctionnalités du réseau de senseurs sans interruption malgré la présence de nœuds senseurs défaillants.
- **Coûts de production :** Puisque les réseaux de senseurs consistent en un grand nombre de nœuds senseurs, le coût d'un seul nœud est très important pour justifier le coût global du réseau. il est nécessaire de maintenir un coût réduit pour les captateurs. (Actuellement un nœud ne coûte souvent pas beaucoup plus que 1$).
- **Agrégation de données :** Les capteurs génèrent des grands volumes de données bruts. Alors, il est nécessaire de développer des techniques d'agrégation ou de compression des données lors de leur routage.

4. Les communications dans les réseaux de capteurs sans fil :

Différents des réseaux ad-hoc, les RCSF exigent de nouvelles contraintes pour la conception des protocoles de communications [12].

Ces protocoles doivent, en plus de leurs fonctions classiques tenir compte:

- La synthèse et l'agrégation des données
- Les limitations matérielles des capteurs
- La mobilité
- La consommation d'énergie

4.1. La pile protocolaire dans les RCSF

La pile de protocoles utilisée par un nœud puits (**sink**) et un nœud senseur est composée de [1]:

- Couche physique.
- Couche lien de données.
- Couche réseau.
- Couche transport.
- Couche application.

- Plan de gestion de l'énergie.
- Plan de gestion de la mobilité.
- Plan de gestion de la tâche.

➢ **La couche physique :**

S'occupe de la spécification des caractéristiques matérielles, des fréquences porteuses, etc. Elle doit assurer des techniques d'émission, de réception et de modulation de données d'une manière robuste.

➢ **La couche liaison :**

Spécifie comment les données sont expédiées entre deux nœuds dans une distance d'un saut, elle est aussi responsable du multiplexage des données, du contrôle d'erreurs, elle assure la liaison point à point et multipoints dans un réseau de communication

Comme l'environnement des réseaux de capteurs est bruyant et les nœuds peuvent être mobiles, la couche de liaison de données doit garantir une faible consommation d'énergie et minimiser les collisions entre les données diffusées par les nœuds voisins.

➢ **La couche réseau :**

Gère l'adressage et le routage des données, c'est-à-dire leur acheminement via le réseau. Ce routage diffère de celui des réseaux de transmission ad-hoc sans fils par les caractéristiques suivantes:

- Impossible d'établir un système d'adressage global pour le grand nombre de nœuds.
- Les applications des RCSF exigent l'écoulement des données mesurées de sources multiples à un puits particulier. (N →1)
- Les multiples capteurs peuvent produire de mêmes données à proximité d'un phénomène (redondance).
- Gestion rigoureuse des ressources.

➢ **La couche transport :**

Elle est chargée du transport des données, de leur découpage en paquets, du contrôle de flux, de la conservation de l'ordre des paquets la gestion des éventuelles erreurs de transmission.

➢ **La couche application :**

Elle assure l'interface avec les applications. Il s'agit donc du niveau le plus proche des utilisateurs, gère directement les logiciels.

- **Plan de gestion de l'énergie :**

Il montre comment un nœud senseur doit utiliser son énergie.

Par exemple:

- Le senseur met son Emetteur/Récepteur en mode Off après avoir reçu un message d'un de ses voisins. Cela permet d'éviter de recevoir des messages dupliqués.
- Quand le niveau d'énergie d'un nœud senseur est bas, le senseur avise ses voisins par un broadcast que son niveau d'énergie est bas et donc il ne peut participer aux messages de routage. Le reste de l'énergie est réservé à la détection.

- **Plan de gestion de la mobilité :**

Il détecte et enregistre le mouvement des nœuds senseurs, ainsi, un retour arrière vers l'utilisateur est toujours maintenu, les nœuds senseurs peuvent garder la trace de leurs voisins senseurs. En connaissant leurs voisinages, les nœuds senseurs peuvent balancer leur énergie et la tâche d'usage.

- **Plan de gestion de tâches**

Il balance et ordonnance les tâches de détection pour une région spécifique. Ce ne sont pas tous les nœuds de cette région qui sont nécessaires à la tâche de détection en même temps. Comme résultat, certains nœuds senseurs exécutent cette tâche plus que les autres en fonction de leurs niveaux d'énergie.

Remarque :

Ces plans de gestion sont nécessaires pour que les nœuds puissent travailler ensembles pour une meilleure efficacité énergétique, pour router la donnée dans un réseau de senseurs et partager les ressources entre les nœuds senseurs. Sans ces plans de gestion, chaque nœud senseur travaillera de manière individuelle. D'un point de vue d'un réseau de senseurs, il est plus efficace aux nœuds senseurs de collaborer pour pouvoir prolonger la durée de vie du réseau.

4.2. Zone de couverture :

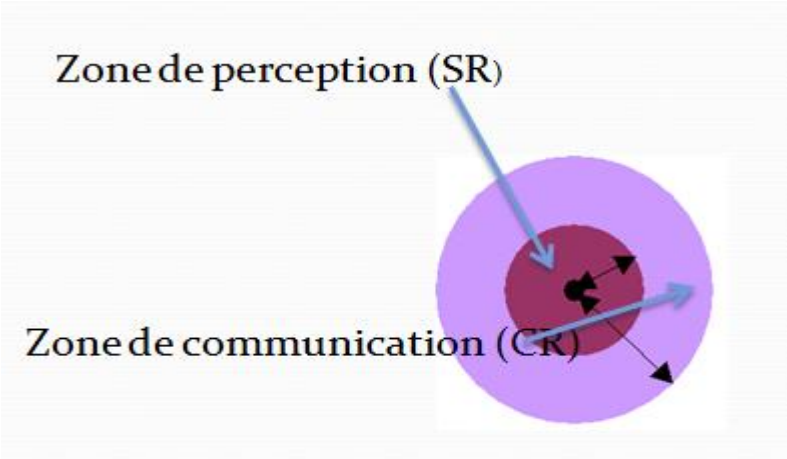


Figure I.7 : Zone de couverture.

Les zones CR et SR représentent la zone de couverture d'un capteur. Pour qu'une zone soit complètement couverte, il faut que la densité de capteurs soit suffisante.

Comme les capteurs sont disposés sur la zone à couvrir de façon aléatoire, il est nécessaire de disposer d'une densité importante de capteurs.

Si la densité de capteurs est trop importante et que la zone que l'on veut surveiller est "trop" couverte, alors des capteurs vont fonctionner inutilement, (gaspillage d'énergie) et pour cela inutilement vont se mettre en veille. Ce mécanisme va permettre d'augmenter la durée de vie du réseau [2].

4.3. Protocole de routage dans les RCSF :

Les protocoles de routage pour les RCSF ont été largement étudiés, et différentes études ont été publiées. Les méthodes employées peuvent être classifiées suivant plusieurs critères ainsi que plusieurs contraintes doivent êtres prises en compte [3] :

- Limitations : Contraintes d'énergie, toutes les couches doivent tenir compte de limitation d'énergie pour maximiser de la durée de vie du réseau.
- Bande-passante.
- Absence d'adressage global.
- Données redondantes.
- Réseau à sources multiples/destination unique.
- Gestion des ressources.
- Capacité de calcul.
- Stockage.

4.4. Topologie du réseau :

La topologie détermine l'organisation des capteurs dans le réseau. Deux principales topologies [7]:

- **Topologie plate :** tous les nœuds possèdent le même rôle, les nœuds sont semblables en termes de ressources**.**
- **Topologie hiérarchique:** divise les nœuds en plusieurs niveaux de responsabilité. L'une des méthodes les plus employées est le clustering, où le réseau est partitionné en groupes appelés "clusters".

  Un cluster est constitué:
  - D'un chef (**cluster-head**).
  - De ses membres.

4.5. Type d'application :

La méthode de captage des données dans un RCSF dépend de l'application et d l'importance de la donnée. De ce fait, les RCSF peuvent être catégorisés comme time-driven ou event-driven [6].

- **Application time-driven :** un réseau time-driven est approprié pour des applications qui nécessitent un prélèvement périodique des données.
- **Application event-driven :** dans des applications temps réel, les capteurs doivent réagir immédiatement à des changements soudains des valeurs captées.

5. Comparaison entre les RCSF et les réseaux sans fil classiques :

Nous distinguons plusieurs critères faisant la différence entre les réseaux de capteurs et les réseaux ad hoc conventionnels, entre autres [5]:

- Le nombre de nœuds est nettement plus grand dans les réseaux de capteurs que dans les réseaux ad hoc
- Les nœuds capteurs sont déployés d'une manière dense.
- Les nœuds capteurs sont plus exposés aux pannes
- Les réseaux de capteurs utilisent principalement les communications broadcast alors que la plupart des réseaux ad hoc sont basés sur la communication point à point.
- Les nœuds capteurs sont caractérisés par des ressources plus limités (ressource d'énergie, puissance de calcul et mémoire).

- Les nœuds capteurs ne possèdent aucune identification (ID) globale tel que les adresses IP dans les réseaux ad hoc.

6. Contraintes spécifiques aux réseaux de capteurs :

Du fait de leur petite taille, les capteurs sont confrontés à plusieurs contraintes [12] :

- **Energie :** La courte durée de vie des batteries exige une gestion des ressources énergétiques du capteur.

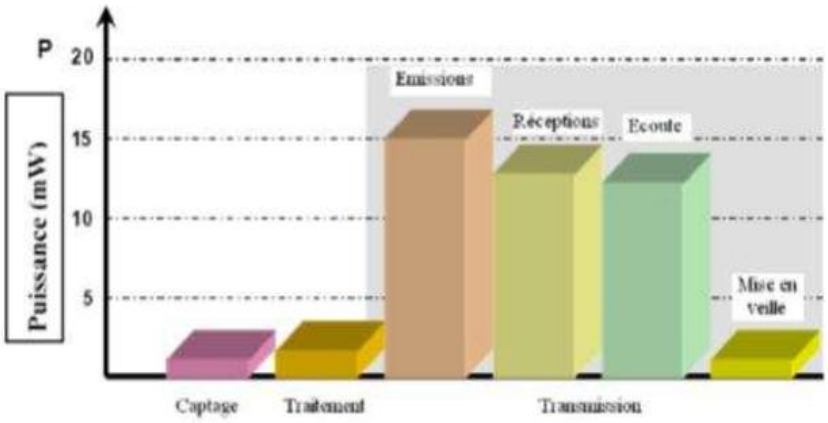


Figure I.8 : Consommation énergétique des unités d'un capteur.

- **Dimension :** La petite taille des capteurs les rend très avantageux ; cependant, ils nécessitent des batteries à leur taille.
- **Puissance de calcul et de stockage :** Les modestes capacités du processeur ne permettent pas de faire de grands calculs, ainsi le rôle du capteur est de rassembler les données de son environnement puis de les transmettre vers la station de base pour traitement.
- **Topologie dynamique :** La topologie d'un réseau peut changer à tout instant, soit par contrainte énergétique ou bien par un nouveau déploiement ; ce qui nécessite à chaque fois une mise à jour au niveau de chaque capteur.
- **Transmission :** La portée des capteurs est souvent affectée à cause de plusieurs paramètres : bruit, obstacles de l'environnement, baisse d'énergie

7. Domaines d'application :

La taille de plus en plus réduite, le coût de plus en plus faible, ainsi que l'utilisation de la technologie sans fil permettent aux réseaux de capteurs de s'étendre à plusieurs domaines d'utilisation [4] [6] [7].

Parmi les domaines les plus connus, on site : le domaine militaire, médical, environnemental, domestique, etc.

- **Militaire :** Les premières applications des réseaux de capteurs ont été introduites dans le domaine militaire, elles consistent en des applications de surveillance : interception des mouvements de l'ennemi, étude des caractéristiques du champ de bataille, détection des attaques nucléaires et chimiques ainsi que la poursuite de la cible.

Figure I.9 : Localisation et surveillance d'un champ de bataille.

Figure I.10 : Déploiement dans un champ de bataille.

- **Environnemental :** Les applications environnementales constituent la plus grande part d'utilisation des réseaux de capteurs, cela est dû au fait que les réseaux de capteurs s'adaptent bien aux multitudes disciplines offertes : surveillance du milieu naturel (air, mer, terre), prévention des incendies et des inondations, étude de la vie sauvage, contrôle de la pollution, aide à l'agriculture.
- **Monitoring des cours d'eau :** Des capteurs (Taux de pollution, niveau des eaux) sont placés aux bords des rivières et autres cours d'eau.

- **Contrôle de la pollution :** Des capteurs placés en haut des bâtiments collaborent entre eux pour mesurer la qualité d'air d'une ville.

Figure I.11 : Contrôle de la pollution dans les cours d'eau.

- **Détection des feux de forêt :** Des capteurs (température, humidité, vitesse du vent, taux de CO2 dans l'air …) sont placés dans différents endroits d'une forêt pour détecter les incendies. Une station de base rassemble et traite les données puis les transmets à l'unité de contrôle

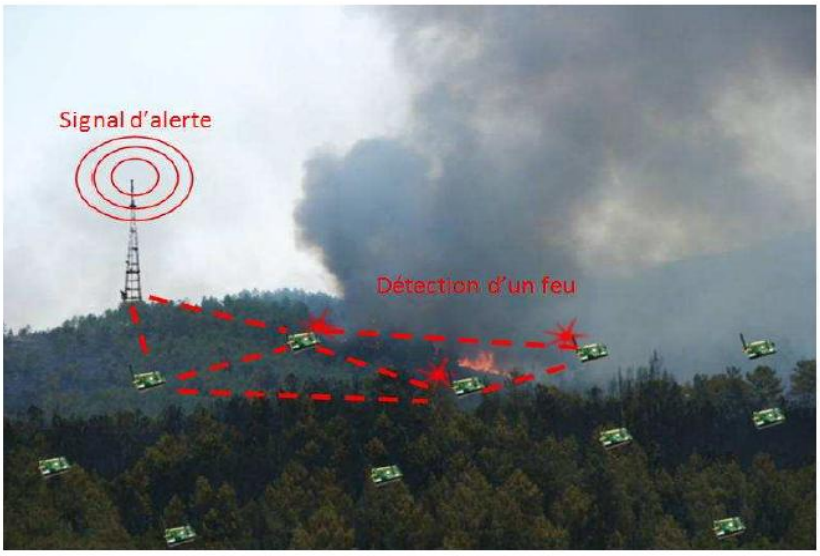


Figure I.12 : Détection d'un feu de forêt.

- **Etude de l'activité volcanique et sismique :** Un réseau de capteurs étalé autour d'un volcan ou sur une grande surface terrestre permet un suivi continu des activités volcaniques ou sismiques.

Figure I.13 : Etude de l'activité volcanique.

- **Contrôle de l'agriculture :** Des capteurs (humidité, température, teneur du sol en sels minéraux) sont déployé sur une surface agricole ou enfouis sous terre pour améliorer la productivité agricole.

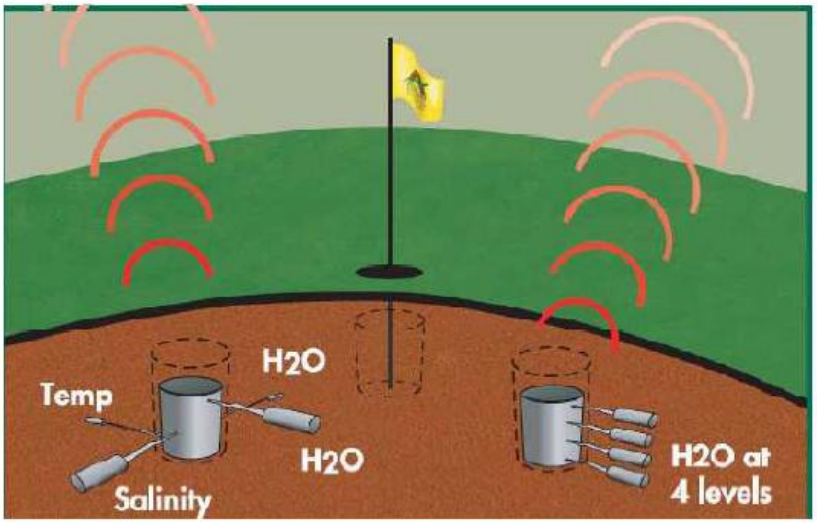


Figure I.14 : Contrôle de l'agriculture.

- **Domotique :** La domotique qui signifie : ensemble de techniques de l'électronique, de l'informatique et de télécommunication utilisées dans les bâtiments, est un domaine bien adapté aux réseaux de capteurs (Mesure de température, lumière, bruit …).

Figure I.15 : Application à la domotique.

8. Déploiement des réseaux de capteurs :

La miniaturisation des équipements électroniques constituant les capteurs, les rend faciles à déployer quel que soit l'endroit [19].

Les nœuds capteurs peuvent être éparpillés sur le champ de captage en masse ou placés d'une manière individuelle et ceci par le biais de plusieurs moyen tel que :

- les jeter d'un avion
- utiliser une artillerie, roquette ou missile, ou
- les placer nœud par nœud d'une façon manuelle ou en utilisant des robots.

Le nombre important de nœud utilisés dans un réseau de capteurs empêche leur déploiement suivant un plan soigneusement établi, cependant un schéma général pour le déploiement initial doit être conçu pour permettre :

- de réduire les coûts d'installation
- augmenter la flexibilité d'arrangement des nœuds
- faciliter l'auto-organisation des nœuds et leur tolérance aux pannes

Après la phase de déploiement, la topologie du réseau peut subir des changements dues aux :

- changement de position des nœuds
- accessibilité à cause du brouillage ou des obstacles en mouvements
- épuisement d'énergie
- mal fonctionnement des nœuds ou
- des besoins pour leur application.

Conclusion

Les réseaux sans fil ont connu un grand essor grâce à leur facilité d'utilisation ainsi que leur coût bas. Concentrés au départ sur la téléphonie mobile, leurs domaines d'application se sont multipliés au fil des années.

Dans ce chapitre nous avons vu les concepts généraux liés aux réseaux de capteurs sans fil. Parmi ces concepts nous avons identifié les contraintes de conception des réseaux de capteurs sans fil. En raison de ces contraintes, la manipulation d'un réseau de capteur sans fil dans un environnement réel impose des problèmes. C'est pour cela qu'il est nécessaire de passer par une phase de test avant de le mettre en place et la solution la plus fiable et la moins couteuse consiste en « La simulation ».

Dans le chapitre suivant, nous présenterons les concepts de base liés à la simulation des réseaux. Nous parlerons brièvement des simulateurs de réseaux existants ainsi nous présenterons le simulateur choisi pour nos simulations « OMNeT++ ».

Introduction :

Dans ce chapitre, nous traitons les méthodes et les techniques pour simuler les opérations des systèmes de réseaux informatiques et des applications réseau dans l'environnement réel. **La simulation** est l'une des techniques les plus utilisées dans la conception et la gestion des réseaux pour prédire la performance d'un système réseau ou d'une application réseau avant que le réseau ne soit construit physiquement ou que l'application soit déployée.

I. Simulation des réseaux :

1. Définition :

La simulation des réseaux est une technique par laquelle un logiciel (simulateur) modélise le comportement d'un réseau, soit par le calcul de l'interaction entre les entités du réseau en utilisant des formules mathématiques, ou en capturant et reproduisant des observations à partir d'un réseau réel [32] [18].

2. Objectifs de la simulation des réseaux :

Pour montrer l'utilité et l'objectif de la simulation des réseaux, nous avons présenté une comparaison entre un réseau réel et une simulation de réseau qui est décrit dans le tableau ci-dessous [8].

Un réseau réel	**Une simulation de réseau**
○ Le coût de tout le matériel, les serveurs, les commutateurs, etc. doit être pris en charge. ○ Il faut beaucoup de temps pour mettre en place de grands réseaux spécialisés utilisés pour les entreprises ou le milieu universitaire. ○ L'apport de modifications à un réseau préexistant nécessite une planification, et si une modification est effectuée par erreur, cela pourrait provoquer l'échec du réseau. ○ Obtenir la vraie chose, alors ce que nous observons du réseau réel est en train de se produire.	○ Le coût d'une seule machine indépendante avec un simulateur réseau installé (qui est gratuit). ○ Il faut du temps pour apprendre à créer des simulations, mais une fois que vous savez comment cela se fait, il est beaucoup plus facile de créer de nouvelles. ○ L'apport de modifications à un réseau simulé d'un véritable réseau préexistant ne pose aucun risque. Le résultat de la simulation peut être analysé pour déterminer comment le réseau réel sera affecté. ○ S'il y a un bug dans le logiciel de simulation, cela pourrait provoquer une simulation incorrecte.

Tableau II.1 : Comparaison entre un réseau réel et une simulation de réseau.

Comme nous pouvons le constater, il existe des avantages à utiliser à la fois des réseaux réels et des simulations de réseau lors de la création et de la vérification d'un réseau. Le point que nous voulons transmettre, c'est que les simulations de réseau peuvent rendre la conception de réseau moins chère et moins coûteuse.

3. Domaines d'application de la simulation :

La simulation nous permet de prévoir des problèmes potentiels sans d'abord vivre ces problèmes. Les exemples d'application sont très variés, citons par exemple [8] :

- **Production :**
  - Montrer comment fonctionne la gestion du travail, comme l'efficacité du travailleur, et comment les rotas et divers autres facteurs affecteront la production.
  - Pour montrer ce qui se passe lorsqu'un composant échoue sur une ligne de production.
- **Gestion de la foule:**
  - Pour montrer la longueur des files d'attente dans les parcs à thème et comment cela affectera les entreprises.
  - Pour montrer comment les gens vont s'asseoir assis lors d'un événement dans un stade.
- **Aéroports:**
  - Montrer les effets des retards de vol sur le contrôle du trafic aérien.
  - Montrer combien de sacs peuvent être traités à tout moment sur un système de manutention des bagages, et ce qui se produit quand il échoue.
- **Prévisions météorologiques:**
  - Prévoir les conditions météorologiques à venir.
  - Prévoir l'effet du changement climatique sur la météo.

4. Modèles de simulation :

Les modèles de simulation peuvent être classés de plusieurs façons. Les classifications les plus courantes sont les suivantes [27]:

4.1. Modèles de simulation statiques et dynamiques:

- **Modèle statique** caractérise un système indépendamment du temps.
- **Modèle dynamique** représente un système qui change avec le temps.

4.2. Modèles stochastiques et déterministes:

Si un modèle représente un système qui comprend des éléments aléatoires, il s'appelle un modèle **stochastique**. Sinon, c'est **déterministe**, il peut être impossible de

l'obtenir analytiquement (par exemple, un système compliqué d'équations différentielles). Les systèmes de mise en file d'attente, les systèmes sous-jacents des modèles de réseau, contiennent des composants aléatoires, tels que l'heure d'arrivée des paquets dans une file d'attente, le temps de service des files d'attente de paquets, la sortie d'un port de commutation, etc. [27].

4.3. Modèles discrets et continus:

- **Modèle continu** représente un système avec des variables d'état en constante évolution au fil du temps. Les exemples sont des équations différentielles qui indiquent les relations pour l'ampleur du changement de certaines variables d'état en fonction du changement de temps.
- **Modèle discret** caractérise un système où les variables d'état changent instantanément à des points discrets dans le temps. À ces moments discrets, un événement ou des événements peuvent se produire, en modifiant l'état du système. Par exemple, l'arrivée d'un paquet sur un routeur à un certain moment est un événement qui modifie l'état du tampon de port dans le routeur [27].

5. Types de simulations dans les RCSF :

Voici trois types de simulation:

- Simulation de Monte Carlo.
- Simulation par traçage.
- Simulation d'événement discret.

Les simulateurs de réseau de capteurs sans fil utilisent simulation par traçage ainsi que la simulation d'événement discret.

5.1. Simulation de Monte Carlo :

La simulation de Monte Carlo est une technique mathématique informatisée qui permet de tenir compte du risque dans l'analyse quantitative et la prise de décision.

La simulation Monte Carlo présente au responsable de la décision une plage d'issues possibles et leurs probabilités de réalisation suivant l'action choisie. Elle révèle les

possibilités extrêmes (les issues des décisions les plus audacieuses et les plus prudentes), ainsi que toutes les conséquences possibles des décisions intermédiaires [31].

5.2. Simulation axée sur la traçabilité:

La simulation par traçage joue surtout un rôle crucial dans les applications en temps réel. Elle fournit des informations qui permettent aux utilisateurs d'avoir des détails sur le modèle de simulation. Mais la simulation côté trajectoire comporte plusieurs inconvénients: parfois, des détails augmentent la complexité de la simulation [13].

5.3. Simulation d'événement discret:

La simulation d'événement discret est principalement utilisée dans le réseau de capteurs sans fil en raison de sa facilité dans la simulation de divers travaux fonctionnant sur différents nœuds de capteurs.

La simulation d'événement discret répertorie les événements en attente qui peuvent être simulés par des routines. Les variables globales répertorient le temps de simulation qui permet au planificateur de prévoir le temps d'avance. La simulation comprend diverses routines telles que l'entrée, la sortie ainsi que la trace. La simulation fournit la fonctionnalité de la gestion de la mémoire dynamique qui facilite l'ajout et la suppression de diverses entités dans la simulation ainsi que l'installation du débogueur pour vérifier le code côte à côte sans interrompre le fonctionnement normal de la simulation [13].

6. Simulations terminales et non terminales :

Dans le cadre de la mise en place de l'expérience de simulation, il faut choisir le type de simulation à exécuter. Les simulations se distinguent généralement par l'un des deux types: terminale ou non terminale.

6.1. Simulations terminales :

Une simulation terminale est celle dans laquelle la simulation commence à un état ou à un temps défini et se termine quand elle atteint un autre état ou un temps défini.

Les simulations terminales ne sont pas destinées à mesurer le comportement à l'état d'équilibre d'un système [28].

6.2. Simulations à l'état d'équilibre (simulations non terminales) :

Une simulation non terminale ou stable est celle dans laquelle le comportement en régime permanent du système est analysé. Elle ne signifie pas que la simulation ne se termine jamais, et cela ne signifie pas non plus que le système à simuler n'a pas de terminaison éventuelle. Cela signifie seulement que la simulation pourrait théoriquement continuer indéfiniment sans modification statistique du comportement. Pour les simulations non terminales, le modélisateur doit déterminer une durée appropriée pour exécuter le modèle [28].

7. Différentes étapes d'une simulation :

- Formuler un problème et planifier l'étude.
- Collecte des données et définissez le modèle.
- Le modèle conceptuel est-il valide?
- Construire un programme informatique et vérifier.
- Faire des essais pilotes.
- Le modèle programmé est-il valide?
- Expériences de conception.
- Faire fonctionner la production.
- Analyser les données de sortie.
- Document, présent et résultats d'utilisation [29].

8. Avantages, inconvénients et pièges de la simulation :

➢ **Avantage :**

- La simulation est souvent le seul type possible d'enquête.
- Permet d'estimer le comportement d'un système existant dans un ensemble projeté de conditions de fonctionnement.
- Les autres modèles de systèmes proposés peuvent être comparés.
- Le contrôle des conditions expérimentales peut être maintenu.
- Permet d'étudier des systèmes avec un temps long dans le temps compressé ou d'étudier le fonctionnement détaillé en temps déployé [29].

➢ **Inconvénients :**

- Chaque modèle de simulation stochastique ne produit que des estimations des caractéristiques réelles d'un modèle.
- Les modèles de simulation sont souvent coûteux et nécessitent beaucoup de temps pour développer parfois des temps d'exécution excessifs.
- Le grand volume de données de sortie ou l'animation réaliste crée souvent une tendance à accroître la confiance dans le résultat d'une étude que justifié [29].

➢ **Piège :**

- Objectifs non bien définis au début.
- Niveau de détail inapproprié (abstraction).
- Échec de la communication avec la direction tout au long de l'étude.
- Le traiter comme un exercice de programmation.
- N'ayant aucune personne ayant des connaissances sur la simulation et les statistiques.
- Défaut de tenir compte du caractère aléatoire du modèle.
- Échec de collecte de données.
- Logiciel de simulation inapproprié.
- Logiciel de simulation avec des fonctionnalités non documentées / peu claires.
- Mauvaise utilisation de l'animation.
- Faire une réplication simple d'une conception et d'un traitement de système particuliers.
- Les statistiques de sortie en tant que "vraie réponse".
- En utilisant de mauvaises mesures [29].

9. Différents simulateurs des RCSF :

La simulation est un outil essentiel pour étudier les réseaux de capteurs sans fil en raison de la non-conviction de l'analyse et des difficultés de mise en place d'expériences réelles. Cette étude fournit des lignes directrices pour aider à sélectionner un modèle de simulation approprié pour un RCSF et une description des outils disponibles les plus utilisés.

L'objectif est de fournir une étude sur de différents simulateurs de réseau de capteurs et présenter les avantages et les inconvénients de chaque simulateur et qui permet aux utilisateurs de sélectionner celui qui convient le mieux à leurs tests [16] [25].

9.1. NS-2 :

NS-2 est considéré comme un outil de simulation d'événement discret et a fait ses preuves dans la recherche de réseaux de communication dynamiques. NS-2 a été développé en 1989 et, depuis sa création, diverses contributions ont été réalisées et ont apporté diverses révolutions dans le domaine de la recherche en réseau. Il est totalement basé sur la programmation orientée objet. Il se compose de deux langues: langage de commande d'outil orienté objet **C ++** et objet **OTcl**. C ++ est principalement utilisé pour la mise en œuvre de divers protocoles et l'extension des bibliothèques de simulation, tandis que les scripts **OTcl** permettent de configurer le simulateur, le paramétrage de la topologie du réseau, la création de scénarios de réseau et l'affichage des résultats de simulation [16].

La figure suivante montre l'architecture basique de simulateur NS-2.

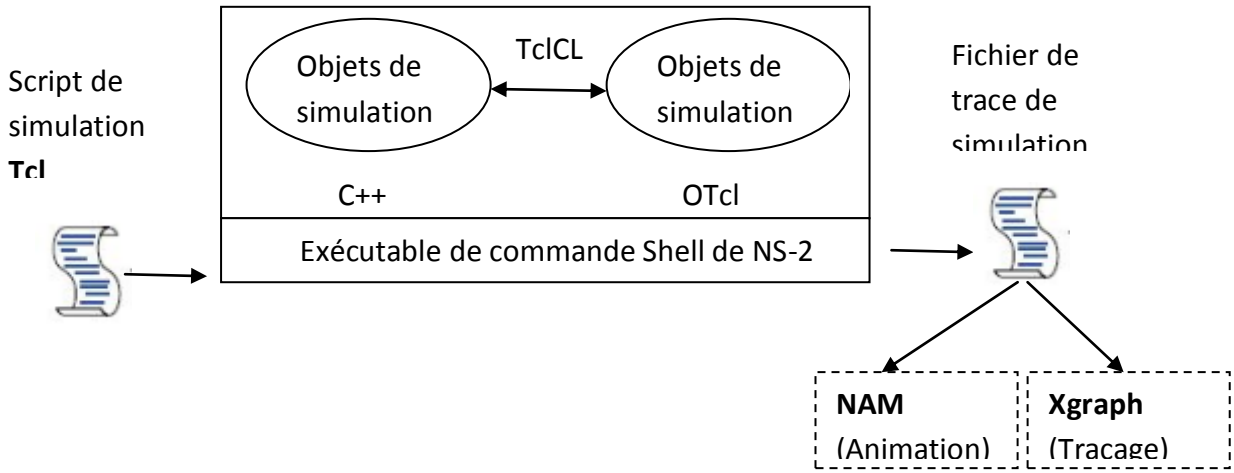


Figure II.1 : Architecture de base du simulateur NS-2.

9.2. OMNeT ++:

OMNet ++ est un simulateur de réseau d'événements discrets orienté objet pour les réseaux de capteurs sans fil. Il est considéré comme un framework de bibliothèque de simulation C ++ modulaire et à base de composants pour la construction de simulations de réseaux sans fil. Fondamentalement, OMNeT ++ n'est pas un simulateur, mais il fournit des cadres et des outils pour les scénarios de simulation d'écriture. Il est compatible pour fonctionner sur différents systèmes d'exploitation comme Windows, Linux et MAC OS X.

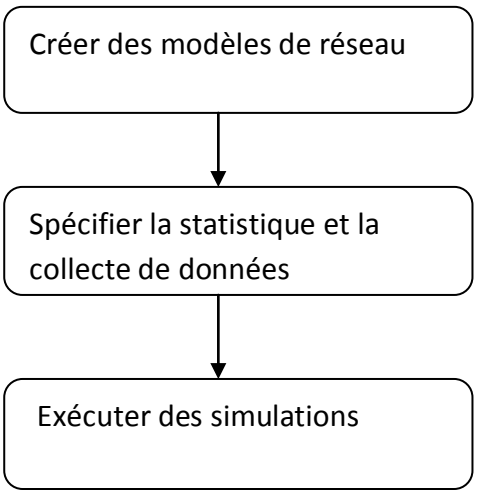
Depuis la version 4.6 qui a été lancée en juillet 2014 principalement axée sur l'amélioration de l'utilisation de l'environnement graphique **Runtime** (**TKenv**). **TKenv GUI** est

réorganisé pour un mode fenêtre unique pour améliorer l'expérience utilisateur et l'utilisation du simulateur. Le rôle principal de **TKenv** est de montrer l'animation du modèle de réseau, le mouvement des nœuds, l'affichage des résultats, les changements de visualisation, etc. OMNeT ++ est gratuit pour téléchargement à des fins d'éducation et de recherche. Il comprend un environnement **IDE** basé sur **eclipse** qui permet la programmation C ++ et le débogage des modules ainsi que l'édition graphique et textuelle des fichiers **NED** [26].

9.3. OPNET Modeler Suite :

OPNET est un simulateur d'événements discrets commercial et est disponible en 32 bits ainsi qu'en 64 bits pour Windows et Linux. OPNET utilise la technologie orientée objet pour créer une cartographie allant de la conception graphique à la mise en œuvre du système réel. Il modélise le réseau dans une approche hiérarchique qui correspond quelque peu à la structure hiérarchique d'Internet tels que les réseaux, satellites et nœuds. OPNET prend essentiellement en charge trois types de liens: point à point, bus et sans fil. Le lien sans fil est essentiellement utilisé dans le réseau sans fil, mobile ou par satellite. OPNET dispose de diverses bibliothèques pour les réseaux sans fil et filaires, il prend également en charge les réseaux de capteurs sans fil. Il fournit aussi différents éditeurs pour faciliter une simulation de réseau.

Afin de créer un réseau correctement dans OPNET, il faut suivre un processus de travail, à partir de la création des modèles réseau, d'autre part, choisir les données et les statistiques à collecter, puis exécuter la simulation, enfin, afficher et analyser les résultats. La figure ci-dessous illustre le flux de travail [16].



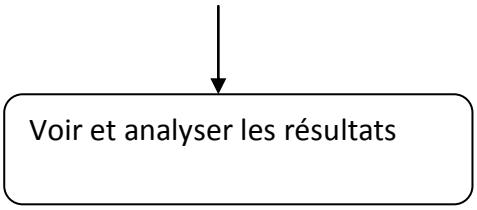


Figure II.2 : Flux de travail de simulation dans OPNET.

10. Comparaison entre ces différents simulateurs :

La simulation est souvent moins chère que l'expérimentation et comporte beaucoup moins de risques lorsque l'homme fait partie du système étudié. La simulation est basée sur une connaissance des phénomènes qui ne peut être obtenue que par l'expérimentation.

Une simulation ne peut donc être réalisé que si nous disposons d'un acquis de connaissances suffisant obtenues par des expérimentations sur des phénomènes antérieurs et analogues. Quelle que soit la qualité de la simulation, elle ne remplace pas totalement l'expérimentation. Voici un tableau général décrivant les avantages et les inconvénients des simulateurs décrit auparavant [16].

Simulateur	**Avantage**	**Inconvénient**
NS-2	○ Non commercial. ○ Permet la simulation des comportements des protocoles standards. ○ Un grand nombre de protocoles disponibles publiquement. ○ Simulateur multicouches.	○ Difficulté d'ajout de nouveau modèle à cause des dépendances entre modules. ○ Intégration difficile à d'autres applications. ○ Faible performance des simulations de réseaux importants.
OMNeT++	○ Non commercial. ○ Architecture modulaire	○ Le nombre de protocoles n'est pas assez grand.

	permettant l'intégration de nouveaux modèles. o Puissante interface utilisateur graphique (rendant le suivi et le débogage plus facile). o Conception de modèles se rapprochant de la réalité.	o Problème de compatibilité (non portable).
OPNET Modeler Suite	o Possède de nombreux modèles de protocole. o Bonne performance du débogage et du traçage.	o Commercial. o Ne possède pas un environnement d'exécution graphique.

Tableau II.2 : Comparaison entre différents simulateurs des réseaux de capteurs sans fil.

11. Choix du simulateur :

Le déploiement d'un réseau de capteur exige une étape de simulation avant son installation sur site. La simulation permet de tester à moindre coût les performances d'une solution.

OMNeT++ est un environnement de simulation à évènements discrets basé sur le langage C++, une application open source. Il est totalement programmable, paramétrable et modulaire ainsi grâce à son architecture flexible et générique, il a été utilisé avec succès dans divers domaines.

OMNeT++ sera notre environnement de simulation, grâce à son architecture modulaire et sa conception de modèles se rapprochant de la réalité.

II. Présentation d'OMNeT++ :

1. Définition :

OMNeT++ est un logiciel libre destiné à la simulation d'événements discrets écrit en langage objet C++. Il a été developpée par Andras Varga, chercheur à l'université de Budapest. Il a été conçu pour la simulation des réseaux informatiques, des systèmes multiprocesseurs et d'autres systèmes répartis. Il est utilisé par de nombreux groupes de recherche pour l'évaluation et l'estimation des performances d'un réseau de télécommunication. Il est utile pour l'illustration, la correction, l'évaluation et l'amélioration des performances. Il fournit un ensemble d'outils qui aident à modifier ou à mettre en application de nouvelles caractéristiques d'une manière simple.

Plusieurs exemples de modèles peuvent être simulés avec OMNeT++ ; nous pouvons citer :

- Algorithmes de routage.
- Réseaux informatiques et Modèles de trafic : files d'attente, etc.
- Multiprocesseur et systèmes répartis.

2. Installation d'OMNeT++ :

- Télécharger le code source OMNeT ++ à partir de **http://omnetpp.org**. Assurer de sélectionner l'archive spécifique à Windows, nommée **omnetpp-4.6-src-windows.zip**.
- Le paquet est presque autonome: en plus des fichiers OMNeT ++, il comprend un compilateur C ++, un environnement de génération de ligne de commande et toutes les bibliothèques et programmes requis par OMNeT ++.
- Copier l'archive OMNeT ++ dans le répertoire où nous souhaitons l'installer, choisir un répertoire dont le chemin complet ne contient aucun espace; Par exemple, ne pas mettre OMNeT ++ sous Program Files.
- Extraire le fichier téléchargé, un répertoire nommé omnetpp-4.6 est crée [20].

2.1. Configuration et construction d'OMNeT ++ :

Le processus de compilation et d'installation sur Windows est très autonome. Tout d'abord, il faut vérifier le contenu du fichier **configure.user** pour s'assurer qu'il contient les paramètres dont nous avons besoin. Dans la plupart des cas, nous n'avons pas besoin de changer quoi que ce soit.

Pour compiler et installer, il faut suivre les étapes ci-dessous:

- Entrer dans le répertoire OMNeT ++ décompressé plus tôt.
- Exécuter le fichier appelé **Mingwenv.cmd**.
- Lorsque le terminal apparaît, entrer les deux commandes suivantes en série:

. /configure

make

- Enfin, pour assurer que l'installation s'est bien déroulée, nous pouvons vérifier l'exécution de l'un des exemples intégrées dans le répertoire **samples**.

L'exemple **dyna** est démarré en entrant les commandes suivantes:

$ cd samples / dyna

$./dyna

Tous les fichiers échantillons seront exécutés, l'environnement graphique **GUI Tkenv** apparaît.

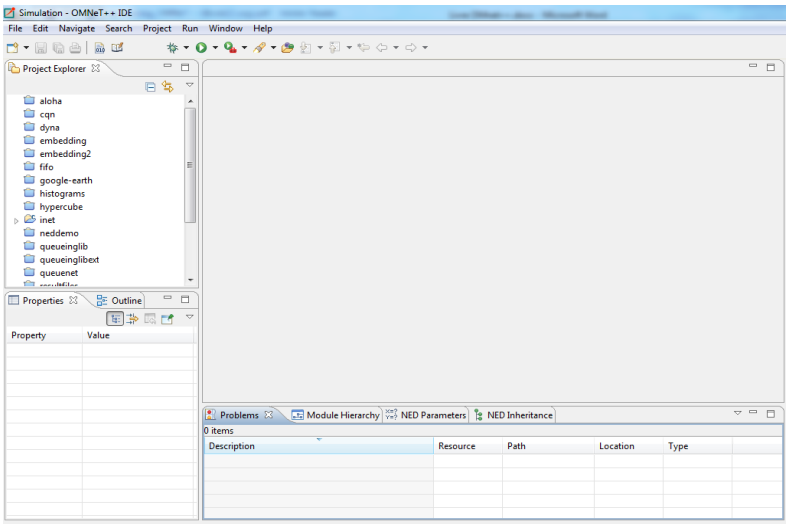


Figure II.3 : Environnement graphique GUI Tkenv.

2.2. Exécution d'OMNeT ++ IDE :

Maintenant qu'OMNeT ++ est installé, nous pouvons l'exécuter de deux manières différentes :

- Dans le dossier ide de dossier OMNeT ++, nous trouvons une application appelée omnetpp; qui est une forme courte pour OMNeT ++. Il est recommandé de créer un raccourci pour omnetpp sur le bureau.
- Nous pouvons également ouvrir l'IDE OMNeT++ en exécutant **Mingwenv.cmd** en tapant la commande suivante:

$ omnetpp

Si l'écran de démarrage suivant apparaît, cela signifie que l'IDE OMNeT ++ s'exécute correctement:

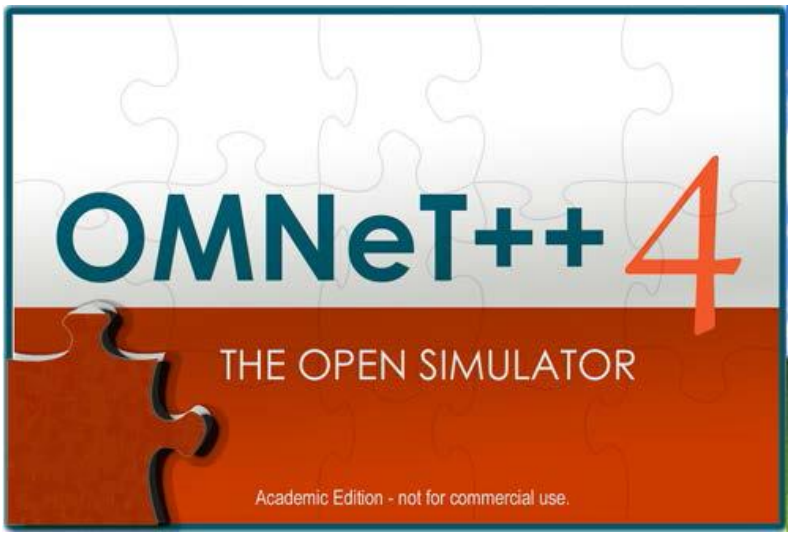


Figure II.4 : Ecran de démarrage d'OMNeT++.

Une fois qu'OMNeT ++ est exécuté, nous pouvons sélectionner l'espace de travail par défaut.

Un espace de travail est une collection logique de projets. Sur notre installation, nous avons créé notre propre dossier d'espace de travail que nous avons appelé Projets qui ressemble à la capture d'écran suivante:

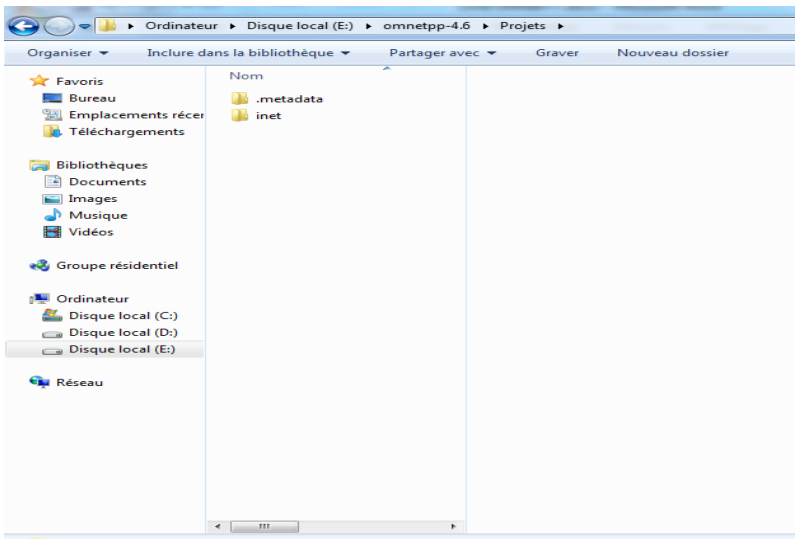


Figure II.5 : L'espace de travail d'OMNeT++.

3. Caractéristiques d'OMNET ++:

- C'est un noyau de simulation.
- Un éditeur de réseau graphique pour les fichiers NED.
- Outils pour tracer des données.
- Compiler pour les langages de description de topologie NED.
- Deux types d'interfaces utilisateur pour l'exécution de la simulation: une interface utilisateur de ligne de commande et une interface utilisateur graphique [21].

3.1. Langage de description de réseau (NED) :

Un réseau est formé de "nœuds" reliés par des "liens". Les nœuds représentent les blocs (modules simples ou composés), tandis que les liens représentent les canaux, raccordements, etc.

La façon dont les éléments fixes (c.-à-d. nœuds) sont reliés ensemble dans un réseau s'appelle la topologie.

OMNeT++ utilise le langage NED qui facilite la création, l'édition, et la représentation graphique d'un certain réseau à simuler. Il suit une structure hiérarchique tenant compte de différents niveaux d'organisation. Un nœud par exemple peut être structuré selon les

différentes couches OSI dont il se compose : couche physique, liaison de données, réseau, transport, application, etc. De même, chaque couche peut être structurée selon les applications ou protocoles qui tournent au niveau de cette couche [21].

Les fichiers de description de réseau ont un suffixe **«.ned».** Le compilateur NEDC traduit les descriptions de réseau en code C++. Puis, ces descriptions seront compilées par le compilateur C++ et incorporées dans la simulation exécutable. Donc une description NED peut contenir les composants suivants, d'une façon et en nombre arbitraires:

- Les déclarations d'Importation.
- Définitions de Canaux.
- Modules Simples et Composés.

3.2. Interfaces utilisateur :

L'interface utilisateur d'OMNeT++ concerne l'exécution de simulation. Elle permet à l'utilisateur de lancer et terminer des simulations et de changer des variables à l'intérieur des modèles de simulation. L'utilisateur peut examiner et corriger la simulation avec une interface graphique puissante [14].

Actuellement, deux interfaces utilisateur sont soutenues :

3.2.1. Interface utilisateur graphique (Tkenv):

Tkenv est une interface graphique qui permet le « traçage », la correction et l'exécution de simulation. Elle a la capacité de fournir une image détaillée de l'état de la simulation à un point quelconque pendant l'exécution [14] [11].

- **Caractéristiques de Tkenv:**

  - Fenêtre séparée pour chaque module.
  - Les messages programmés peuvent être observés dans une fenêtre pendant que la simulation progresse.
  - Exécution d'événement-par-événement.
  - Animation de l'exécution.
  - Fenêtres pour examiner et changer des objets et des variables dans le modèle.
  - Affichage graphique des résultats de simulation pendant l'exécution.

- La simulation peut être remise en marche.

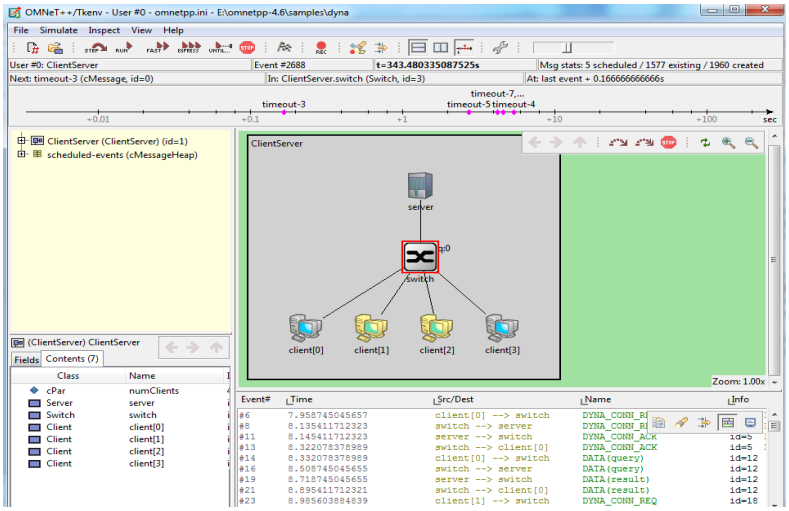


Figure II.6 : Tkenv avec OMNeT++.

3.2.2. Interface utilisateur de ligne de commande (Cmdenv) :

Cmdenv conçue principalement pour l'exécution en batch. C'est une interface par ligne de commande portative, petite et rapide. Cmdenv exécute simplement toutes les simulations **« runs »** qui sont décrites dans le fichier de configuration [14].

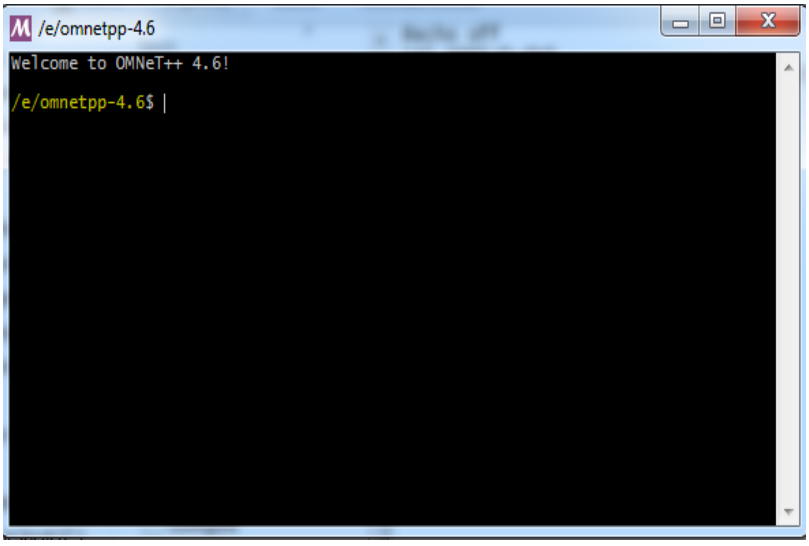


Figure II.7 : Cmdenv avec OMNeT++.

4. Architecture d'OMNeT++:

Un modèle OMNeT ++ se compose de modules qui communiquent avec le passage des messages. Les modules actifs sont appelés modules simples; ils sont écrits en C ++, en utilisant la bibliothèque de classes de simulation. Les modules simples peuvent être regroupés en modules composés et ainsi de suite; le nombre de niveaux de hiérarchie est illimité. Le modèle entier, appelé réseau dans OMNeT ++, est lui-même un module composé. Les messages peuvent être envoyés via des connexions qui couvrent des modules ou directement à d'autres modules [21] [11].

Dans la **figure II.8**, les boîtes représentent des modules simples (fond gris) et des modules composés. Les flèches reliant les petites boîtes représentent les connexions et les portes.

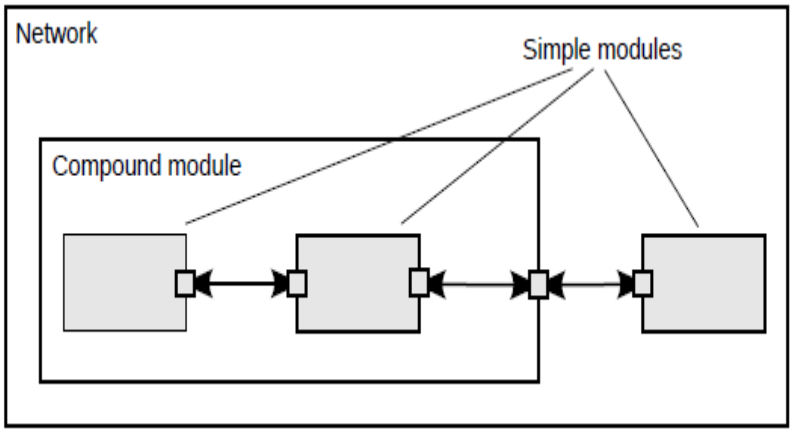


Figure II.8 : Modules simples et modules composés.

Les modules communiquent avec des messages qui peuvent contenir des données arbitraires. Les modules simples envoient généralement des messages via des portes, mais il est également possible de les envoyer directement à leurs modules de destination. Les portes sont les interfaces d'entrée et de sortie des modules: les messages sont envoyés par des portes de sortie et arrivent par des portes d'entrée. Une porte d'entrée et une porte de sortie peuvent être reliées par une connexion. Les connexions sont créées dans un seul niveau de hiérarchie de module.

À l'intérieur d'un module composé, des grilles correspondantes de deux sous-modules ou une grille d'un sous-module et une grille du module composé peuvent être reliées. Les connexions couvrant des niveaux de hiérarchie ne sont pas autorisées. En raison

de la structure hiérarchique du modèle, les messages traversent typiquement une chaîne de connexions, commençant et arrivant dans des modules simples.

OMNeT ++ fournit des outils efficaces permettant à l'utilisateur de décrire la structure du système actuel.

Certaines des caractéristiques principales sont les suivantes:

- Modules hiérarchiquement imbriqués.
- Modules sont des exemples de types de modules.
- Les modules communiquent avec des messages via des canaux.

4.1. Modules hiérarchiques :

Un modèle OMNeT ++ consiste en des modules hiérarchiquement imbriqués qui communiquent en transmettant des messages les uns aux autres. Les modèles OMNeT ++ sont souvent appelés réseaux. Le module de niveau supérieur est le module système. Le module système contient des sous-modules qui peuvent également contenir des sous-modules eux-mêmes (**figure II.8**). La profondeur de l'imbrication de module est illimitée, permettant à l'utilisateur de refléter la structure logique du système réel dans la structure du modèle [21].

La structure du modèle est décrite dans le langage NED d'OMNeT ++.
Les modules qui contiennent des sous-modules sont appelés modules composés, par opposition aux modules simples au niveau le plus bas de la hiérarchie des modules. Les modules simples contiennent les algorithmes du modèle. L'utilisateur implémente les modules simples en C ++, en utilisant la bibliothèque de classes de simulation OMNeT ++ .

4.2. Types de modules :

Les modules simples et composés sont des exemples de types de modules. En décrivant le modèle, l'utilisateur définit des types de modules. Tous les modules du réseau sont instanciés sous forme de sous-modules et sous-sous-modules du module système.

Lorsqu'un type de module est utilisé comme bloc de construction, il ne fait aucune différence qu'il s'agisse d'un module simple ou composé. Cela permet à l'utilisateur de diviser un module simple en plusieurs modules simples intégrés dans un module composé,

ou vice versa, pour agréger la fonctionnalité d'un module composé en un seul module simple, sans affecter les utilisateurs existants du type de module [21].

4.2.1. Module simple :

Un objet de module simple dans OMNeT ++ est un composant qui se trouve à l'intérieur d'un objet réseau et qui peut envoyer des messages à d'autres objets du module.

Dans une simulation, cela pourrait être un routeur, un serveur web, une machine autonome ou tout autre composant, capable de faire la communication sur un réseau [8].

Lorsque nous ajoutons un objet module à notre objet réseau **My_Network**, il ressemble à ce qui suit:

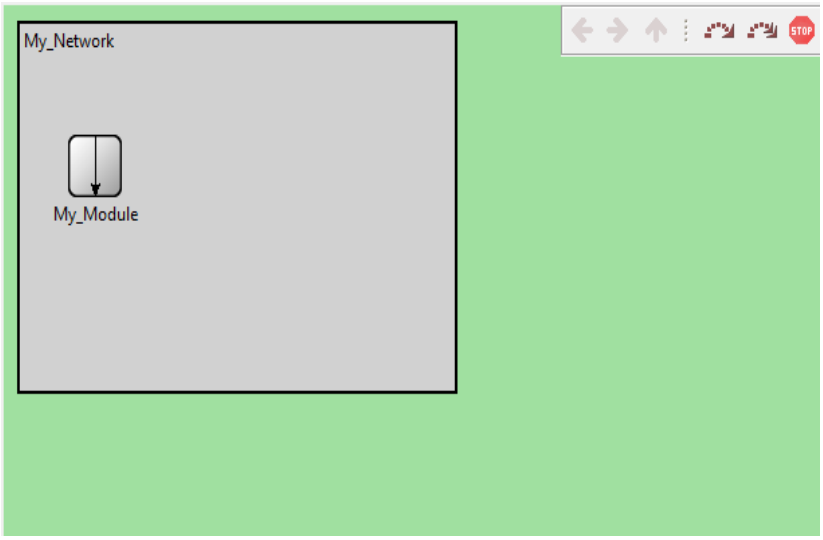


Figure II.9 : Exemple d'une simulation Module simple.

4.2.2. Modules composés :

Un objet module composé se compose de plusieurs objets module simple.

Voici ce qu'un module **StandardHost** composé ressemble à l'objet réseau **My_Network** :

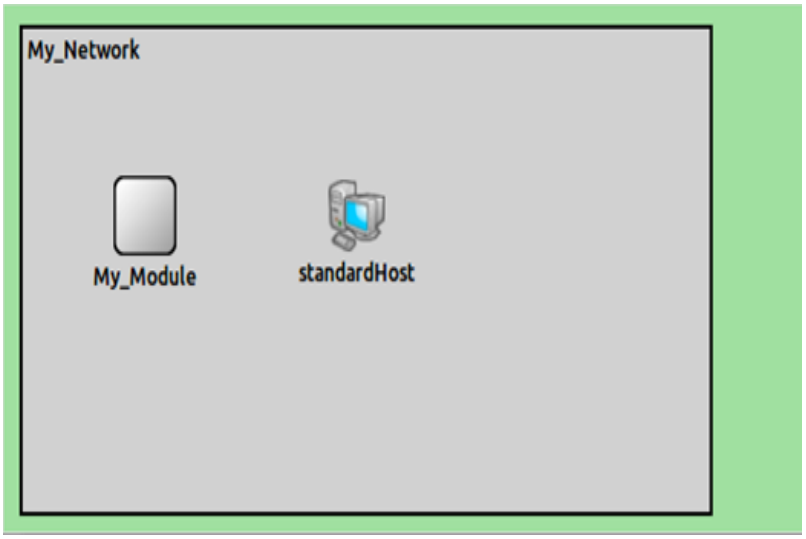


Figure II.10 : Exemple d'une simulation Module composé.

Voici maintenant ce que l'objet du module composé **StandardHost** ressemble sous le coffre:

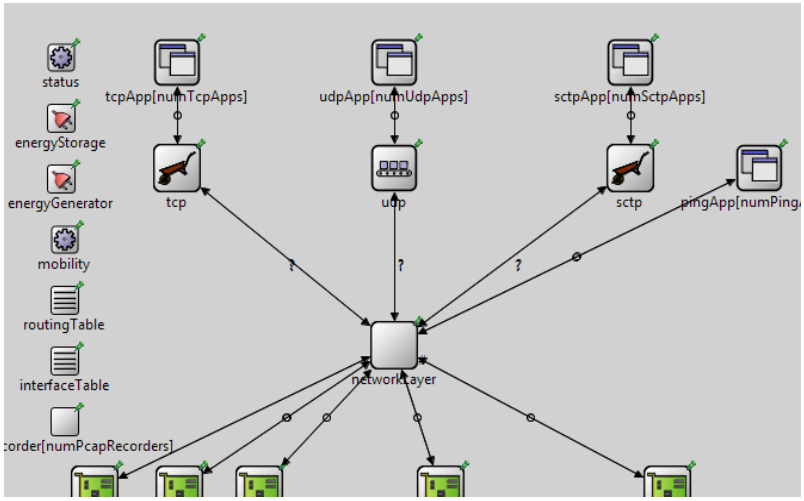


Figure II.11 : Module composé StandardHost.

4.2.3. Messages, portails, liens :

Les modules communiquent par échange de messages. Dans une simulation réelle, les messages peuvent représenter des trames ou des paquets dans un réseau informatique, des travaux ou des clients dans un réseau de mise en file d'attente ou d'autres types

d'entités mobiles. Les messages peuvent contenir des structures de données arbitrairement complexes.

Les modules simples peuvent envoyer des messages soit directement à leur destination soit le long d'un chemin prédéfini, via des portes et des connexions.

Cela pourrait être un **StandardHost** en parlant à un autre **StandardHost** via un câble **Ethernet** comme indiqué dans la **figure II.12**.

Le "temps de simulation local" d'un module avance lorsque le module reçoit un message. Le message peut provenir d'un autre module ou du même module [21].

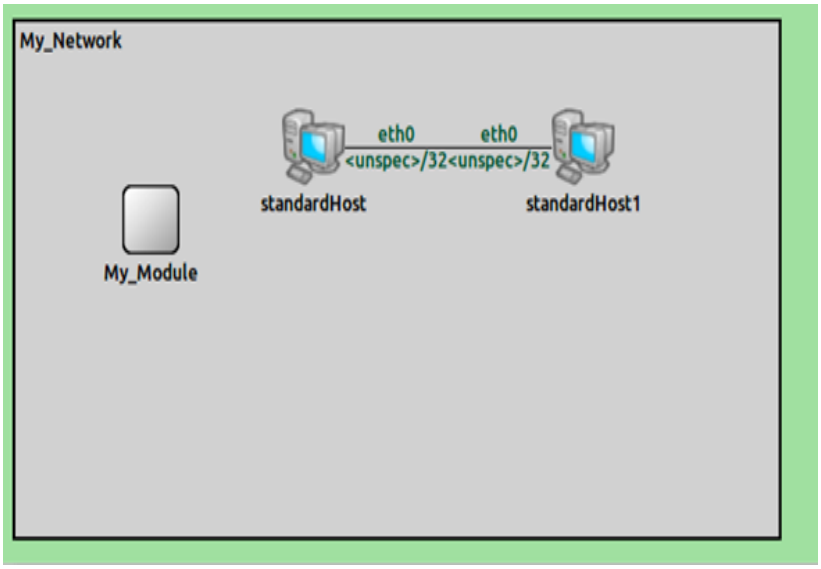


Figure II.12 : Communication entre deux StandardHost.

5. Les principaux fichiers d'OMNeT++ :

L'environnement de développement intégré OMNeT ++ est basé sur la plate-forme eclipse et l'étend avec de nouveaux éditeurs, des vues, des assistants et des fonctionnalités supplémentaires.

OMNeT ++ ajoute des fonctionnalités pour créer et configurer des modèles (fichiers **NED** et **INI**), exécuter des exécutions par lot et analyser les résultats de simulation, tandis que eclipse fournit une édition C ++ et d'autres fonctionnalités facultatives (modélisation UML, accès à la base de données, etc.) via divers plug-ins open source et commerciaux [13].

5.1. L'éditeur NED :

NED est le langage de description de topologie d'OMNeT ++. Il a une syntaxe simple, mais il est très puissant lorsqu'il s'agit de définir des topologies régulières telles que la chaîne, l'anneau, le maillage, l'hyper cube, les structures d'arbres, etc.

Lorsque l'IDE détecte des erreurs dans un fichier NED, le problème sera signalé avec un marqueur d'erreur dans l'explorateur de projets et la vue des problèmes sera mise à jour pour afficher la description et l'emplacement du problème. En outre, les marqueurs d'erreur apparaîtront dans la fenêtre de texte ou sur la représentation graphique du composant problématique. L'ouverture d'un fichier NED contenant une erreur ouvrira le fichier en mode texte. Passer en mode graphique n'est possible que si le fichier NED est syntaxiquement correct [13].

5.1.1. Fonctionnalités NED :

- Structure de module hiérarchique
  - Écrire le type de module composé une fois et créer plusieurs instances.
  - Profondeur de nidification non limitée.
- Description de la topologie flexible
  - Les paramètres peuvent définir: nombre de sous-modules, types de sous-modules, structure d'interconnexion.
  - Modèles de topologie: un moyen de réutiliser la structure d'interconnexion.
- Paramètres
  - Peuvent contenir des variables aléatoires (elles peuvent donc être utilisées comme source de nombres aléatoires).
  - Peut être passé par valeur ou par référence.
  - Peut contenir des expressions d'autres paramètres.
- Prend en charge le partitionnement flexible du modèle pour une exécution parallèle [21].

5.1.2. Création de nouveaux fichiers NED :

Une fois que nous avons un projet OMNeT ++ vide, nous pouvons créer de nouveaux fichiers NED. Pour se faire nous cliquons sur **« file |new| network description file (NED) »**. Le fichier du menu affichera un assistant où nous pouvons spécifier le répertoire cible et le

nom du fichier. Et aussi nous pouvons choisir de créer un fichier NED vide, un module simple / composé ou un réseau. Une fois qu'on appui sur le bouton **« finish »**, un nouveau fichier NED sera créé avec le contenu demandé.

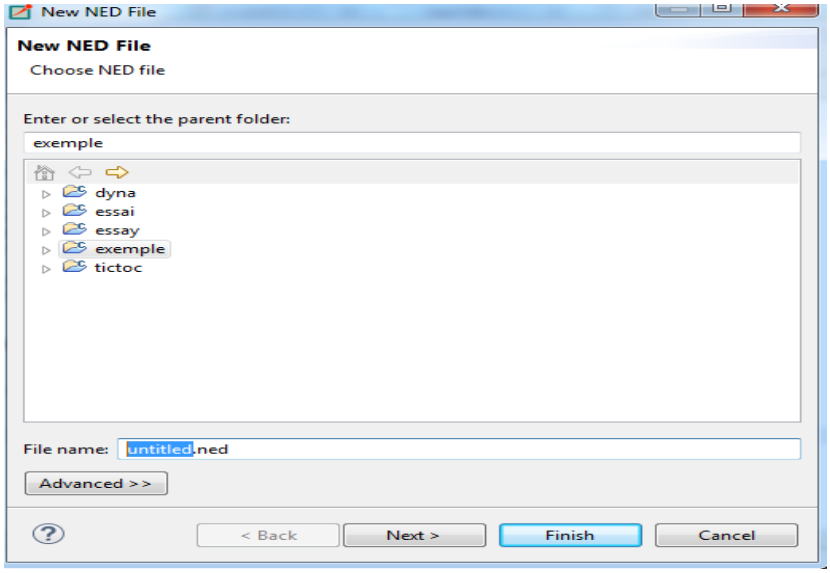


Figure II.13 : Création d'un nouveau fichier NED.

Remarque :

Il faut s'assurer que le fichier NED et le module contenu ont le même nom. Par exemple, un module composé nommé **Wireless42** devrait être défini dans son propre fichier **Wireless42.ned**.

5.1.3. Dossiers source NED :

Il est possible de spécifier quels dossiers de l'IDE doit rechercher les fichiers NED et que l'IDE utilisera comme base le répertoire de notre hiérarchie de paquets NED. Seuls les fichiers situés dans les dossiers source NED seront ouverts avec l'éditeur graphique. Si un fichier NED n'est pas dans les dossiers source NED, il sera ouvert dans un éditeur de texte standard. Pour spécifier le répertoire où les fichiers NED seront stockés, nous cliquons avec le bouton droit sur le projet dans « **Project Explorer|Properties|OMNeT ++|NED Source**

Folders », puis nous cliquons sur les dossiers où nous stockons nos fichiers NED. La valeur par défaut est la racine du projet [13].

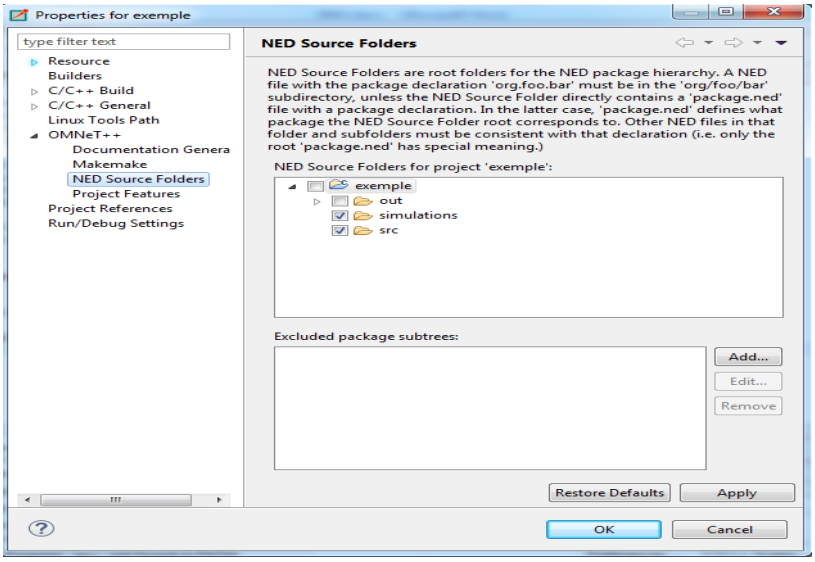


Figure II.14 : Dossiers source NED.

5.1.4. Utilisation de l'éditeur NED :

L'éditeur NED peut éditer les fichiers NED à la fois en mode graphique ou en mode texte, et l'utilisateur peut basculer entre les deux modes à tout moment en utilisant les onglets situés au bas de la fenêtre de l'éditeur.

a. En mode graphique :

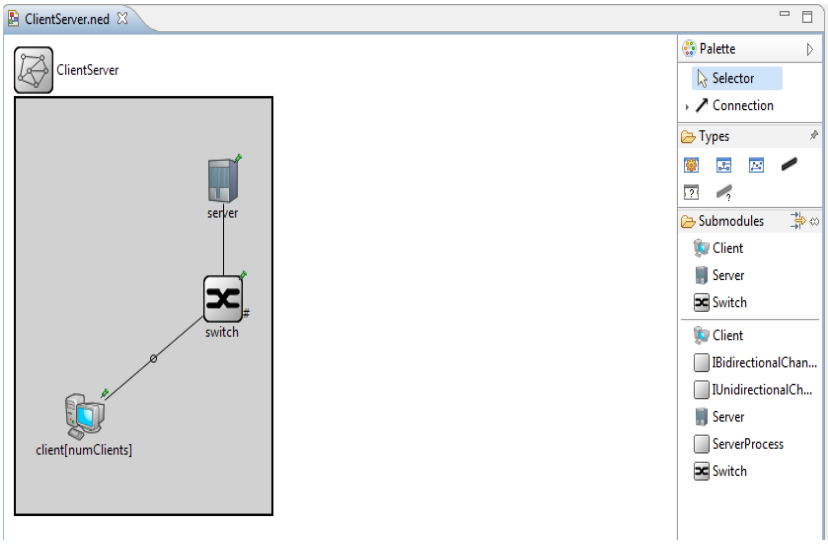


Figure II.15 : Éditeur NED en mode d'édition graphique.

En mode graphique, nous pouvons créer des modules composés, des canaux et d'autres types de composants. Les modules et les sous-modules simples sont représentés sous forme d'icônes tandis que les modules composés et les réseaux sont affichés sous forme de rectangles où d'autres sous-modules peuvent être abandonnés.

Les connexions entre les sous-modules sont représentées par des lignes ou des flèches selon que la connexion était unidirectionnelle ou bidirectionnelle. Les sous-modules peuvent être déplacés ou redimensionnés à l'aide de la souris et connectés en utilisant l'outil de connexion dans la palette. L'éditeur offre de nombreuses fonctionnalités telles que l'annulation, le clonage d'objets, le déplacement et le redimensionnement contraints, l'alignement des objets et le zoom [13].

Les sous-modules peuvent être épinglés (ayant une position fixe) ou non fixés (mise en page automatique). Les caractéristiques graphiques qui peuvent être modifiées sont l'image d'arrière-plan, la grille de fond, les icônes par défaut (via l'héritage des chaînes d'affichage), le dimensionnement et la coloration de l'icône, la plage de transmission et bien d'autres.

b. En mode texte :

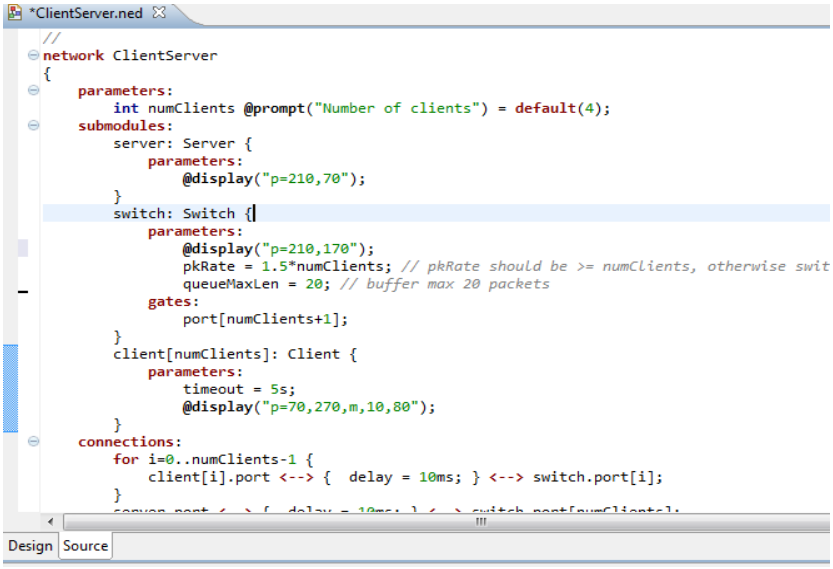


Figure II.16 : Éditeur NED en mode d'édition source.

Le mode texte permet à l'utilisateur de travailler directement avec la source NED. Lorsque nous appuyons sur **Ctrl + Espace**, l'éditeur offre une mise au point contextuelle des mots-clés et du type de module, des paramètres, des portes et des noms de sous-modules. Des propositions de modèles pour insérer des squelettes de modules composites complets, des sous-modules, diverses structures de connexion, etc, sont également disponibles. La documentation des types de modules référencés peut être visualisée en planant le nom de type NED. La source NED est continuellement analysée et validée à mesure que l'utilisateur tape, et les erreurs sont affichées en temps réel sur la marge de gauche.

5.1.5. Modification d'une propriété de module :

Pour modifier une propriété de module, nous cliquons dessus avec le bouton droit et nous sélectionnons l'élément **« Properties »** dans le menu contextuel, comme nous pouvons appuyer sur **Ctrl + Entrée** lorsque le module est sélectionné. Les propriétés NED telles que le nom, le type et la taille du vecteur sont disponibles sur l'onglet **« General ».** Les propriétés visuelles comme l'icône, la taille, la couleur, la bordure, etc, peuvent être définies dans l'onglet **« Appearance ».** Nous pouvons vérifier la façon dont notre module ressemblera dans le panneau d'aperçu en bas de la boîte de dialogue.

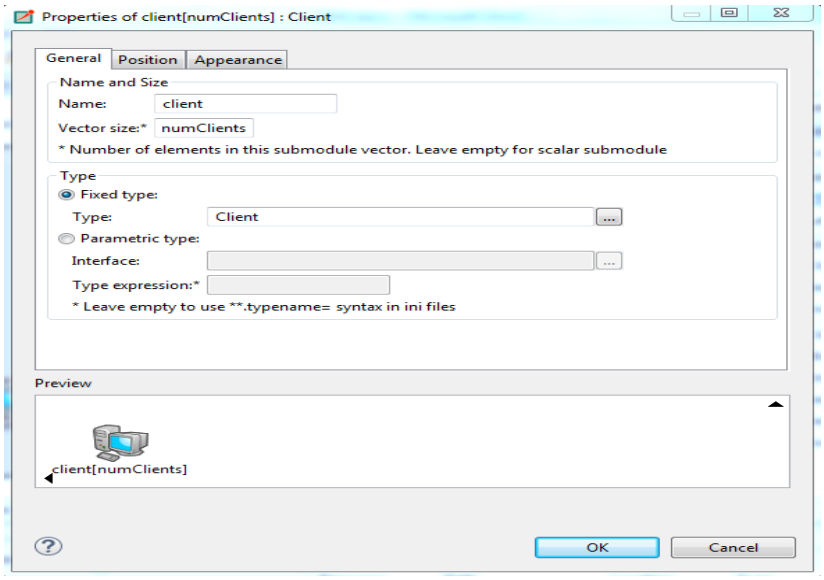


Figure II.17 : Modification des propriétés visuelles du module.

5.1.6. Modification d'un paramètre de module :

Pour modifier un paramètre de module, nous cliquons dessus avec le bouton droit et nous sélectionnons l'option **« Parameters »** dans le menu contextuel. La boîte de dialogue nous permet d'ajouter ou de supprimer des paramètres de module ou de leur attribuer une valeur.

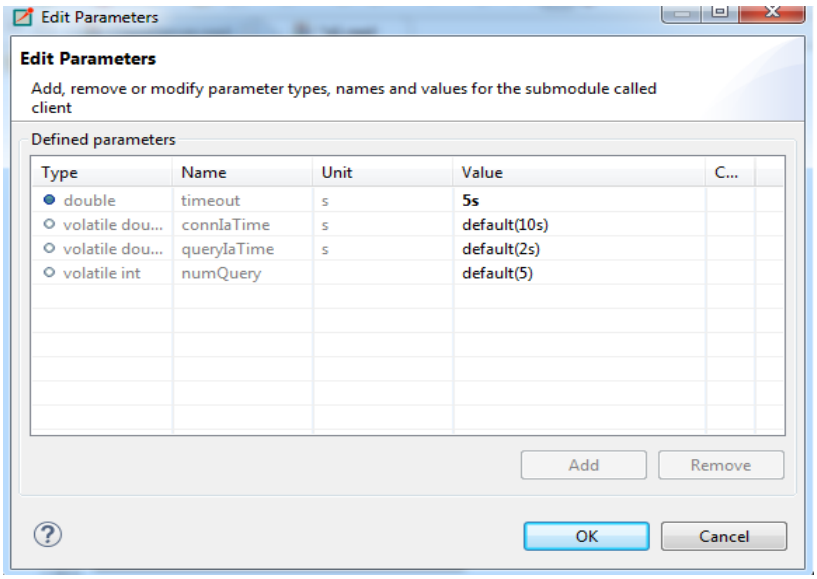


Figure II.18 : Modification des paramètres du module.

5.2. L'éditeur de fichiers INI :

Dans OMNeT ++, les modèles de simulation sont paramétrés et configurés pour l'exécution à l'aide de fichiers de configuration avec l'extension **.ini**. Les fichiers **INI** sont des fichiers texte, qui peuvent être édités à l'aide d'un éditeur de texte. Cependant, OMNeT ++ 4.x présente un outil expressément conçu pour l'édition de fichiers INI. L'éditeur de fichiers INI fait partie de l'IDE OMNeT ++ et est très efficace pour aider l'utilisateur à créer des fichiers INI. C'est une fonctionnalité très utile car elle possède une connaissance détaillée du modèle de simulation, de la syntaxe du fichier INI et des options de configuration disponibles.

L'éditeur de fichiers INI est un éditeur à double mode. La configuration peut être modifiée à l'aide de formulaires et de dialogues, ou en texte brut. Les formulaires sont organisés autour de sujets tels que la configuration générale, **Cmdenv**, **Tkenv**, les fichiers de sortie, les extensions, etc. L'éditeur de texte fournit la mise en surbrillance de syntaxe et l'achèvement automatique. Plusieurs vues peuvent afficher des informations, ce qui est utile lors de l'édition de fichiers INI. Par exemple, nous pouvons voir les erreurs dans le fichier INI actuel ou tous les paramètres du module disponibles dans une seule vue [21].

5.2.1. Création de fichiers INI :

Pour créer un nouveau fichier INI, nous choisissons **« file |new|Initialization File (ini) »** à partir du menu. Il ouvre un assistant où on peut entrer le nom du nouveau fichier et sélectionner le nom du réseau à configurer.

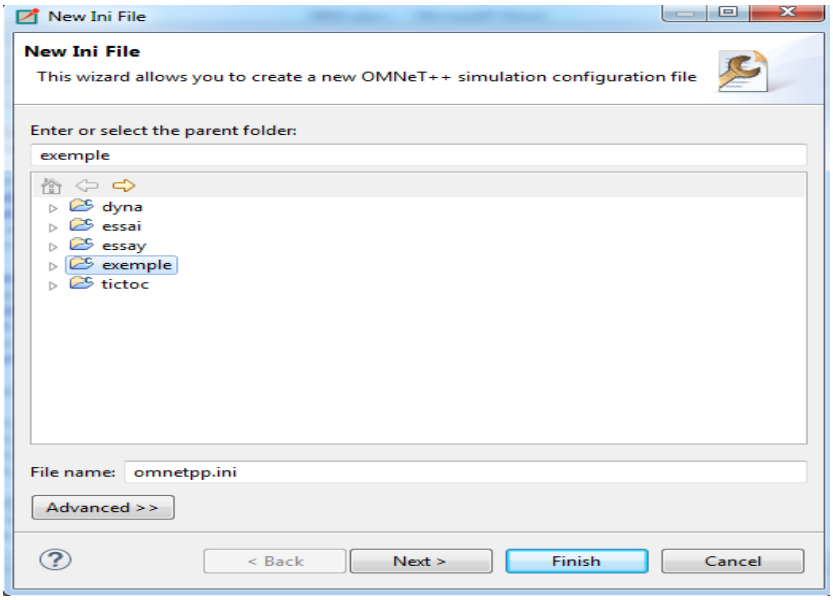


Figure II.19 : Création d'un nouveau fichier INI.

5.2.2. Utilisation de l'éditeur de fichiers INI :

L'éditeur de fichiers INI a deux modes : mode formulaire et mode texte (source), nous pouvons basculer entre les modes en sélectionnant les onglets en bas de l'éditeur.

a. En mode formulaire :

Le mode formulaire nous permet de modifier la configuration en entrant des valeurs comme le nom du réseau à simuler, le temps d'exécution, etc.

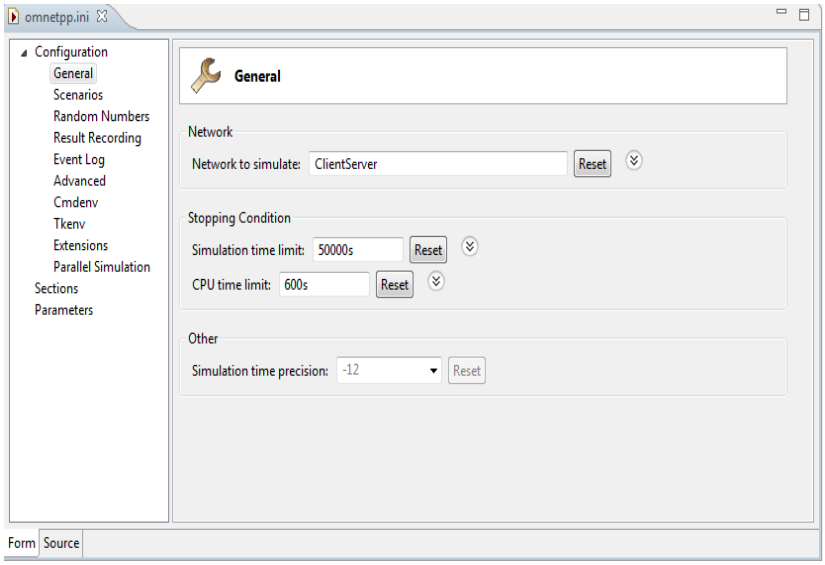


Figure II.20 : Modification du fichier INI en mode formulaire.

b. En mode texte :

Si nous souhaitons modifier le fichier INI en tant que texte brut, on passe en mode source. L'éditeur offre plusieurs fonctionnalités en plus des fonctions habituelles de l'éditeur de texte, comme copier / coller, annuler / refaire et rechercher des textes.

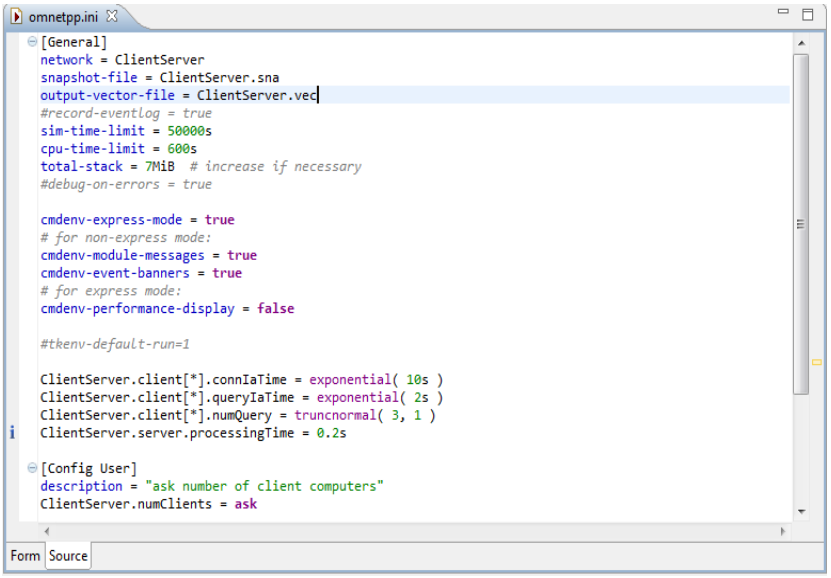


Figure II.21 : Modification du fichier INI en mode texte.

5.3. L'éditeur de fichiers de messages :

L'éditeur de fichier de message est un éditeur de texte de base avec support de mise en évidence de syntaxe.

La communication entre les différents modules se fait à travers les échanges de messages (**.msg**). Les messages peuvent représenter des paquets, des trames d'un réseau informatique, des clients dans une file d'attente ou bien d'autres types d'entités en attente d'un service. Les messages sont envoyés et reçus à travers des ports qui représentent les interfaces d'entrée et de sortie pour chaque module [21].

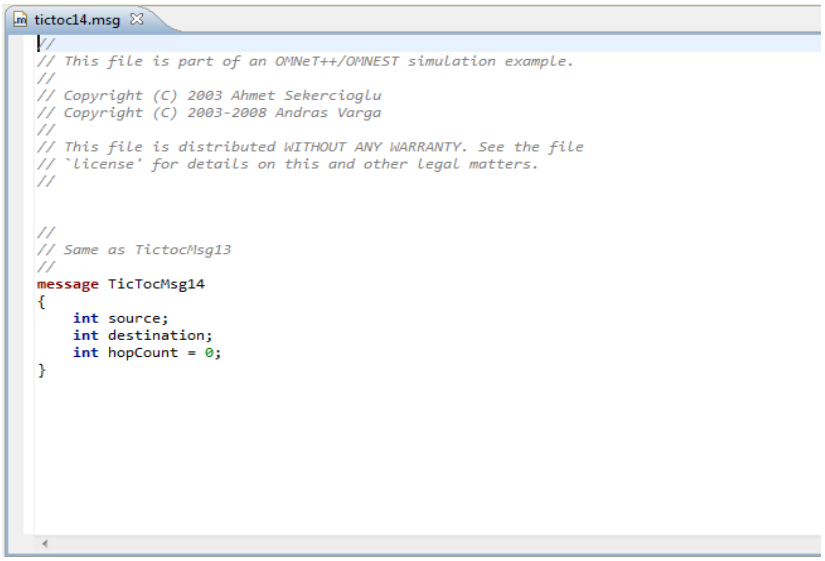


Figure II.22 : Éditeur de fichiers de messages.

5.3.1. Création de fichiers de message :

Pour créer un nouveau fichier**.msg**, nous choisissons **« File | New | message definition (msg) »** .Il ouvre un assistant dans lequel nous pouvons spécifier le répertoire cible et le nom du fichier pour la définition de message.

Nous pouvons choisir de créer un fichier **msg** vide ou choisir parmi les modèles prédéfinis.

Une fois que nous appuyons sur le bouton **« Finish »,** un nouveau fichier**.msg** sera créé avec le contenu demandé.

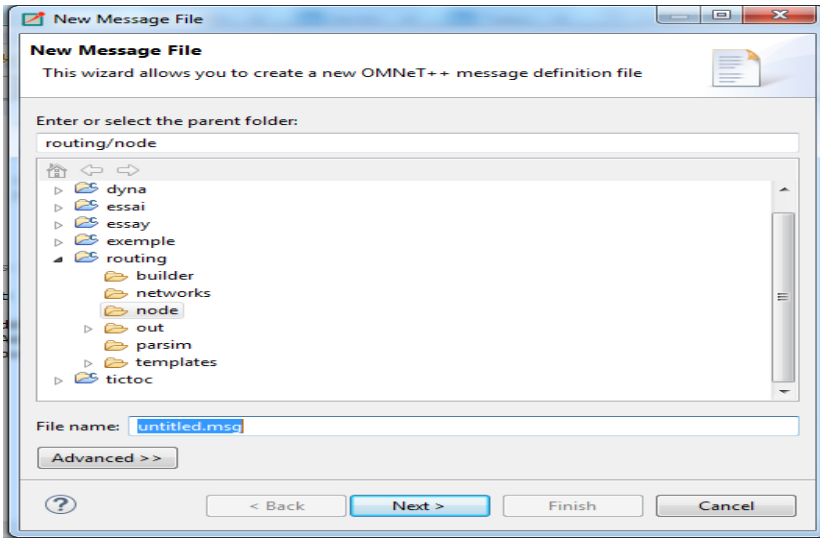


Figure II.23 : Création d'un nouveau fichier MSG.

5.4. L'éditeur de fichiers source :

L'éditeur de fichiers source avec l'extension **.cc** sert à initialiser les différents modules et décrire leur fonctionnement.

5.4.1. Création de fichiers source :

Pour créer un nouveau fichier**.cc**, nous choisissons **« File | New | source file ».** Il ouvre un assistant dans lequel nous pouvons spécifier le répertoire cible et le nom du fichier.

Une fois que nous appuyons sur le bouton **« Finish »**, un nouveau fichier source sera créé avec le contenu demandé.

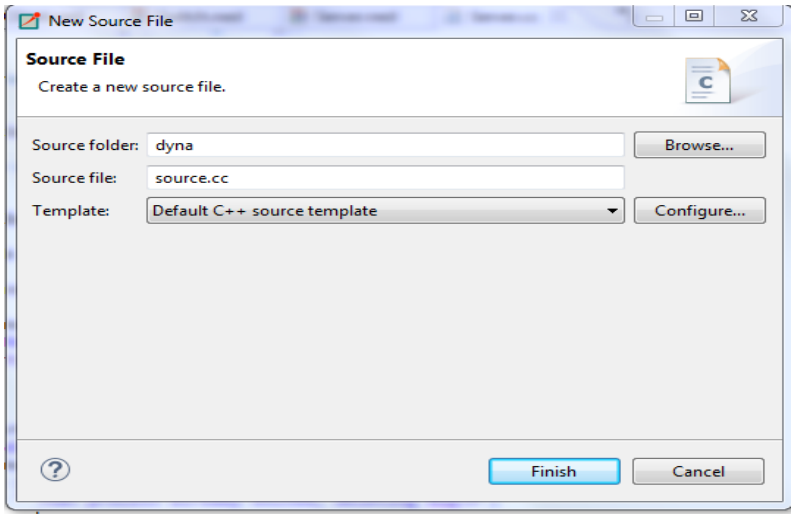


Figure II.24 : Création d'un nouveau fichier source.

5.5. L'éditeur de fichiers d'entête :

L'éditeur de fichiers d'entête avec l'extension **.h** est crée pour la déclaration des classes et variables qui seront utilisées dans le fichier de simulation C++ [21].

5.5.1. Création de fichiers d'entête :

Pour créer un nouveau fichier**.h**, nous choisissons **« File | New | Header File ».** Il ouvre un assistant dans lequel nous pouvons spécifier le répertoire cible et le nom du fichier.

Une fois que nous appuyons sur le bouton **« Finish »**, un nouveau fichier d'entête sera créé avec le contenu demandé.

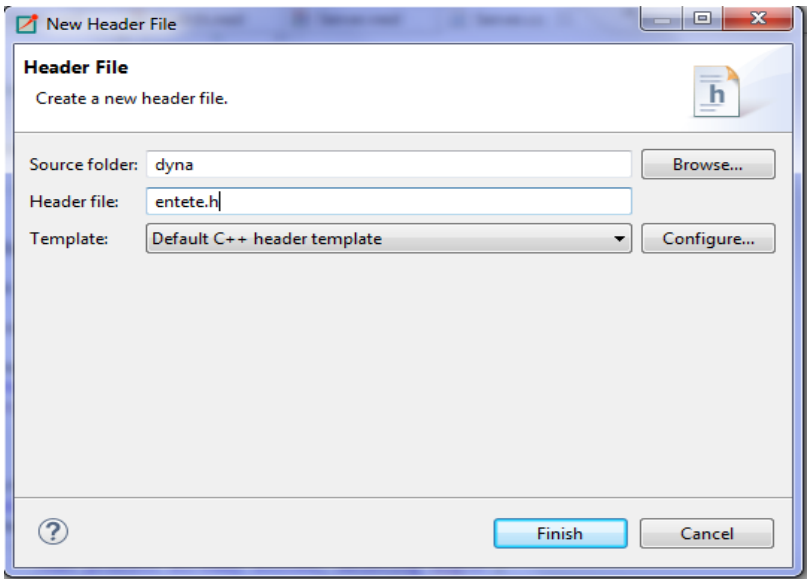


Figure II.25 : Création d'un nouveau fichier d'entête.

6. Exemple de simulation avec OMNeT++ :

6.1. Création de nouveau projet :

Dans ce projet, nous allons simuler un réseau qui contient deux Pc **"Tic"** et **"Toc".** Ces derniers, échangent un message "**TicTocMsg**" en boucle.

Nous allons commencer par créer un nouveau projet nommé **'tic-toc'**.

Nous choisissons **« File | New | OMNeT++ Project »,** une boîte de dialogue de l'assistant apparaîtra. Nous entrons **tic-toc** en tant que nom de projet, choisissons **« Empty project »** lors de la question sur le contenu initial du projet, puis nous cliquons sur **« Finish ».**

Un projet vide sera créé, comme nous pouvons le voir dans **« Project Explorer ».**

Un fichier **package.ned** sera généré dans le projet. Ouvrir ce fichier en mode source puis supprimer la première ligne : **package tic-toc**; et modifier la deuxième ligne **@license(LGPL)** par **@license(omnetpp)**.

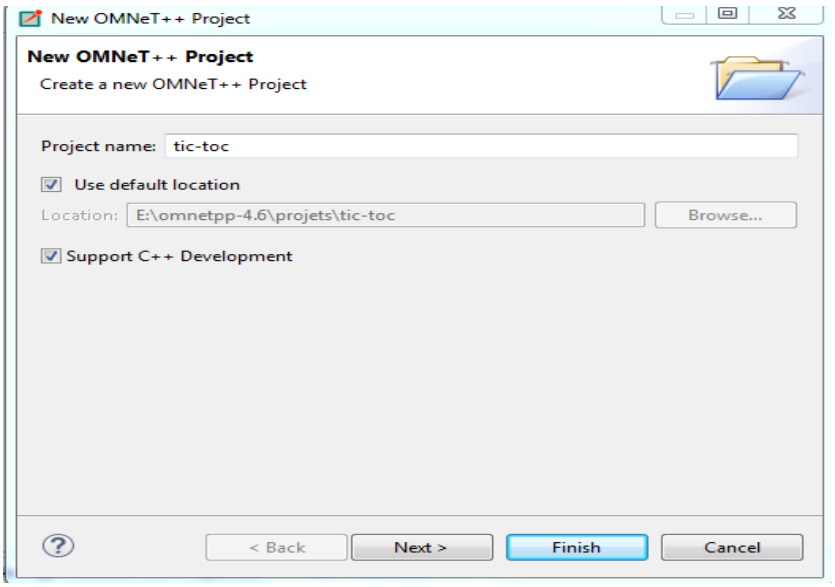


Figure II.26 : Création d'un nouveau projet nommé « tic-toc ».

6.2. Ajout du fichier NED :

OMNeT ++ utilise des fichiers NED pour définir des composants et les assembler en unités plus grandes comme des réseaux. Nous commençons à mettre en œuvre notre modèle en ajoutant un fichier NED.

Pour ajouter le fichier au projet, cliquons avec le bouton droit sur le répertoire du projet dans le panneau **« Project Explorer »** à gauche, puis nous sélectionnons **« New | File |Network Description File (NED) »** dans le menu. Nous nommons notre fichier **« tictoc.ned »** et nous choisissons **« Empty NED File »** puis **« Finish ».**

- Dans ce fichier nous créerons un module simple nommé **« Pc »** qui représentera un ordinateur.

```
simple Pc
    {
     }
```

- Puis nous ajoutons les entrées et les sorties pour ce module:

```
simple Pc
{
  @display("i=device/laptop");
  gates:
    input in;
    output out;

  }
```

- Création d'un réseau nommé **« TicToc »** contenant les deux Pc :

```
network TicToc
{
  @display("bgb=449,292");
  submodules:
    tic: Pc {
      @display("i=device/laptop");
    }
    toc: Pc {
      @display("p=41,30;i=device/laptop");
    }
  connections:
    tic.out --> {  delay = 100ms; } --> toc.in;
```

```
    tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

Le premier bloc dans le fichier déclare **Pc** comme un type de module simple. Les modules simples sont atomiques au niveau NED. Ils sont également des composants actifs, et leur comportement est implémenté en C ++. La déclaration indique également que **Pc** possède une porte d'entrée nommée et une porte de sortie nommée.

Le deuxième bloc déclare **TicToc** en tant que réseau. **TicToc** est assemblé à partir de deux sous-modules, tic et toc, les deux instances du type de module **Pc**. La porte de sortie de tic est connectée à la porte d'entrée de toc, et vice versa. Il y aura un délai de propagation de 100 ms dans les deux sens.

Lorsque nous avons terminé, revenons en mode graphique de ce fichier. Nous devrions voir quelque chose comme ceci :

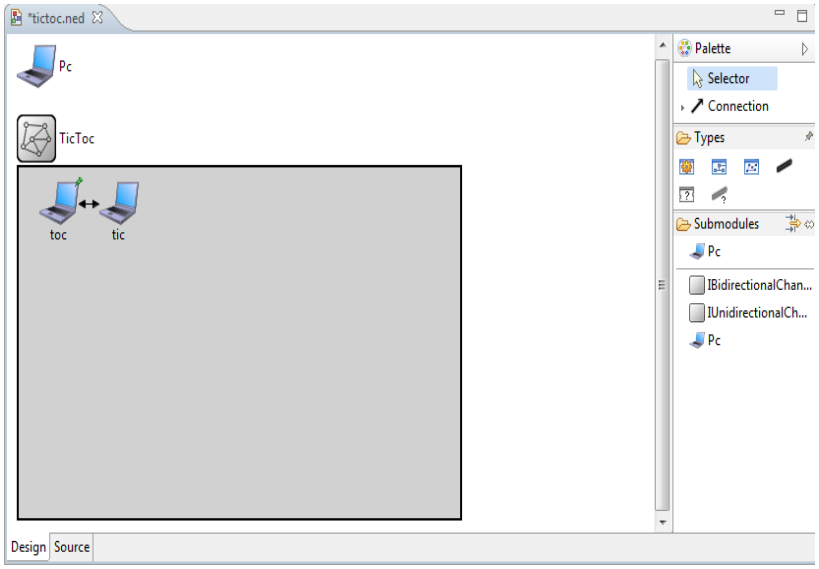


Figure II.27 : Fichier « tictoc.ned » en mode graphique.

6.3. Ajout du fichier C ++ :

Nous devons maintenant implémenter la fonctionnalité du module simple **Pc** en C ++. Création d'un fichier nommé **Pc.cc** en **« New | File |Source File »** dans le menu contextuel.

Voici le code source du fichier **Pc.cc** :

```
#include <string.h>
#include <omnetpp.h>
```

```
class Pc : public cSimpleModule
{
  protected:
    //La fonction virtuelle redéfinie suivante contient l'algorithme.
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
// La classe du module doit être enregistrée avec OMNeT ++
Define_Module(Pc);
void Pc::initialize()
{
    // L'initialisation est appelée au début de la simulation.
    // Pour démarrer le processus tic-toc-tic-toc, l'un des modules a besoin
   // pour envoyer le premier message. Laissons ceci être `tic '.
    // Suis-je Tic ou Toc?
    if (strcmp("tic", getName()) == 0)
    {
        // crée et envoyez le premier message à la porte "out". "TictocMsg" est une
       // chaîne arbitraire qui sera le nom de l'objet message.
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}
void Pc::handleMessage(cMessage *msg)
{
    // La méthode handleMessage () est appelée chaque fois qu'un message arrive
    // au module. Ici, nous l'envoyons à l'autre module, à travers
    // porte `out '. Parce que `` tic 'et `toc' font de même, le message
    // va rebondir entre les deux.
    send(msg, "out");
}
```

6.4. Ajout du fichier de configuration (.ini) :

Pour pouvoir exécuter la simulation, nous devons créer un fichier **omnetpp.ini**. **Omnetpp.ini** indique au programme de simulation le réseau que nous souhaitons simuler (car les fichiers NED peuvent contenir plusieurs réseaux), nous pouvons passer des paramètres au modèle, spécifier explicitement les graines pour les générateurs de nombres aléatoires, etc.

Créons un fichier **omnetpp.ini** à l'aide de l'élément de menu **« File | New | Initialisation file (INI) »**. Le nouveau fichier s'ouvrira dans un éditeur d'un fichier ini.

Pour l'instant, passons au mode source et entrons ce qui suit:

[General]

network =TicToc

Nous pouvons vérifier le résultat en mode formulaire:

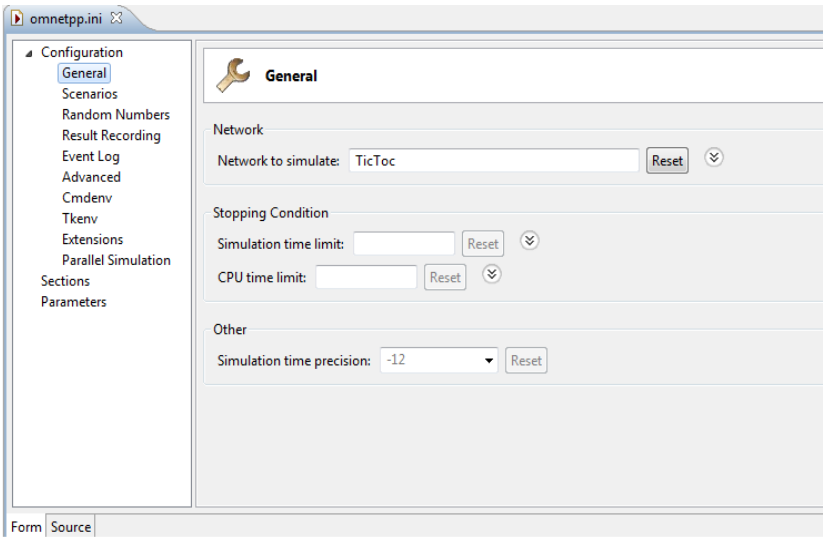


Figure II.28 : Fichier de configuration « omnetpp.ini » en mode formulaire.

6.5. Compilation et exécution de la simulation :

L'IDE va construire notre projet automatiquement. S'il existe des erreurs de compilation, nous devons les rectifier jusqu'à ce que nous obtenions une compilation et une liaison sans erreur. Nous pouvons déclencher manuellement une version en appuyant sur le choix de **tic-toc| Build Project**, ou nous appuyons sur **Ctrl + B.**

Après avoir réussi à créer notre projet, passons au lancement de notre simulation.

Allons à **« Project Explorer | tic-toc | bouton droit | Run as | OMNet++ simulation »** nous devrions voir apparaître une nouvelle fenêtre GUI, similaire à celle de la capture d'écran ci-dessous. La fenêtre appartient à Tkenv, la GUI principale d'exécution de la simulation OMNeT ++. Nous devrions également voir le réseau contenant **tic** et **toc** affiché graphiquement dans la zone principale.

Appuyons sur le bouton **« Run »** de la barre d'outils pour lancer la simulation. Ce que nous devrions voir, c'est que **tic** et **toc** échangent des messages les uns avec les autres.

Remarque :

Si nous souhaitons construire l'exécutable de simulation sur la ligne de commande, nous créons un **Makefile** à l'aide de la commande **opp_makemake**, puis entrons **make** pour créer le projet. Il produira un exécutable qui peut être exécuté en entrant : **./tic-toc**.

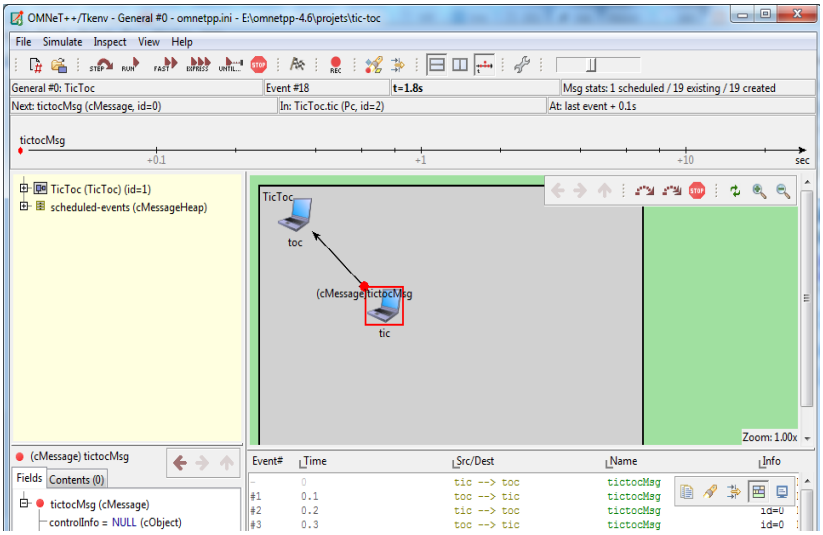


Figure II.29 : Résultat d'exécution de tic-toc.

7. Plates formes d'OMNeT++:

Le simulateur OMNeT++ n'est pas spécialisé pour les réseaux de capteurs sans fil, pour cela il existe plusieurs plates formes basées sur OMNeT++ qui essayent d'introduire ce manque telles que Mobilité FrameWork, INET et Castalia.

7.1. Mobilité FrameWork :

La librairie MF est une extension du simulateur OMNeT++. Elle a été développée par une équipe de chercheurs à l'université de Berlin. Cette librairie est destinée à soutenir les

réseaux sans fil et mobiles au sein de simulations OMNeT++. En effet, elle permet une bonne manipulation des nœuds mobiles et la gestion des connexions dynamiques pour avoir un modèle de mobilité de réseau sans fils qui fournie des résultats le plus proche possible du monde réel. En outre, MF prévoit des modules de base qui peuvent être utilisés pour former de nouveaux modules. Avec ce concept, un programmeur peut facilement développer ses propres protocoles avec l'interface nécessaire [9].

Elle peut être utilisée pour la simulation de :

- Les réseaux sans fil fixes
- Les réseaux sans fil mobiles
- Les réseaux distribués (ad-hoc) et les réseaux centralisés
- Les réseaux de capteurs
- Les multi-réseaux sans fil

7.2. INET :

INET est une librairie open source pour la simulation des réseaux informatiques dans l'environnement OMNeT++. Elle contient IPv4, IPv6, TCP, UDP, des protocoles implémentés, et plusieurs modèles d'application. Les modèles de couche de liaison sont PPP, Ethernet et 802.11.Le routage statique peut être configuré à l'aide du réseau auto configurateur, ou nous pouvons utiliser le protocole de routage mis en œuvre. INET prend en charge les simulations des réseaux sans fil et mobiles, ainsi les réseaux ad-hoc [9].

7.3. CASTALIA :

Castalia est un simulateur pour réseaux de capteurs sans fil, Body Area Networks (BAN) et généralement des réseaux de périphériques intégrés de faible puissance. Il est développé dans le thème Networked Systems à NICTA depuis 2007. Castalia est utilisé par des chercheurs et des développeurs pour tester leurs algorithmes et / ou protocoles distribués dans des modèles réalistes de chaînes et de radios sans fil, avec un comportement de nœud réaliste, en particulier en ce qui concerne l'accès à la radio. Les caractéristiques principales de Castalia incluent: un modèle pour la variation temporelle de la perte de chemin, l'interférence des grains finis et le calcul RSSI, la modélisation des processus physiques, la dérive de l'horloge des nœuds et plusieurs protocoles MAC populaires mis en

œuvre. Castalia est très paramétrique. Il fournit des outils pour aider à exécuter de grandes études de simulation paramétrique, traiter et visualiser les résultats [9].

Conclusion :

Dans ce chapitre, nous avons présentés les différents simulateurs existants pour les réseaux de capteurs sans fil. Parmi ces simulateurs notre choix a été fixé sur OMNeT++ essentiellement à cause de sa construction modulaire et sa flexibilité.

Le manque du simulateur OMNeT++ au niveau des protocoles de réseaux de capteur a été solutionné avec l'implémentation des différentes plates formes telles que Castalia.

Dans le chapitre suivant, nous présenterons d'une façon détaillée la plate forme Castalia basée sur OMNeT++.

Introduction :

Parmi les modèles de simulation actuellement disponibles sous le simulateur OMNeT++, nous avons Castalia. Ce dernier est le plus avancé et le plus utilisé dans la communauté scientifique pour la simulation des réseaux de capteurs sans fil.

Nous pouvons citer certains de ses avantages:

- Son architecture est modulaire permettant l'intégration de nouveaux modèles.
- Basé sur C++ (récemment, il inclut aussi C#) pour le développement du noyau.
- Les classes de base du simulateur peuvent être étendues et personnalisées.
- Les modèles conçus en l'utilisant sont plus proches de la réalité.

Ce chapitre fournit une étude détaillée du simulateur Castalia et met en évidence ses exigences système et son installation, ses composants principaux, tels que sa structure et sa modélisation.

1. Présentation de CASTALIA :

1.1. Définition :

Castalia est un simulateur pour réseaux de capteurs sans fil, Body Area Networks (BAN) et généralement des réseaux de périphériques intégrés de faible puissance. Il est basé sur OMNeT ++ et peut être utilisé par des chercheurs et des développeurs qui souhaitent tester leurs algorithmes et / ou protocoles distribués dans des modèles de chaînes et de radio sans fil réalistes, avec un comportement de nœud réaliste, en particulier en ce qui concerne l'accès à la radio. Castalia peut également être utilisé pour évaluer différentes caractéristiques de plate-forme pour des applications spécifiques, car elle est hautement paramétrique et peut simuler une large gamme de plates-formes [22].

1.2. Caractéristiques de CASTALIA :

Le simulateur Castalia dispose de plusieurs fonctionnalités qui lui permettent de modéliser, à travers la simulation, un comportement proche de la réalité d'un nœud de capteur. Les principales caractéristiques de Castalia sont [15] [22]:

- Un canal variant dans le temps.
- Plusieurs méthodes de calcul des interférences.

- Définition de plusieurs modes (**RX**, **TX**, et en veille) et les énergies et délais de transitions entre ces modes.
- Modèle de processus physique hautement flexible.
- MAC et protocoles de routage disponibles.

1.3. Structure de CASTALIA :

Pour obtenir un comportement de nœud réaliste, Castalia simule tous les composants des nœuds de capteurs réels. La **figure III.1** montre la structure interne du module nœud Castalia, à partir de cette figure nous pouvons constater que le module de nœud est composé de plusieurs modules de périphériques, y compris les gestionnaires d'applications, de gestion de capteurs, de gestion de ressources, de gestion de mobilité et de communication. Sur cette figure, les flèches solides représentent le message passant entre les modules et les flèches pointillées représentent l'appel de fonction. Comme nous le voyons, de nombreux modules appellent le gestionnaire de ressources pour simuler la consommation d'énergie. L'application, le routage et les modules MAC sont les plus fréquemment modifiés par l'utilisateur pour la mise en œuvre de nouveaux algorithmes et protocoles [22] [10].

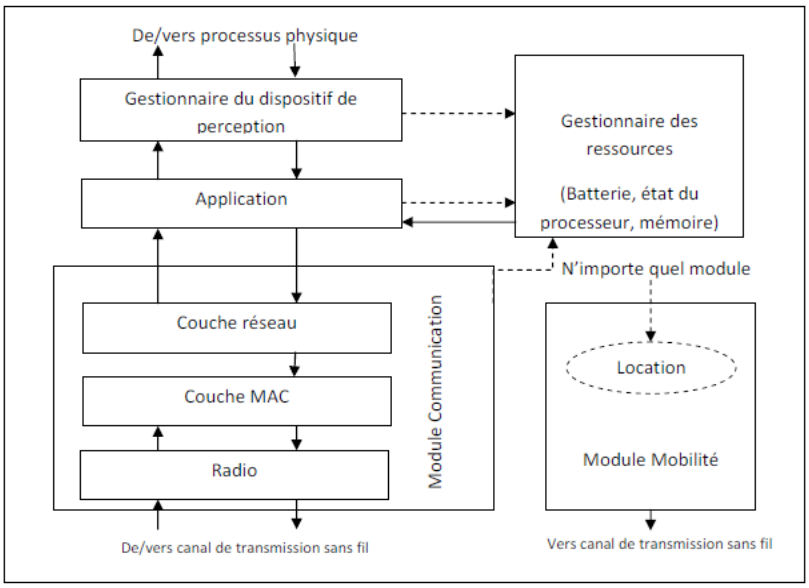


Figure III.1 : Structure interne du module nœud Castalia.

2. Installation de CASTALIA :

D'abord, nous devons installer OMNeT ++ et Castalia, ensuite, nous pouvons appliquer le script **updateCastalia.sh**. Enfin, nous pouvons importer Castalia vers l'IDE ou l'exécuter à partir de l'invite de commande [23].

2.1. Installation d'OMNeT ++ :

La description de l'installation d'OMNeT version 4.6 sous Windows est déjà faite dans le chapitre précédent.

2.2. Installation de Castalia-3.2 :

D'abord, il faut télécharger le fichier **Castalia-3.2.tar.gz** [24]:
Copier l'archive téléchargée dans le répertoire où on souhaite l'installer. Choisir un répertoire dont le chemin d'accès complet ne contient aucun espace.

Extraire le fichier **tar** en utilisant un programme spécifique tel que **WinRAR**. Cela créera un répertoire nommé Castalia-3.2 contenant des répertoires nommés **bin**, **Simulations, src** et les fichiers nommés **CHANGES.txt**, **LICENCE, makemake**, etc.

Ensuite, nous devons ajouter le répertoire bin de Castalia à la variable d'environnement **Path**. Pour ce faire, il faut suivre les étapes ci-dessous:

- Cliquer sur le bouton **« Démarrer »** et commencer à taper **« variables d'environnement »** dans la barre de recherche.
- Sélectionner l'option **« Modifier les variables d'environnement système »** lorsqu'elle apparaît dans la liste des résultats de recherche. Cela ouvrira l'onglet avancé de la fenêtre des propriétés du système.
- Dans cet onglet, nous cliquons sur le bouton des variables d'environnement. Cela ouvrira la fenêtre des variables d'environnement.
- Dans cette fenêtre, nous sélectionnons la variable « Path » (sous Variables système) et nous cliquons sur le bouton **« Modifier ».**
- Maintenant, à la fin du champ de la valeur variable, nous ajoutons le chemin d'accès au répertoire bin de Castalia, en utilisant un point-virgule pour le séparer des autres chemins. Nous avons extrait Castalia en **E: \ Castalia-3.2**, alors nous devons ajouter: **E: \ Castalia-3.2 \ bin** au champ de la valeur variable (voir **Figure III.2**).
- Appuyons sur le bouton **OK** jusqu'à ce que toutes les fenêtres se ferment.

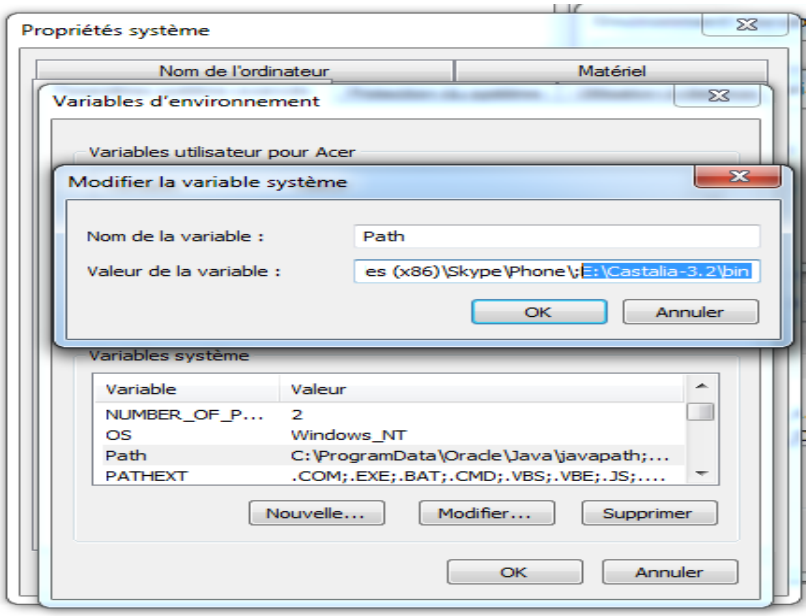


Figure III.2 : Ajout de Castalia au chemin.

2.3. Exécution du script updateCastalia.sh :

Démarrer **mingwenv.cmd** dans le répertoire omnetpp-4.6, nous avons aussi téléchargé le fichier **Castalia-3.2_OMNeT-IDE_Windows_Linux-master** et nous l'avons extrait vers **E: \ Castalia-3.2_OMNeT -IDE_Windows_Linux-master.**

Maintenant nous devons exécuter les commandes suivantes pour modifier ce répertoire puis nous exécutons le script updateCastalia.sh:

$ cd E:/Castalia-3.2_OMNeT-IDE_Windows_Linux-master

$ ./updateCastalia.sh

Si les commandes ont été exécutées avec succès, nous devons voir un écran comme celui de la figure ci-dessous.

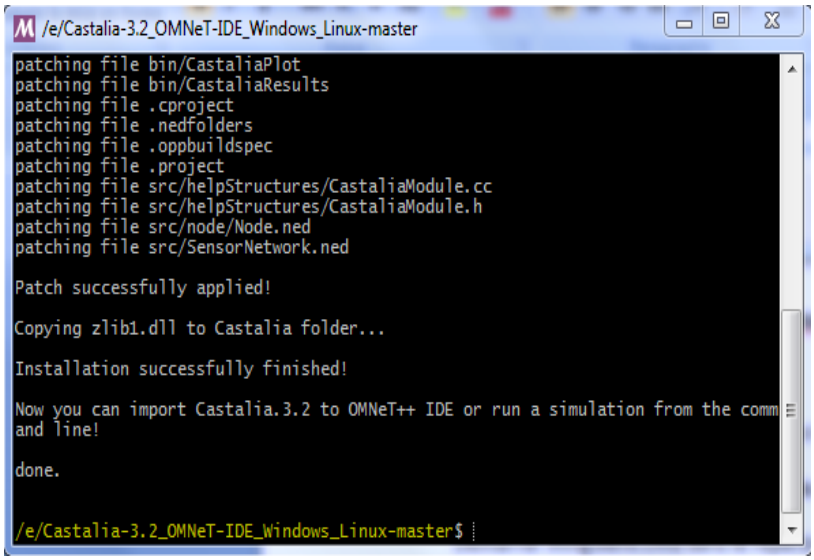


Figure III.3 : Exécution avec succès updateCastalia.sh.

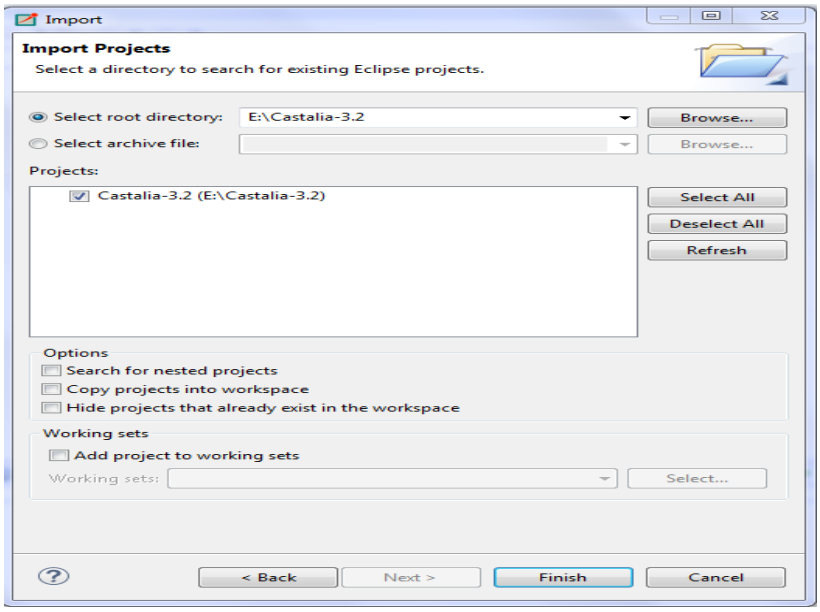
En ce moment, nous pouvons construire et exécuter des simulations de Castalia à partir de la ligne de commande (mingwenv.cmd). De plus, nous pouvons importer Castalia dans OMNeT ++ IDE et exécuter des simulations.

2.4. Exécution de CASTALIA sur l'OMNeT ++ IDE :

Dans cette section, nous décrivons comment importer Castalia dans OMNeT ++, la construire, exécuter des simulations et afficher les résultats de la simulation (trace).

2.4.1. Importation de Castalia-3.2 à OMNeT ++ IDE :

Ouvrir l'IDE OMNeT ++ et cliquer sur **« File|Import ».** Dans la fenêtre d'importation, développer le dossier Général, nous sélectionnons l'option **« Existing Projects into Workspace »** et nous cliquons sur **« Next ».** Ensuite, nous cliquons sur le bouton **« Browse »** (voir **Figure III.4**) et sélectionner le répertoire d'installation Castalia-3.2 et enfin nous cliquons sur **« Finish ».**



Le projet Castalia-3.2 devrait apparaître dans l'onglet **« Projects Explorer »** (voir la **Figure III.5** ci-dessous).

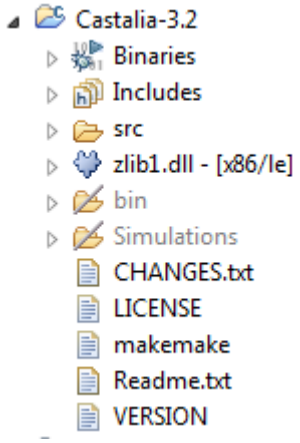


Figure III.5 : Castalia-3.2 dans l'Explorateur de projets.

2.4.2. Construction de Castalia-3.2 à partir d'OMNeT ++ IDE :

Pour construire Castalia, nous cliquons avec le bouton droit sur le projet Castalia-3.2 dans **« Projects Explorer »** et nous sélectionnons l'option « **build projet** ».

Si le processus de construction a terminé avec succès, la console devrait ressembler à celle de la figure ci-dessous.

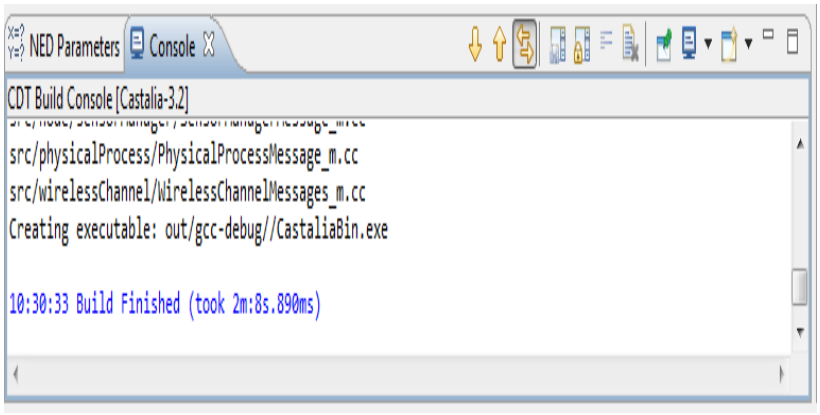


Figure III.6 : Résultat Castalia Build dans la console.

2.4.3. Exécution d'une simulation de Castalia à partir d'OMNeT ++ IDE :

Pour exécuter la simulation d'exemple de **RadioTest**, nous accèdons au répertoire **Simulations / radioTest**, nous cliquons avec le bouton droit sur le fichier **« omnetpp.ini »** et nous sélectionnons l'option **« Run As > 1 OMNeT ++ Simulation».**

Si tout se passe bien, nous devrions pouvoir voir la trace de simulation dans la console, tout comme dans la figure ci-dessous.

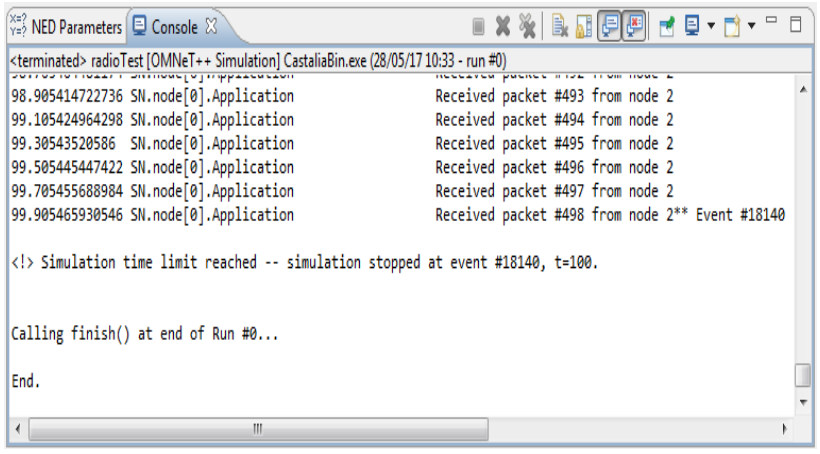


Figure III.7 : Trace de simulation dans la console.

2.5. Exécution de Castalia à partir de l'invite de commande :

Dans Windows, pour exécuter Castalia à partir de l'invite de commande, tout comme les utilisateurs de Linux, il nous suffit d'ouvrir l'invite **mingwenv.cmd** et de taper les commandes ci-dessous.

Ensuite, nous montrons brièvement comment utiliser les principales commandes de Castalia, afin que nous puissions vérifier si notre installation de Castalia a réussi.

Avant d'exécuter une simulation, nous devons d'abord créer Castalia. Nous avons déjà installé Castalia dans **E: \ Castalia-3.2**, nous ouvrons **mingwenv.cmd** et nous tapons les commandes suivantes l'une après l'autre :

$ cd E:/Castalia-3.2
$ make clean
$ ./makemake
$ make

2.6. Instllation de gnuplot :

Si nous souhaitons voir les résultats graphiquement, nous pouvons utiliser le script **CastaliaPlot**. Cependant, nous devons d'abord télécharger et installer **Gnuplot**, puis ajouter son répertoire **bin** au chemin d'accès (comme nous l'avons fait avec le répertoire bin de Castalia dans la section **2.2**).

Nous devons fermer et de rouvrir l'invite mingwenv.cmd, de sorte que l'installation Gnuplot prend effet [33].

3. Utilisation de CASTALIA :

Depuis la sortie de Castalia 3.0, la gestion des entrées et des sorties a considérablement changé, nous avons deux scripts pour aider à exécuter des simulations et à interpréter les résultats. On les appelle **Castalia** et **CastaliaResults** respectivement. Depuis la version 3.1, nous avons également **CastaliaPlot** pour nous aider à créer des graphiques. Ils résident tous à **Castalia / bin**. L'exécutable du simulateur s'appelle **CastaliaBin** réside dans Castalia [22].

3.1. Utilisation Castalia et CastaliaResults :

Le fichier de sortie et le fichier de résultat seront générés dans le dossier en cours après l'exécution de la simulation. Le fichier de résultat est un fichier **txt** dont le nom est nommé par date et heure. Par exemple, un nom de fichier de résultat typique est **170528-105358.txt**, ce qui signifie qu'il a été généré le 2017-05-28 à 10:53:58.

Il existe de nombreuses options pour **CastaliaResult**, comme -s est utilisé pour montrer les données, -n montrer les données de chaque nœud, -f est utilisé pour le filtre.etc.

3.2. Utilisation de CastaliaPlot :

CastaliaPlot appelle le module **gnuplot** pour générer des chiffres. CastaliaPlot utilise le fichier de résultat généré par **CastaliaResult** comme entrée pour tracer les données [22]. Plusieurs options sont également proposées pour modifier la couleur, la légende, le style etc.

3.3. Fichier de configuration :

Castalia dispose d'une structure modulaire avec de nombreux modules d'interconnexion. Chacun des modules a un ou plusieurs paramètres qui affectent son comportement.

Un fichier **NED** (fichier avec extension **.ned** dans le code source Castalia) définit la structure de base d'un module en définissant ses portes d'entrée / sortie et ses paramètres. Dans le fichier NED, nous pouvons également définir des valeurs par défaut pour les paramètres. Un fichier de configuration (généralement appelé **omnetpp.ini** et résidant dans l'arborescence **dir Simulations**) attribue des valeurs aux paramètres, ou simplement les réaffecte à une valeur différente de leur valeur par défaut. De cette façon, nous pouvons créer une grande variété de scénarios de simulation [22].

Le fichier de configuration commence par la section **« General »** dont on peut définir le scénario de base que décrit ce fichier. Les paramètres tels que le temps de simulation et le nombre de nœuds n'ont pas de valeurs par défaut et doivent être définis dans cette section.

4. Modélisation dans CASTALIA :

Le simulateur Castalia modélise les différents aspects d'un réseau de capteurs sans fil, des communications aux processus physiques, il fournit également les détails de tous les modules non-composites et leurs paramètres. La communication est l'aspect le plus soigneusement modelé du simulateur Castalia, du canal sans fil au comportement de la radio et à la mise en œuvre de différents protocoles MAC [22].

4.1. Module canal sans fil :

Le modèle de canal sans fil de Castalia a été créé en fonction de centaines de milliers de mesures expérimentales, et comprend également la modélisation moyenne de la perte de chemin et le comportement de variation temporelle. Selon la perte de trajet, la variation temporelle et la puissance d'émission du nœud capteur, le modèle de canal sans fil peut être utilisé pour calculer la puissance du signal reçue du nœud transmetteur. Par conséquent, Castalia est le simulateur le plus réaliste que l'on puisse trouver pour RCSF [22].

4.1.1. Modèles naïfs :

Il serait utile de définir des modèles plus simples afin que les résultats de simulation puissent être comparés aux résultats d'autres simulateurs et, plus important encore, afin que l'utilisateur puisse tester différentes hypothèses sur la raison pour laquelle un algorithme distribué ne fonctionne pas (ou moins performants) lorsque des hypothèses plus réalistes sont prises en compte. Ceci est particulièrement vrai avec les modèles de communication, en particulier ceux de bas niveau (canal et radio) [22].

Dans un fichier omnetpp.ini, nous pouvons rendre la chaîne sans fil plus simple en définissant simplement:

SN.wirelessChannel.sigma = 0

SN.wirelessChannel.bidirectionalSigma = 0

Et pour que la réception d'un paquet soit parfaite, nous proposons un mode spécial avec un schéma de modulation idéal. Pour le choisir, il suffit de définir:

SN.node[*].Communication.Radio.mode = "IDEAL"

4.2. Module radio :

Le module radio tente de capturer de nombreuses fonctionnalités d'une radio réelle à faible puissance, qui est susceptible d'être utilisée dans les plates-formes de réseau de capteurs sans fil. Les principales caractéristiques de module radio sont [22]:

- Etats multiples: le sommeil, la transmission et l'écoute.
- Les retards de transition d'un état à l'autre.
- Consommation d'énergie différente pour les différents états.
- Modèles de modulation multiples supportés nativement (modulation idéale).

4.3. Module MAC :

Le protocole de contrôle d'accès moyen est une partie importante du comportement du nœud, il existe donc un module distinct qui le définit. Dans Castalia, nous avons mis en place quatre modules MAC principaux [22]:

- **TunableMAC :**

Un MAC en service qui expose de nombreux paramètres à l'utilisateur et à l'application pour le réglage.

Cependant, il convient de noter que ce protocole a été construit avec la communication de diffusion à l'esprit (c'est-à-dire sans unicast), donc il ne prend pas en charge les accusés de réception. Il peut être réglé en ce qui concerne sa persistance et ses politiques de sauvegarde. Une autre fonction majeure est de recycler la radio et de transmettre un train approprié de balises avant chaque transmission de données pour réveiller les récepteurs potentiels (puisque les nœuds ne sont pas alignés dans leurs horaires de sommeil) [22].

- **TMAC et SMAC** :

T-MAC est un MAC populaire pour les RCSF car il emploie de nombreuses techniques pour réduire la consommation d'énergie (en utilisant un cyclisme et une synchronisation agressifs)

tout en essayant de maintenir les performances (par exemple, la livraison des paquets) en adaptant son cycle de service en fonction des besoins de trafic.

S-MAC peut être considéré comme le prédécesseur de T-MAC car il a lancé plusieurs des techniques, mais utilise un cycle de travail plus rigide [22].

- **MAC IEEE 802.15.4 :**

C'est la norme pour les réseaux sans fil à faible puissance, bien que pas fréquent dans les RCSF [22].

4.4. Module Réseau :

Le routage dans Castalia reçoit moins d'attention par rapport aux autres modules de communication. Même si la couche de routage bénéficie d'un niveau d'abstraction similaire pour définir un protocole comme d'autres couches (par exemple, Application, MAC), il existe moins de protocoles mis en œuvre et diffusés avec Castalia.

Le premier support de Castalia pour le routage est livré avec la version **1.2**. Dans la version **1.3**, deux protocoles de routage sont publiés **simpleTreeRouting** et **multipathRingsRouting** qui sont restés actifs jusqu'à la version **2.3b**. Dans **Castalia 3.2** il existe que des **multipathRings** et **bypassRouting** (un module qui, comme son nom l'indique, n'implémente aucun routage) [22].

4.5. Module application :

C'est le module dans lequel, l'utilisateur effectue normalement beaucoup de changements pour implémenter de nouveaux algorithmes [22].

4.6. Module gestionnaire de capteurs :

Le module de nœud du capteur se connecte au canal sans fil pour communiquer avec d'autres nœuds. Il se connecte aux processus physiques afin qu'il puisse les échantillonner.

4.7. Module processus physique :

Pour avoir un environnement de simulation proche de la réalité, nous avons besoin d'avoir des modèles flexibles de processus physique (corrélation spatiale des données, variabilité dans le temps, …). Pour ce besoin, le simulateur offre un module générique pour le processus physique pour fournir les données aux nœuds de capteurs. La base de ce modèle est les valeurs des sources dont l'influence est diffusée dans l'espace. Les sources peuvent changer leurs positions et leurs valeurs [22].

4.8. Module gestionnaire de ressources :

Le module gestionnaire de ressources surveille l'énergie dépensée par le nœud et détient également des quantités spécifiques aux nœuds telles que la dérive de l'horloge et la consommation d'énergie de base. Il existe certaines dispositions pour suivre la mémoire ou

pour définir différentes consommations d'énergie pour le processeur, mais elles ne sont pas entièrement opérationnelles. Les modules qui proposent des périphériques matériels (c'est-à-dire la radio et le gestionnaire de capteurs) envoient des messages au gestionnaire de ressources afin de signaler la puissance qu'ils dessinent actuellement. Le gestionnaire de ressources dispose alors d'une vue complète de la puissance totale tirée et, à partir de cela, il calcule l'énergie consommée chaque fois que nous avons un changement de puissance ou périodiquement (si une modification de puissance n'est pas survenue depuis un certain temps) [22].

4.9. Module gestionnaire de mobilité:

Le module de mobilité spécifie comment les nœuds se déplacent dans l'espace. Il détient l'emplacement où d'autres modules peuvent accéder à tout moment à l'aide d'un appel de fonction, et il notifie périodiquement le canal sans fil de la position d'un nœud. La notification de la chaîne sans fil est effectuée pour des raisons d'efficacité. Étant donné que la chaîne sans fil a besoin de l'emplacement de tous les nœuds, Il est beaucoup mieux pour le module de mobilité d'avertir le canal si quelque chose a changé (par exemple, il pourrait l'avertir, lorsque le nœud vient de changer de cellule) [22].

5. Liste des modules dans CASTALIA :

Le diagramme suivant fournit un moyen simple de voir tous les modules et de montrer leurs relations hiérarchiques. Notons que parfois un module peut être l'un de plusieurs modules similaires possibles (modules qui partagent une interface commune). Cela se produit avec des modules tels que l'application, le routage, le MAC et gestionnaire de mobilité. Par exemple, nous avons plusieurs modules d'applications qui partagent le même nom hiérarchique complet: **SN.node [*]. Application**. Lequel des différents modules d'application que nous utiliserons dépendent d'un paramètre du module de noeud nommé **ApplicationName**. ApplicationName donne le nom du fichier NED qui implémente le module d'application désiré [22].

Dans le schéma ci-dessous, nous fournissons une liste complète des modules qui suit la structure du module (et du répertoire) dans Castalia, à partir du module le plus haut.

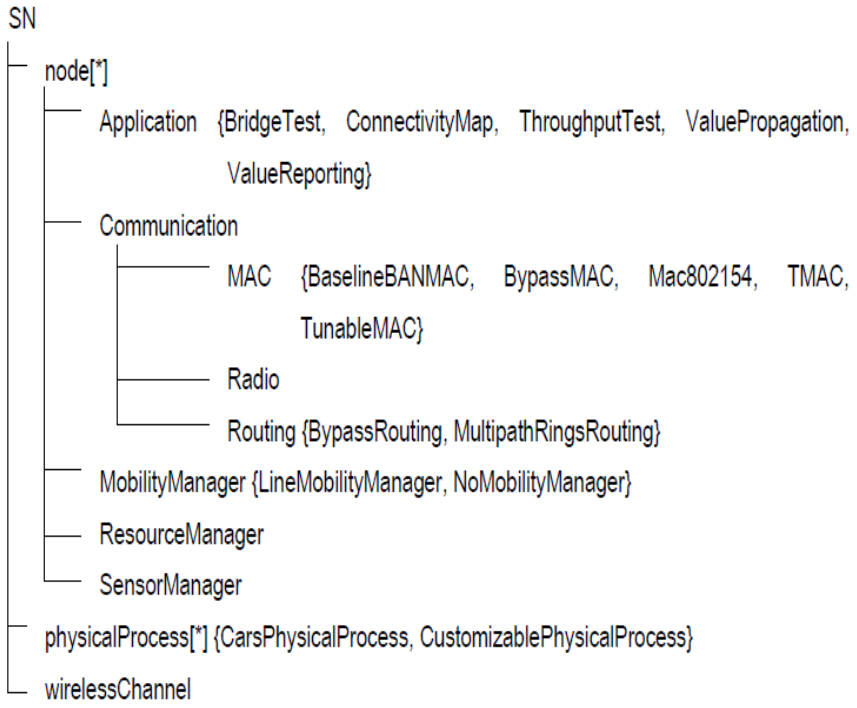


Figure III.8 : Diagramme représentant la liste des modules dans Castalia.

Conclusion :

Dans ce chapitre nous avons abordé une description du simulateur Castalia qui est spécifique à la simulation des réseaux de capteurs sans fil. Parmi ses caractéristiques importantes, nous citons la capacité de donner des résultats plus proches de la réalité et pour cette raison que nous l'avons choisi comme simulateur de notre application.

Notre contribution consiste à simuler un réseau de capteur sans fil dans l'environnement de la simulation OMNeT++ avec la plate forme Castalia dans le cas d'un calcul distribué.

Dans le chapitre suivant, nous allons mettre en place cette contribution dont on décrira nos étapes de conception de notre application.

Introduction :

L'objectif de ce chapitre est la réalisation d'un nouveau module d'application qui implémentera notre application. En premier lieu, nous présenterons une description globale du modèle qui illustre l'architecture physique pour la construction de notre application, puis nous spécifierons les outils, les paramètres et techniques qui nous a aidé à réaliser notre travail et nous finirons par donner les différents codes sources implémenté pour la réalisation.

Problématique :

Les réseaux de capteurs sans fil représentent une technologie émergente qui vise à offrir des capacités innovantes. Leur utilisation ne devrait cesser d'augmenter et ceci dans de nombreux domaines qu'ils soient scientifiques, logistiques, militaires, etc. Cependant, la limitation de ressources des nœuds capteurs constitue une contrainte importante, principalement en termes de capacité mémoire et d'autonomie d'énergie. L'une des conséquences de cette limitation de ressources est que dans un réseau de capteur sans fil, un capteur tout seul ne peut pas réaliser un calcul complexe.

Pour remédier à ce problème, plusieurs capteurs mettent en commun leurs ressources afin d'offrir une puissance de calcul que nous pouvons exploiter.

1. Environnement technique du développement :

Nous allons détaillés les outils utilisés dans la réalisation de notre simulation.

1.1. Environnement matériel :

La simulation a été réalisée sur un ordinateur de caractéristiques suivantes :

- Un microprocesseur : Intel® Core ™ i3-2348M, avec fréquence de: 2.30 GHz
- Une carte mémoire de 4.00 Go.
- Un disque dur de 500 Go.

1.2. Environnement logiciel :

Pour les logiciels utilisés nous avons opté pour :

- Windows 7 comme système d'exploitation.
- Le simulateur OMNeT++ 4.6.
- Le simulateur Castalia 3.2.

2. Implémentation :

2.1. Présentation de notre application :

Notre application fonctionne de la manière suivante:

Nous définissons les emplacements de départ pour nos 4 nœuds manuellement, ces nœuds sont très rapprochés de sorte qu'il n y aura pas une perte de paquet et être sur que la requête est reçu par tout les nœuds.

Le nœud 0 c'est le nœud puits et c'est lui qui va initier la requête et l'envoyer pour les autres nœuds de réseaux.

Chaque nœud recevant la requête lance un téléchargement d'une partie de fichier se trouvant sur le disque et renvoie à son tour un autre paquet au nœud puits.

A la fin le nœud puits collecte toutes les sorties qui viennent des nœuds et les affiche comme sortie d'application.

Pour effectuer une simulation dans Castalia nous devons choisir chacun des éléments nécessaires à celle-ci.

2.2. Description de la conception d'un nœud :

Dans cette partie, on va modéliser le module nœud avec ses différents sous modules, pour cela nous présenterons l'organigramme du simulateur Castalia afin de voir la hiérarchie des différents modules contenant dans celui-ci, puis nous allons présenter une partie de fichier graphique **«Node.ned»** qui illustre les différents modules utilisés.

2.2.1. Organigramme du simulateur Castalia :

L'organigramme suivant représente l'architecture du simulateur Castalia sous OMNeT++ et les différents modules qui le compose, on a identifié ainsi la couche application.

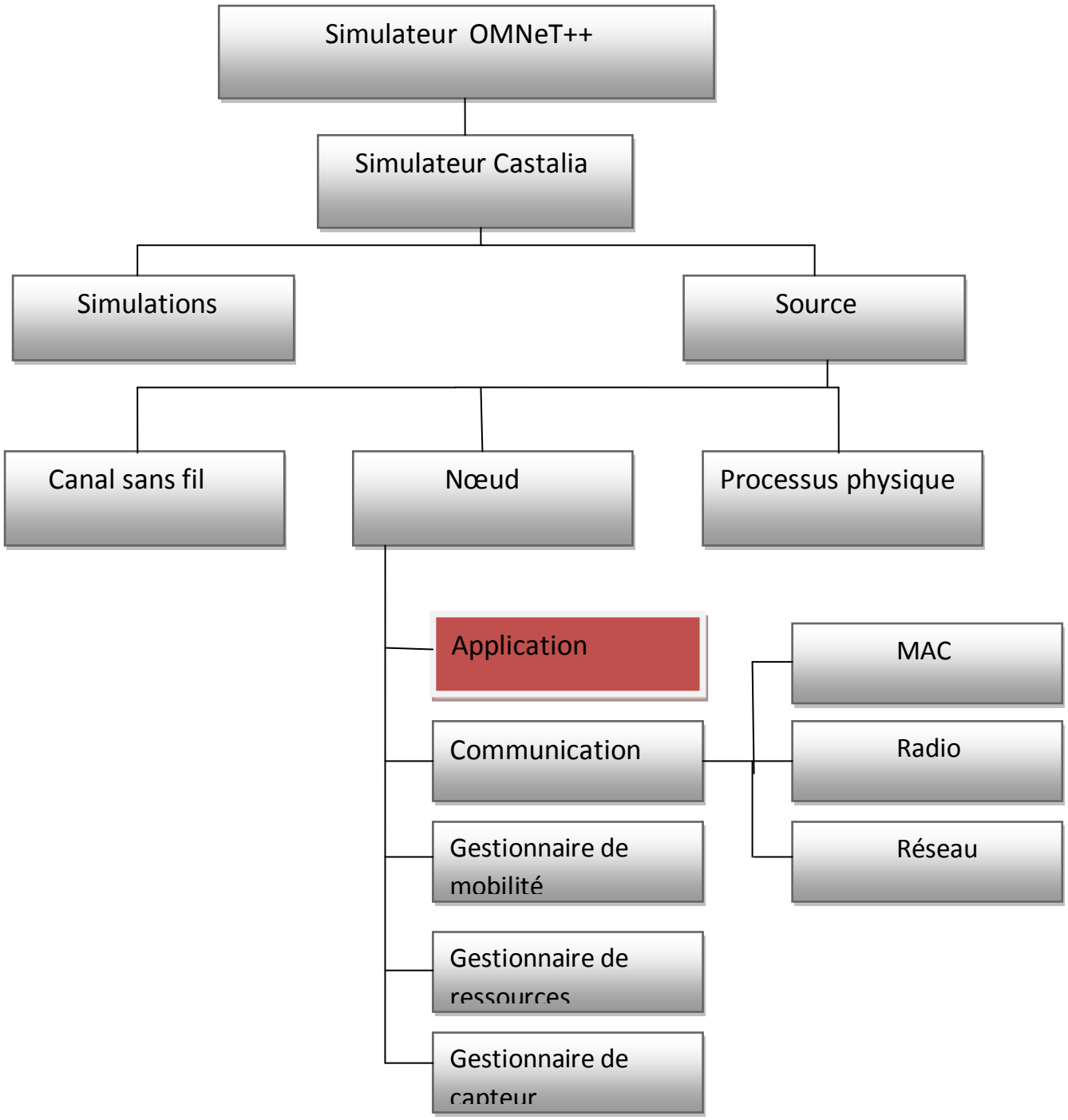


Figure IV.1 : Organigramme du simulateur Castalia.

2.2.2. Fichier « .ned » du nœud sous Castalia :

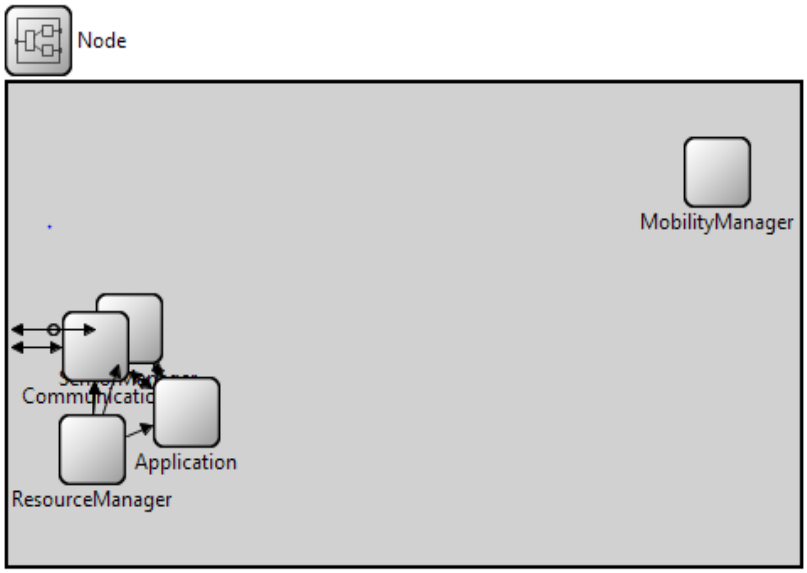


Figure IV.2 : Fichier graphique « .Ned » du nœud.

```
gates:
   output toWirelessChannel;
   output toPhysicalProcess[];
   input fromWirelessChannel;
   input fromPhysicalProcess[];

submodules:
   MobilityManager: <MobilityManagerName> like node.mobilityManager.iMobilityManager;
   ResourceManager: node.resourceManager.ResourceManager;
   SensorManager: node.sensorManager.SensorManager {
    gates:
       fromNodeContainerModule[sizeof(toPhysicalProcess)];
       toNodeContainerModule[sizeof(toPhysicalProcess)];
   }
   Communication: node.communication.CommunicationModule;
   Application: <ApplicationName> like node.application.iApplication;

connections:
   Communication.toNodeContainerModule --> toWirelessChannel;
   fromWirelessChannel --> Communication.fromNodeContainerModule;
   Application.toCommunicationModule --> Communication.fromApplicationModule;
   Application.toSensorDeviceManager --> SensorManager.fromApplicationModule;
   Communication.toApplicationModule --> Application.fromCommunicationModule;
   SensorManager.toApplicationModule --> Application.fromSensorDeviceManager;
```

Figure IV.3. Code source du fichier graphique « .Ned » du nœud.

D'après les 2 fichiers, on peut dire que le nœud dans notre simulation il contient les différentes couches suivante :

- Couche Application.
- Couche Réseau.
- Couche Physique.
- Couche Mac.

En plus de ces couches on a encore 2 sous module supplémentaire :

- Module gestion de mobilité.
- Module gestion de ressource.

Vu que notre objectif est de concevoir un nouveau module d'application alors pour tout les autres modules nécessaire au fonctionnement de notre application nous avons décidé de réutiliser des modules déjà existant et implémentés par Castalia.

2.3. Création d'un nouveau module d'application :

Pour créer notre nouveau module d'application, on ouvre **« Castalia-3.2 | src |node »**, puis bouton droit sur **« application|New|Folder».** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

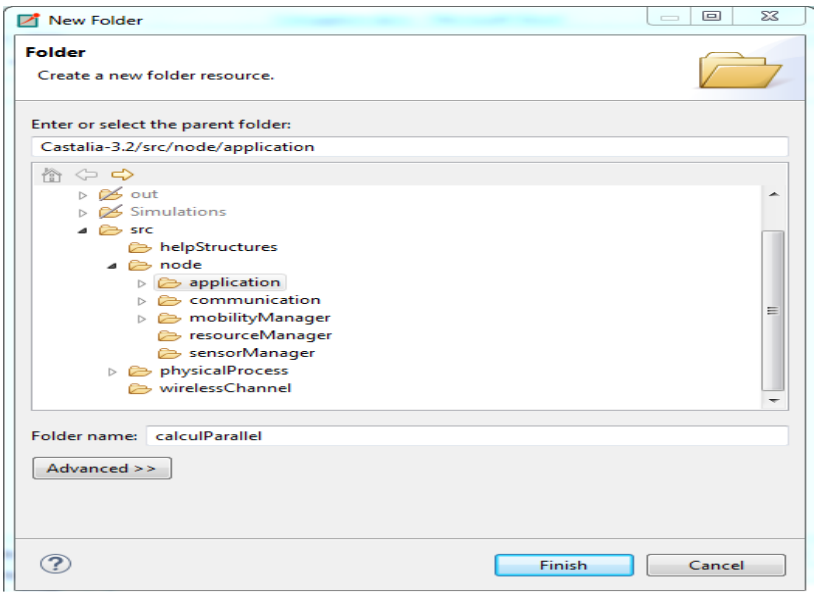


Figure IV.4. Création d'un nouveau module d'application.

2.3.1. Création des différents fichiers :

2.3.1.1. Fichier NED :

Dans ce fichier on va définir la structure de base d'un module en définissant ses portes d'entrée / sortie et ses paramètres. Nous allons définir aussi les valeurs par défaut pour les paramètres.

Pour créer notre nouveau fichier NED, on clique avec le bouton droit sur notre fichier d'application déjà créée **« calculParallel»**, puis on choisi **«New|Network Description File (Ned)»** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

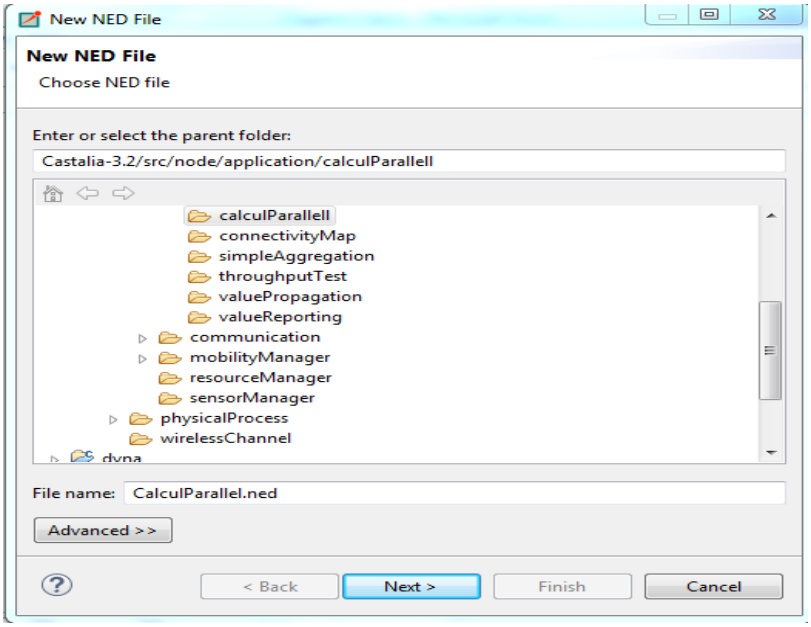


Figure IV.5 : Création d'un nouveau fichier « Ned ».

2.3.1.2. Fichier d'entête :

Pour créer notre nouveau fichier d'entête, on clique avec le bouton droit sur notre fichier d'application déjà créée **« calculParallel»**, puis on choisi **«New|Header File»** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

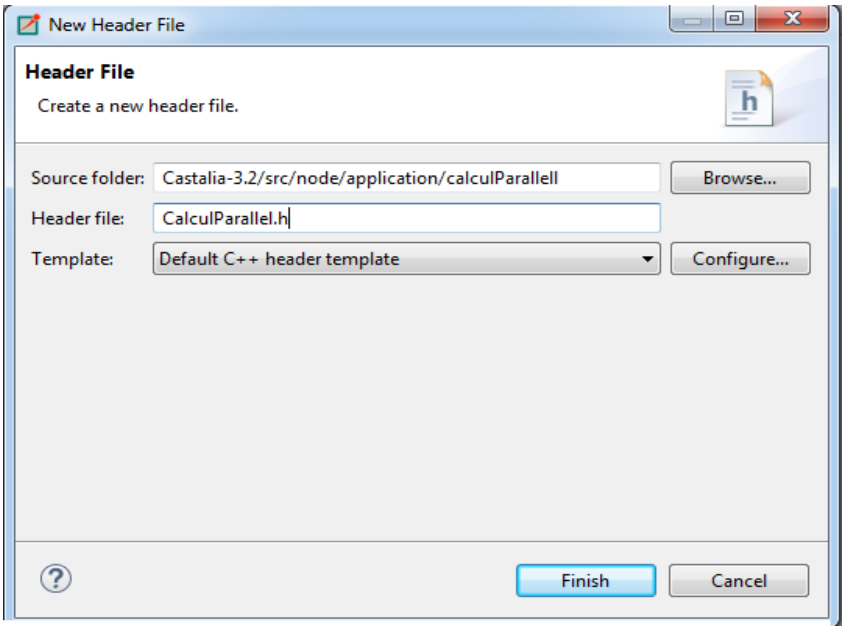


Figure IV.6 : Création d'un nouveau fichier d'entête.

2.3.1.3. Fichier source :

Pour créer notre nouveau fichier source, on clique avec le bouton droit sur notre fichier d'application déjà créée **« calculParallel»**, puis on choisi **«New|Source File »** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

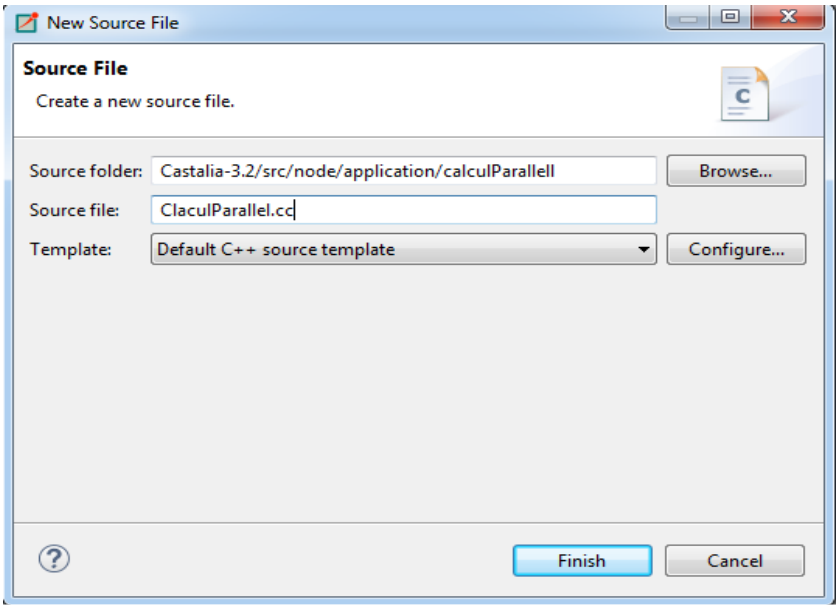


Figure IV.7 : Création d'un nouveau fichier source.

2.3.1.4. Fichier de messages :

Pour créer notre nouveau fichier message, on clique avec le bouton droit sur notre fichier d'application déjà créée **« calculParallel»**, puis on choisi **«New|Message Definition (msg) »** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

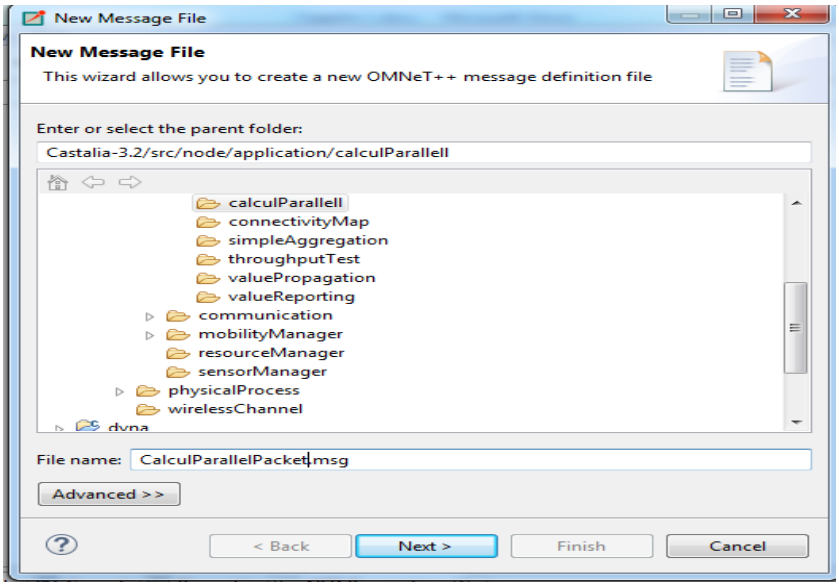


Figure IV.8 : Création d'un nouveau fichier de message.

2.3.1.5. Fichier de configuration :

Ce fichier nous permet d'attribuer des valeurs aux paramètres qu'on va utiliser dans notre simulation.

Pour créer un nouveau fichier de configuration, on ouvre **« Castalia-3.2 »**, puis on clique avec le bouton droit sur **« Simulations»,** et on choisi **« New|Initialization File (ini)».** Il ouvre un assistant dont on spécifie le nom du nouveau fichier.

Le fichier de configuration prend généralement toujours le même nom qui est **«omnetpp.ini».**

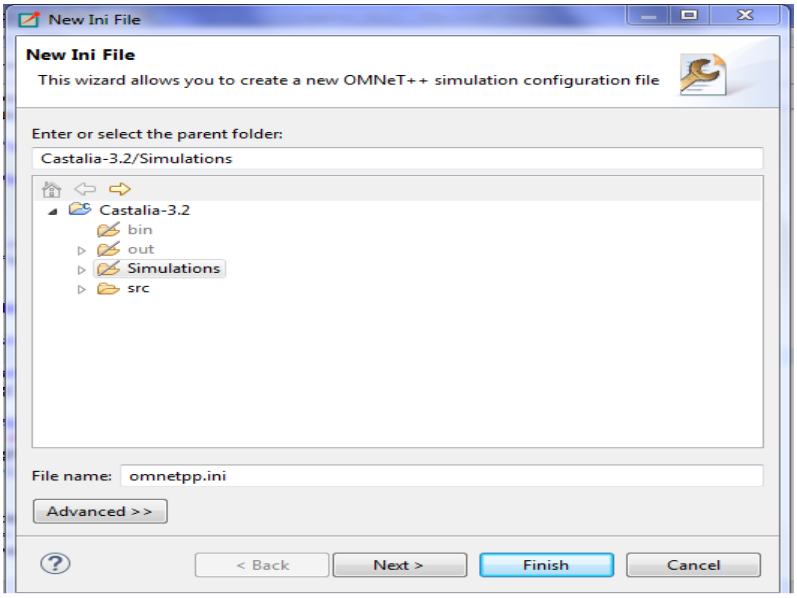


Figure IV.9 : Création d'un nouveau fichier de configuration.

2.3.2. Les paramètres utilisés dans la simulation:

Avant de lancer notre simulation, plusieurs paramètres doivent être fixés selon plusieurs facteurs comme le type de réseau de capteurs à simuler (surface, nombre de nœuds, etc.), le protocole simulé (nombre de paquets transmis, longueurs des paquets, etc.).

2.3.2.1. Paramètres du niveau supérieur

Ce sont les paramètres que l'architecture de notre réseau prend. Les paramètres tels que le temps de simulation et le nombre de nœuds n'ont pas de valeurs par défaut c'est pour cela qu'on doit les définir dans ce fichier de configuration.

- **Le temps de simulation :**

C'est le seul paramètre OMNeT générique que nous assignons dans un fichier de configuration Castalia.

Nous définissons le temps de simulation à **100 secondes**.

- **la taille du champ et le nombre de nœuds :**

Pour réaliser notre application nous avons choisi de simuler sur une surface de **100m x 100m** qui contient **4 nœuds actifs** qui sont placés dans le champ à l'aide d'une distribution aléatoire**.**

2.3.2.2. Paramètres du module :

Après avoir configuré les paramètres de niveau supérieur, nous allons définir les paramètres du module.

- **Les paramètres du canal réseau sans fil :**

- Sigma = 0
- bidirectionalSigma = 0

- **Les paramètres de routage et MAC:**

Ceci est réalisé par deux paramètres du module de communication :

- Communication.MACProtocolName ="BypassMAC"
- Communication.RoutingProtocolName="BypassRouting".

Ce sont des noms de modules qui, en substance, n'imposent aucune fonctionnalité MAC et aucune fonctionnalité de routage. Pour la simulation de notre application **calculParallel**, nous n'avons pas besoin de MAC ou de routage, c'est pour cela que nous avons laissé ces paramètres assignés à leur valeur par défaut.

- **Les paramètres du Radio :**

- Conforme aux composants Telos (CC2420) : Représente le type de radio que nous allons utiliser.
- RadioParametersFile="../Parameters/Radio/CC2420.txt": Est un fichier spécialement formaté définissant les propriétés opérationnelles de base d'une radio.
- Mode = "IDEAL"
- `TxOutputPower = "-5dBm"` : Définit la puissance que la radio transmet ses paquets.

- **Les paramètres de la gestion de mobilité :**

Nous écrivons les «emplacements de départ» des nœuds de notre réseau, dans notre cas tout les nœuds ont le gestionnaire de mobilité par défaut: **NoMobilityManager**, qui est utilisé pour décrire les nœuds statique.

- MobilityManagerName = "NoMobilityManager"

3. Réalisation de notre application :

A présent nous allons commencer à donner un aperçu sur le massage utilisé dans notre application puis les différents codes sources implémentés dans les fichiers **« .ned »**, **« .h »**, **« .cc »** et dans le fichier de configuration.

3.1. Définition de notre paquet d'application:

Pour implémenter notre paquet d'application la première étape consiste à dériver notre nouveau paquet depuis **ApplicationPacket** puisque ce dernier définit les champs utilisés par le code **virtualApp**. Voici le contenu du fichier **«ApplicationPacket .msg »** dans Castalia.

```
struct AppNetInfoExchange_type {
    double RSSI;    // the RSSI of the received packet
    double LQI;     // the LQI of the received packet
    string source;  // the routing layer source of the received packet
    string destination; // the routing layer dest of the packet to be sent
    simtime_t timestamp;    // creation timestamp of the received packet
}

// A generic application packet. If defining your own packet you have to extend
// from this packet. You do not have to use the fields already defined, and you
// can always define your own size.

packet ApplicationPacket {
    AppNetInfoExchange_type appNetInfoExchange;

    string applicationID;       // virtual app uses application ID to filter packet delivery.
    unsigned int sequenceNumber;// a field to distinguish between packets
    double data;                // a simple type to carry some data
}
```

Figure IV.10 : Code source du fichier de message « ApplicationPacket » dans Castalia.

Le code suivant représente le contenu du fichier de notre propre paquet **« CalculParallelPacket.msg »** déjà créé précédemment et qui représente notre propre paquet d'application.

```
cplusplus {{
#include "ApplicationPacket_m.h"
}}

class ApplicationPacket;

struct CalculParallelData {
    unsigned short nodeID;  //the ID of the Node
    double locX;            // x-coordinate of the node
    double locY;
    string data;        // y-coordinate of the node

}

packet CalculParallelDataPacket extends ApplicationPacket {
    CalculParallelData extraData;
}
```

Figure IV.11 : Code source du fichier de message « CalculParallelPacket ».

Après la construction de notre projet, 2 fichiers seront créé automatiquement et ajouté dans notre application qui sont : **« CalculParallelPacket_m.h »** et **« CalculParallelPacket_m.cc ».**

Ces prises d'écrans suivantes représentent les codes des fichiers **« CalculParallelPacket_m.h »** et **« CalculParallelPacket_m.cc ».**

```
// Generated file, do not edit! Created by nedtool 4.6 from src/node/application/calculParallel/CalculParallelPacket.msg.

#ifndef _CALCULPARALLELPACKET_M_H_
#define _CALCULPARALLELPACKET_M_H_

#include <omnetpp.h>

// nedtool version check
#define MSGC_VERSION 0x0406
#if (MSGC_VERSION!=OMNETPP_VERSION)
#    error Version mismatch! Probably this file was generated by an earlier version of nedtool: 'make clean' should help.
#endif



// cplusplus {{
#include "ApplicationPacket_m.h"
// }}

/**
 * Struct generated from src/node/application/calculParallel/CalculParallelPacket.msg:21 by nedtool.
 */
struct CalculParallelData
{
    CalculParallelData();
    unsigned short nodeID;
    double locX;
    double locY;
    opp_string data;
```

Figure IV.12 : Code source du fichier de message « CalculParallelPacket_m.h ».

```
// Generated file, do not edit! Created by nedtool 4.6 from src/node/application/calculParallel/CalculParallelPacket.msg.

// Disable warnings about unused variables, empty switch stmts, etc:
#ifdef _MSC_VER
#  pragma warning(disable:4101)
#  pragma warning(disable:4065)
#endif

#include <iostream>
#include <sstream>
#include "CalculParallelPacket_m.h"

USING_NAMESPACE


// Another default rule (prevents compiler from choosing base class' doPacking())
template<typename T>
void doPacking(cCommBuffer *, T& t) {
    throw cRuntimeError("Parsim error: no doPacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)));
}

template<typename T>
void doUnpacking(cCommBuffer *, T& t) {
    throw cRuntimeError("Parsim error: no doUnpacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)
}
```

Figure IV.13 : Code source du fichier de message « CalculParallelPacket_m.cc ».

3.2. Aperçu des codes sources des différents fichiers implémenté :

3.2.1. Fichier « .Ned »

```
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

package node.application.calculParallel;
// The sensor node module. Connects to the wireless channel in order to communicate
// with other nodes. Connects to psysical processes so it can sample them.
simple CalculParallel like node.application.iApplication {
 parameters:
    string applicationID = default ("calculParallel");
    bool collectTraceInfo = default (false);
    int priority = default (1);
    int packetHeaderOverhead = default (5);     // in bytes
    int constantDataPayload = default (100);    // in bytes
    bool isSink = default (false);
    double startupDelay = default (0);


     gates:
    output toCommunicationModule;
    output toSensorDeviceManager;
    input fromCommunicationModule;
    input fromSensorDeviceManager;
    input fromResourceManager;

}
```

Figure IV.14 : Code source du fichier « .Ned »

3.2.2. Fichier « .h »

```
 * CalculParallel.h

#ifndef CALCULPARALLEL_H_
#define CALCULPARALLEL_H_


#include "VirtualApplication.h"
#include "CalculParallelPacket_m.h"

using namespace std;
enum CalculParallelTimers {
    send_packet = 1

};

class CalculParallel: public VirtualApplication {
 private:

 double startupDelay;
 int dataSN;
 int sequenceNumber;
 bool sentOnce;
 int nodeID;
 int data;
 int taille;
 string var1,var2,var3;




protected:
   void startup();
   void fromNetworkLayer(ApplicationPacket *, const char *, double, double);
   void timerFiredCallback(int );
   void Calcul();

};

#endif /* CALCULPARALLEL_H_ */
```

Figure IV.15 : Code source du fichier « .h »

3.2.3. Le fichier de configuration « omnetpp.ini »

```
[General]
network = SN #le nom de notre réseaux
include ../Parameters/Castalia.ini


sim-time-limit = 100s #le temps de simulation

SN.field_x = 100    # meters
SN.field_y = 100    # meters

#  notre réseaux comprend 4 nœuds , les coordonnées de chaque noeud  seront spécifier manuellement
SN.numNodes = 4 #nombre de noeuds

# Definir les paramétres du canal sans fil
SN.wirelessChannel.onlyStaticNodes = true
SN.wirelessChannel.sigma = 0
SN.wirelessChannel.bidirectionalSigma = 0


# Using ideal parameters for wireless channel and CC2420 radio
include ../Parameters/WirelessChannel/Ideal.ini
SN.node[*].Communication.Radio.RadioParametersFile = "../Parameters/Radio/CC2420.txt"
SN.node[*].Communication.Radio.TxOutputPower = "-5dBm"
SN.node[*].Communication.Radio.mode = "IDEAL"


#spécifier le nom de l'application a simuler
SN.node[*].ApplicationName = "CalculParallel"


# application's trace info for node 0 (receiving node)
SN.node[0].Application.collectTraceInfo = true
#definir le noeud 0 comme puit
SN.node[0].Application.isSink = true


# SN.node[*].MobilityManager.collectTraceInfo = true
SN.node[*].MobilityManagerName = "NoMobilityManager"
#définir l'emplacement des différents noeuds manuellement
SN.node[0].xCoor = 50
SN.node[0].yCoor = 50
SN.node[1].xCoor = 50
SN.node[1].yCoor = 50
SN.node[2].xCoor = 50
SN.node[2].yCoor = 50
SN.node[3].xCoor = 50
SN.node[3].yCoor = 50
```

Figure IV.16 : Code source du fichier de configuration.

Conclusion :

Dans ce dernier chapitre, on a pu implémenter notre réseau de capteur sans fils après avoir présenter notre application et voir comment l'intégrer dans le simulateur Castalia sous OMNeT++.

Ce simulateur nous a permis de voir comment concevoir un modèle de simulation de réseaux de capteurs sans fils, mettre en place son architecture dont les caractéristiques sont très proches de la réalité tout en étudiant les critères de performances et choisir les paramètres convenable a notre application.

Les réseaux de capteurs constituent un axe de recherche très fertile et peuvent être appliqués dans plusieurs domaines différents. Cependant, il reste encore de nombreux problèmes à résoudre dans ce domaine afin de pouvoir les utiliser dans les conditions réelles.

L'un des problèmes qu'on peut rencontrer dans ce genre de réseau est les ressources limitées des nœuds capteur. Quand un nœud est entrain de faire un calcul, les paquets peuvent être supprimés avant que ce nœud puisse les envoyer à la destination à cause sa mémoire limitée, qui est une ressource critique dans les réseaux de capteurs sans fil. Il est donc important de mettre en place des modèles pour assurer le bon fonctionnement de ces réseaux.

Dans ce projet, nous avons mis en place un module d'application utilisant une technique de calcul réparti dont nous avons supposé que tous les nœuds capteurs sont chargés de faire une tâche précise. Au cours de ce travail qui a commencé par une étude détaillée sur les réseaux de capteurs sans fil, nous avons découvert un champ de recherche intéressant qui touche presque tous les domaines d'application. Cela nous a permis de nous initier à la recherche, et nous espérons avoir apporté une contribution, aussi petite qu'elle soit, à ce domaine qui est en pleine évolution.

Ce travail est certainement très perfectible, il faudrait procéder à sa simulation pour optimiser les paramètres susceptibles de l'être, comme les délais des timers, protocole de routage, etc, de sorte à donner les meilleurs résultats possibles.

En guise de perspective, nous allons continuer à améliorer notre travail afin qu'on puisse atteindre les résultats désirés. Nous envisageons aussi d'ajouter d'autres techniques qui prennent en considération d'autres causes de pertes d'énergies (collisions, etc.). Nous envisageons tout cela dans nos futurs travaux de recherche qu'ils soient d'ordre individuels, professionnels ou académiques.

[1] Mémoire de Magister de l'Université ummto, Spécialité Informatique, Option systèmes informatiques.

Thème : Minimisation d'énergie dans un réseau de capteurs.

Réalisé par : YOUNES Yacine.

[2] Thèse présentée pour l'obtention de doctorat d'état de l'Université d'Oran, Spécialité Informatique, Option CAO/IAO & Simulation.

Thème : Problématique de la consommation d'énergie dans les réseaux de capteurs sans fil.

Réalisé par : KECHAR Bouabdellah.

[3] Mémoire de maitrise de l'université de SHERBROOKE, Spécialité : génie électrique.

Thème : Un système de composants distribué pour les réseaux de capteurs sans fil.

Réalisé par : Frédéric SUREAU.

[4] Thèse de Doctorat de l'Université d'Oran, Spécialité Informatique, Option sécurité informatique.

Thème : Surveillance distribué pour la sécurité d'un réseau de capteurs sans fil.

Réalisé par : BENAHMED Khelifa.

[5] Mémoire de Master recherche 2 de l'université IFSIC-Rennes 1, Spécialité Informatique.

Thème : Optimisation de la consommation d'énergie dans les réseaux de capteurs sans fil.

Réalisé par : Sofiane MOAD.

[6] Mémoire Master de l'université de Bechar, Spécialité Informatique et Télécommunications, Option sécurité informatique.

Thème : Approche distribuée pour la sécurité d'un réseau de capteurs sans fils (RCSF).

Présenté par : Nadia BOUNEGTA.

[7] Mémoire de Magister de l'université Hadj Lakhder – Batna, Spécialité Informatique, Option Ingénierie des Systèmes d'Informatiques.

Thème : Traitement et Transfert d'images Par Réseau de Capteurs sans Fil.

Réalisé par: Adel CHOUHA.

[8] Thomas Chamberlain: Learning OMNeT++.

[9] András Varga: The OMNeT++ Discrete Event Simulation System. In the Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic.

[10] Thèse présentée pour l'obtention du titre de Docteur de l'Université Henri Poincaré, Nancy I, Spécialité Automatique, Traitement du signal et Génie Informatique.

Thème : Contribution à la modélisation de produit actif communiquant, spécification et évaluations d'un protocole de communication orienté sécurité des produit.
Réalisé par : Ahmed ZOUINKHI.

[11] Master Informatique Technologies de l'Internet à Université de Pau et des Pays de l'Adour.
Thème : Implémentation d'une couche MAC (accès au support) dans OMNET++/Castalia pour des réseaux de capteurs pour la surveillance.
Réalisé par : DOS SANTOS Leonel et LALANNE Clément.

[12] Thèse en vu d'obtention du Doctorat de l'Université de Toulouse.
Délivré par : Institut National des Sciences Appliquées de Toulouse (INSA de toulouse).
Thème : Modeling, simulation and implementation of an 802.15.4 based adaptive communication protocol in wireless sensor network: application to monitoring the elderly at home.
Réalisé par : Juan LU.

[13] Mémoire Magister : Informatique, Option : Analyse, Commande et surveillance des Systèmes.
Thème : Analyse Graphique pour la surveillance dans un réseau de capteur sans fils (RCSF) Simulateur : OMNeT++.
Réalisé par: Leila Imane Niar.

[14] A thesis submitted in partial fulfilment of the requirements for the degree of Master of Computing. A Reliability Evaluation of Wireless Sensor Network Simulator: Simulation vs. Testbed.
Réalisé Par: ZhengYi Guan (Alex).

[15] Mémoire présenté à l'Université du QUÉBEC À TROIS-RIVIÈRES.
Thème : Gestion De L'anonymat et de la traçabilité dans les réseaux véhiculaires sans fil.
Réalisé par: ADETUNDJI ADIGUN.

[16] Comparison of OMNET++ and Other Simulator for WSN Simulation Chapitre : Comparaison d'OMNET ++ et d'autres Simulateurs pour une Simulation WSN
Sun Limin, Li Jianzhong, Chen Yu, "Wireless Sensor Networks", *Tsinghua University Press*, 2005.

[17] Réseaux sans fil et mobiles, Khaldoun Al Agha, Bibliothèque de département informatique, RES 251\1.

[18] Simulation des réseaux, Monique Becker, André-Luc Beylot, Bibliothèque de département informatique, RES255\1.

[19] Réseaux mobiles : Adhoc et Réseaux de capteurs sans fil, Houda Labiod , Bibliothèque de département informatique, RES 254\1.

[20] OMNeT++, Installation Guide, Version 4.6, Copyright © 2014 András Varga and OpenSim Ltd.

[21] "OMNET++—Discrete Event Simulation System, Version 4.6 User Manual", Copyright © 2014 András Varga and OpenSim Ltd.

[22] Castalia A simulator for Wireless Sensor Networks and Body Area Networks Version 3.2 User‟s Manual, Athanassios Boulis March 2011

[23] Castalia-3.2 Port to OMNeT++ on Windows and Linux, Installation Guide, Authors: Alex Lacerda, Marcella Pinheiro.

[24] https://forge.nicta.com.au/frs/?group_id=301.

[25]www.commentcamarche.net

[26] http://www.omnetpp.org/

[27] http://compalg.inf.elte.hu/~tony/Informatikai-Konyvtar/03 Algorithms%20of%20Informatics%201,%202,%203/Network29May.pdf

[28] http://www.promodel.com/onlinehelp/ProModel/80/C-03%20 %20Terminating%20Versus%20%20Non-terminating%20Simulations.htm

[29] http://www.ccs-labs.org/teaching/nwsim/2012w/01-network-simulation.pdf

[30] http://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm

[31] http://www.palisade.com/risk/fr/simulation_monte_carlo.asp

[32] https://fr.wikipedia.org/wiki/Simulation_r%C3%A9seau

[33] https://sourceforge.net/projects/gnuplot/