Modélisations, Simulations, Synthèses pour des réseaux dynamiques de capteurs sans fil

Pierre-Yves Lucas

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
UFR Sciences et techniques
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École doctorale SICMA

présentée par
Pierre-Yves Lucas
Préparée au Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) — CNRS, UMR 6285

Modélisations, simulations et synthèse pour des réseaux dynamiques de capteurs sans fil

Thèse soutenue le 13 décembre 2016,
devant la commission d'examen composée de :

Christophe CAMBIER
Maître de conférences, IRD Campus France Nord, Bondy / *rapporteur*
Onil GOUBIER PERSADA
Docteur, Cirela, Velizy-Villacoublay / *examinateur*
Laurent Tchamnda NANA
Professeur des Universités, Université de Bretagne Occidentale, Brest / *président du jury*
Congduc PHAM
Professeur des Universités, Université de Pau et des Pays de l'Adour, Pau / *rapporteur*
Alain PLANTEC
Professeur des Universités, Université de Bretagne Occidentale, Brest / *examinateur*
Bernard POTTIER
Professeur des Universités, Université de Bretagne Occidentale, Brest / *directeur de thèse*
Serge STINCKWICH
Docteur, IRD Campus France Nord, Bondy / *rapporteur*

Université de Bretagne Occidentale

UNIVERSITE BRETAGNE LOIRE

Brest MÉTROPOLE

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
UFR Sciences et techniques
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École doctorale SICMA

présentée par

Pierre-Yves Lucas

Préparée au Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) — CNRS, UMR 6285

Modélisations, simulations et synthèse pour des réseaux dynamiques de capteurs sans fil

Thèse soutenue le 13 décembre 2016,
devant la commission d'examen composée de :

Christophe CAMBIER
Maître de conférences, IRD Campus France Nord, Bondy / *rapporteur*

Onil GOUBIER PERSADA
Docteur, Cirela, Velizy-Villacoublay / *examinateur*

Laurent Tchamnda NANA
Professeur des Universités, Université de Bretagne Occidentale, Brest / *président du jury*

Congduc PHAM
Professeur des Universités, Université de Pau et des Pays de l'Adour, Pau / *rapporteur*

Alain PLANTEC
Professeur des Universités, Université de Bretagne Occidentale, Brest / *examinateur*

Bernard POTTIER
Professeur des Universités, Université de Bretagne Occidentale, Brest / *directeur de thèse*

Serge STINCKWICH
Docteur, IRD Campus France Nord, Bondy / *rapporteur*

Table des matières

Préface

*Les travaux présentés ici ont été effectués au département d'informatique de l'*Université de Bretagne Occidentale (UBO), *laboratoire* LabSTICC.

Leur financement était assuré par une allocation de thèse de Brest Métropole Océane (BMO) *que nous remercions très vivement pour son soutien.*

Ils ont eu lieu dans un petit groupe travaillant de manière solidaire sur les réseaux de capteurs sans fil, avec des objectifs très différents, tels que la propagation des sons dans les structures urbaines (Eloi Keita) ou le comportement des ravageurs en Afrique (Mahamadou Traoré).

Ce groupe, ainsi que mes travaux de thèse, ont été animés par Bernard Pottier, Professeur d'informatique à l'UBO, en relation avec d'autres équipes françaises et internationales formant une véritable communauté de recherche tournée vers l'observation de l'environnement. Plusieurs des travaux présentés ici vivent dans des projets en activité, en France et dans des cadres internationaux.

1

Réseaux de capteurs : interfaces entre environnement et systèmes d'information

1.1 Observation pervasive et systèmes d'information

Parmi les évolutions récentes les plus nettes dans le domaine des sciences de l'information, on peut retenir la multiplication des dispositifs capables de communiquer et de percevoir des grandeurs physiques. Ces dispositifs existent en grand nombre, ils sont bien plus répandus que les micro-ordinateurs. On qualifie ces systèmes de plusieurs caractéristiques remarquables : leur abondance, leur miniaturisation, leur intelligence locale et leur capacité de communication qui leur permet d'avoir une intelligence collective, distribuée.

L'abondance. L'usage des systèmes informatiques a évolué au cours des années. Les premiers ordinateurs étaient très coûteux et encombrants, ils étaient principalement utilisés dans les entreprises. Dans les années 1960, un tel système informatique se présente sous la forme de *mainframe*, un ordinateur central dont la puissance de calcul est partagée entre plusieurs utilisateurs [1]. Les machines évoluent suivant une tendance : les composants se miniaturisent, leur capacité de calcul augmente et leur coût diminue. Dans les années 1980, chaque utilisateur peut disposer de son propre matériel, qui devient un ordinateur personnel (PC pour *personal computer*). La tendance se poursuit et dans les années 2000, les systèmes sont si miniaturisés qu'on peut les transporter : ils deviennent mobiles. Désormais on embarque l'informatique dans les objets, ce qui leur apporte une valeur ajoutée. On peut donner l'exemple des téléphones intelligents (*smartphones*), ou des tablettes qui ont des capacités de calcul pour traiter des flux multimédia. Ou encore, dans les véhicules, les systèmes électroniques agissent sur plusieurs organes tels que le moteur, les freins, les suspensions, pour améliorer la conduite. Chaque utilisateur dispose ainsi de plusieurs ordinateurs, dont il n'a pas forcément conscience de son utilisation. L'informatique devient invisible, « ubiquitaire » [1]. Pour comprendre cette évolution, Marc Weiser la compare à celle des moteurs électriques : au début du XX^{e} siècle, un seul moteur entraînait tous les outils d'une usine par l'intermédiaire de courroies et de poulies. Des moteurs plus

1. Une des premières visions du domaine (1943) amenait cette citation de T.J.Watson : *« I think there is a world market for maybe five computers. »*

petits, efficaces, moins coûteux, ont permis de donner à chaque outil sa propre force motrice, puis de placer plusieurs moteurs dans une seule machine.

La transparence. L'augmentation de la finesse de gravure des circuits permet leur miniaturisation. La technologie des micro-systèmes électromécaniques (MEMS) rend maintenant possible la conception de systèmes de taille inférieure au millimètre. On est maintenant amené à employer le terme de « poussière intelligente » (*smartdust*).

L'intelligence locale. Ces systèmes sont intelligents, car ils sont couplés à leur environnement, et ils agissent sur lui en retour, par l'intermédiaire de capteurs physiques et d'actionneurs. Par exemple le système anti-blocage (ABS) d'un véhicule mesure la vitesse de rotation des roues et intervient sur le circuit de frein pour empêcher le dérapage des pneus lors du freinage. C'est grâce à ces capteurs physiques que le système acquière une capacité à *prendre conscience* de lui-même, de l'environnement dans lequel il est placé. L'automatique, ou la cybernétique groupent les méthodes permettant de contrôler ces réactions scientifiquement.

Il est maintenant possible de connaître sa position absolue sur Terre, en utilisant les techniques de positionnement par satellite, comme le GPS (global positionning system, États-Unis), Glonass (Russie), ou Galileo (Europe). Ces systèmes sont suffisamment précis pour situer le récepteur à une précision de quelques mètres, voire quelques centimètres. Les modules de réception de ces signaux sont courants dans les systèmes électroniques mobiles : on les trouve dans les téléphones, les assistants à la conduite…

L'intelligence distribuée. Ces systèmes à base de capteurs sont capables de communiquer entre eux, tout comme les ordinateurs personnels au sein d'un réseau. Cela est rendu possible grâce aux transmetteurs radio à basse consommation électrique et à faible coût. Ces capteurs en réseau forment un nouveau système, distribué, qui donne une nouvelle vision de l'environnement perçu. Ainsi, les capteurs peuvent se coordonner pour réaliser une tâche de plus haut niveau, par exemple une mesure plus précise que celle établie avec un seul capteur.

Cette intersection entre l'environnement et l'informatique forme ce que Edward Lee nomme les systèmes « cyber-physiques ». L'informatique distribuée n'est plus limitée au traitement des données, elle devient capable d'interagir avec l'environnement et de provoquer des modifications de comportements très profonds [2].

Ce sont les réseaux de capteurs sans fil qui permettent d'obtenir des solutions distribuées à des problèmes urbains ou environnementaux de dimensions et d'échelles diverses. Le centre de recherche coopératif université-industrie *Berkeley Sensor and Actuator Center (BSAC)* recense ainsi un grand nombre de projets d'étude et de conception des capteurs physiques, des circuits logiques et des transmissions sans fil [3].

Edward Lee soulève une problématique afférente aux *systèmes cyber-physiques* (CPS) [2] : ils associent en un seul concept (1) les systèmes embarqués en réseau et (2) l'environnement qu'ils surveillent et sur lequel ils effectuent un contrôle. L'environnement intervient à son tour sur l'exécution des calculs de ces systèmes embarqués, à travers une boucle de rétro-contrôle. La concurrence des événements est présente dans le monde physique, et l'on va retrouver cette même concurrence dans les abstractions des processus et des réseaux.

1.2 Une référence pour les applications : le stationnement urbain

Au plus fort de la crise économique de 2008, un réseau de capteurs sans fil pré-industriel a ainsi été déployé avec succès à San Francisco par *Streetline Inc.* [4], une entreprise californienne créée en

2005. L'objectif est d'étudier et valider une aide au stationnement des véhicules. En trois ans, une centaine de villes ont adopté ce système.

À San Francisco, ce sont 6 000 places de parking qui sont équipées d'un capteur et d'une radio fixés sur la chaussée. Le capteur est sensible à la présence d'une masse métallique et peut percevoir quand un véhicule occupe l'emplacement, ou le libère. L'information est transmise par radio et relayée par les capteurs voisins jusqu'à un système central. Pour stationner son véhicule, il faut interroger le central avec un téléphone mobile, qui indique la position et la destination voulue. Le conducteur est ensuite guidé par GPS jusqu'à l'emplacement libre qu'il a choisi.

Il y a d'abord un gain de temps pour se garer, et donc moins de trafic généré par les véhicules en recherche de stationnement. Il y a ensuite une économie de carburant et enfin, moins de pollution. Ces capteurs servent aussi à mesurer l'utilisation des emplacements de stationnement, afin d'ajuster les tarifs en fonction de l'offre et de la demande.

FIGURE 1.1 – San Francisco : pose d'un capteur sur la chaussée.
(source : The New York Times [5])

Cette expérience est significative de changements profonds dans l'instrumentation de l'environnement. L'intérêt pour les réseaux de capteurs sans fil qui récoltent et diffusent l'information se renforce. Ces réseaux deviennent le point de départ de développements industriels majeurs, qui permettent des économies de ressources importantes : du temps et du carburant comme le montre l'exemple du stationnement à San Francisco.

Cet exemple illustre la boucle réalisée par les réseaux de capteurs : capture locale, transport réseau, synthèse de l'information, décision, retour à l'usager sur une situation auparavant imperceptible.

1.3 Une description holiste pour les réseaux de capteurs sans fil

Notre travail de thèse a démarré par un apprentissage des techniques de base : architecture d'un capteur programmable, interface des capteurs, gestion des données et de la mémoire embarquée, intégration dans un réseau radio et contraintes qui en découlent, gestion distribuée des données et contrôle distribué. À ceci s'ajoute la partie intégration dans les systèmes d'information distribués, avec un point de repère important qui est le positionnement géographique. L'interface avec les systèmes d'information géo-localisées est une question majeure car elle conditionne le calcul des

points de contrôle et la validation des données prélevées. Cette partie a donné lieu à la création de logiciels originaux, d'abord *GMap*, avec lequel nous avons pu représenter les services de la *Smart City* de Santander, puis *Quickmap*, version évoluée qui permet de travailler sur des tuilages géographiques arbitraires et qui sert de point d'entrée aux logiciels du laboratoire de l'UBO (*Pickcell*).

Nous introduisons d'abord les éléments matériels d'un capteur sans fil générique : sa structure et son fonctionnement. Nous décrivons ensuite le fonctionnement des réseaux de capteurs sans fil. Le contexte de l'intégration des données est décrit en section 1.4.

1.3.1 Organisation d'un capteur sans fil

Un *capteur sans fil* est un micro-système composé de quatre grandes parties (figure 1.2) :

1. **Un ou plusieurs interfaces de mesures physiques, appelés capteurs physiques :**
   Ces appareils permettent d'effectuer des mesures physiques de leur environnement, comme par exemple :
   - capteurs de température, humidité, pression ;
   - capteurs de bruit (décibel) ;
   - capteurs de position géographique ;
   - capteurs de gaz : CO, CO_2 , H_2S. ils peuvent être utilisés dans des contrôles de pollution urbaine, contrôle de processus industriel et chimique ;
   - capteurs d'événements : les capteurs de mouvements, détecteurs d'ouverture, utilisés pour la détection d'intrusions ;
   - capteurs de radiations : niveau de rayonnement ;
   - capteurs de comptage : mesure d'énergie, de consommation d'eau,… qui peuvent être utilisés pour la détection de fuites, lors du stockage de liquide pour détecter le niveau des réservoirs de stockage, pour des automatismes industriels, ou l'irrigation agricole.

   Ces capteurs physiques permettent de mesurer le « monde réel ». L'information physique est convertie en signal électrique, par exemple une différence de potentiel, qui est fonction de la grandeur physique mesurée. Cette tension est ensuite convertie en une donnée numérique pour être analysée par l'unité de calcul, transmise, stockée.

   La qualité des capteurs physiques est dépendante de leur sensibilité avec par exemple un seuil de détection, de la précision de leurs mesures, de l'étalonnage et de la stabilité des mesures dans le temps.

   Comme les processeurs, les capteurs physiques sont l'objet d'efforts de miniaturisation et d'intégration. Des catégories de capteurs physiques, appelés MEMS (ou *microelectromechanical system*), répondent bien à ces besoins. Les besoins en terme de capteurs paraissent sans fin et suscitent des efforts de nature stratégique.

2. **Un émetteur-récepteur radio :**
   La communication numérique par radio s'opère entre des capteurs voisins les uns des autres. Ces communications transportent les données de proche en proche, mais aussi permettent des diagnostics distribués, ou la prise de décision. C'est dans ce sens que l'on peut dire que les capteurs sans fil s'organisent en réseau. Les communications radio sont aussi un domaine en forte évolution, et on trouve un grand choix de solutions à petite, moyenne et grande échelle.

   Les émetteurs de signaux radio se distinguent par leur puissance d'émission, ainsi que par la fréquence qu'ils utilisent. Les fréquences élevées (supérieures au gigahertz) sont plus facilement atténuées par les obstacles (bâtiments, forêt, …) que les basses fréquences (inférieures

au gigahertz). Les réseaux urbains ou internes *maillés* sont ainsi fréquemment conçus dans la bande de 2,4 GHz, alors que les réseaux environnementaux de longue portée utilisent des fréquences de 800 à 900 MHz. Les portées typiques sont respectivement quelques centaines de mètres et plusieurs kilomètres.

Les différences entre les matériels se situent également au niveau de la sensibilité de réception, de la modulation utilisée, qui a une incidence sur le débit des données. Lorsque celui-ci est faible, le signal est moins sensible aux interférences. Le choix de l'antenne permettra de capter dans toutes les directions, dans le cas d'une antenne omnidirectionnelle, ou de privilégier une direction dans l'espace avec une antenne directive. La direction d'émission peut avec certains émetteurs être choisie et commutée dynamiquement.

Toutes ces caractéristiques déterminent la portée des communications, c'est-à-dire la distance maximale qui peut exister entre l'émetteur et le récepteur le plus éloigné. Il s'agit d'un élément important à considérer pour le déploiement de réseaux de capteurs.

Le débit de données requis est lié à la quantité de données à transmettre ; il est nécessaire de choisir les caractéristiques de l'émetteur-récepteur en conséquence.

L'émetteur-récepteur est un des éléments du capteur sans fil qui est susceptible de consommer beaucoup d'énergie. Or, plus l'émetteur sera puissant, plus sa consommation d'énergie sera importante… cela doit donc être pris en compte lors du choix de l'émetteur.

La sécurité des transmissions est également un point critique, la protection des échanges entre capteurs doit être assurée, comme dans tout système informatique, mais avec d'autant plus d'attention que les capteurs rendent un service critique (véhicules, par exemple).

3. **Une source d'énergie :** une batterie, un panneau solaire, ou un système de récupération d'énergie à partir par exemple de différence de température, de vibrations, …

   Ce composant fournit l'électricité nécessaire à l'alimentation des éléments du capteur : le capteur physique, l'émetteur-récepteur, et l'unité de calcul. Les sources d'énergie se distinguent par leur capacité, c'est-à-dire la quantité de courant qu'elles peuvent fournir, ainsi que par la puissance de crête atteinte. La consommation d'énergie du système détermine son autonomie.

4. **Une unité de calcul :**

   Ce composant collecte les mesures brutes du capteur physique, réalise le traitement de ces données. Il contrôle l'activité ou la mise au repos du capteur sans fil. Il gère les communications avec les capteurs voisins, qui sont à sa portée.

   C'est souvent un micro-contrôleur qui est à la base de l'unité de calcul. Il est programmé par micro-logiciel : cet algorithme permet de traiter les données brutes locales, et intégrer les données globales du réseau pour prendre une décision collective, distribuée.

   Les micro-contrôleurs sont des composants autonomes, qui rassemblent dans un seul circuit intégré, des composants essentiels d'un ordinateur :
   - un processeur, cadencé à une fréquence donnée,
   - une mémoire vive,
   - une mémoire de stockage (de type flash) qui contient le programme à exécuter,
   - des périphériques d'entrées-sorties pour agir avec le monde extérieur (communication, conversion entre signaux analogiques et numériques),
   - des composants dédiés à certaines tâches (timer, DMA) pour accélérer les traitements.

   La fréquence du processeur, ainsi que la capacité des mémoires et la consommation d'énergie, caractérisent ce composant. Ils sont généralement conçus pour une faible consommation d'énergie : de 1 à 10 milliampères en activité et quelques micro-ampères en veille.

Ils sont programmés par micro-logiciel, ce qui assure une souplesse de fonctionnement, à la différence d'un composant câblé, pour qui le fonctionnement n'est pas modifiable.

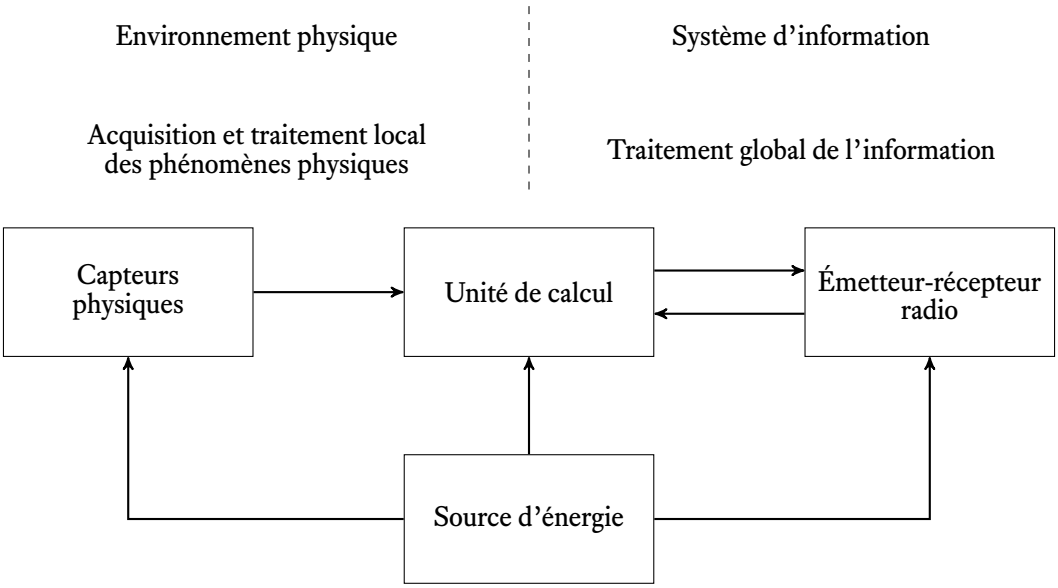


FIGURE 1.2 – Vue d'ensemble de l'architecture d'un capteur sans fil

Les capteurs se répandent grâce aux progrès réalisés dans la miniaturisation et l'intégration des systèmes. On peut ainsi surveiller une grande quantité de paramètres : température, pression, humidité, niveau de bruit, éclairage, déplacement de véhicules. Il devient possible de produire, à bas coût et en grande quantité, ces dispositifs de mesure.

1.3.2 Mise en réseau

Grâce à un émetteur-récepteur radio à faible puissance, le capteur échange des informations avec ses plus proches voisins. Il s'établit un réseau communicant, pour réaliser des tâches de plus haut niveau.

Au sein d'un réseau, on appelle « nœud » le capteur sans fil élémentaire. Un réseau peut être constitué d'un nombre variable de capteurs sans fil, qui peut aller d'une dizaine à plusieurs milliers de capteurs.

Lorsqu'un capteur est à portée d'un autre, il s'établit un lien de communication entre eux. L'ensemble des liens de communication forme la topologie du réseau. Il existe plusieurs types de topologies :

- **des topologies avec hiérarchie :**
  Certains nœuds ont des fonctions particulières pour centraliser l'information qui provient des autres nœuds. C'est ce nœud coordinateur qui a en charge d'initialiser et de maintenir sur le réseau les autres dispositifs qui communiquent directement avec lui. On parle de réseau en étoile (star, LoRa) ou de réseau arbre (tree, ZigBee).
- **des topologies sans hiérarchie :**
  Tous les nœuds du réseau sont équivalents : ils peuvent tous recevoir et traiter l'information qui leur parvient. On parle alors de communication distribuée. Il s'agit de la topologie maillée (mesh, 802.15.4, Xbee).

La topologie sans hiérarchie présente plusieurs avantages, en terme de :

- circulation de l'information : il n'y a pas de passage obligé de l'information par le nœud coordonnateur, ce qui élimine les goulots d'étranglements. De plus, en cas de défaillance d'un nœud, la circulation globale de l'information dans le réseau n'est pas affectée (tolérance aux pannes).
- dynamique du réseau : il est possible d'ajouter ou retirer des nœuds à un réseau sans en perturber son fonctionnement.
- mobilité des nœuds : le déplacement est permis au sein du réseau.
- résilience : l'absence de hiérarchie permet la réorganisation des communications entre les nœuds, et donc le réseau est capable de poursuivre son fonctionnement en cas de défaillance d'un nœud.

Tout cela confère une très forte fiabilité à un tel réseau.

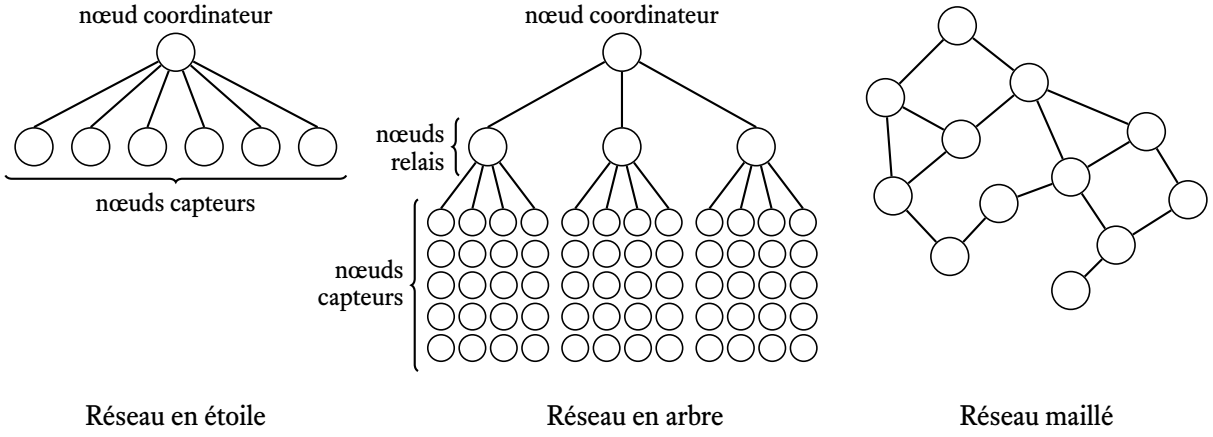


FIGURE 1.3 – Trois topologies différentes : les réseaux en étoile et en arbre sont hiérarchisés, tandis que les réseaux maillés ne le sont pas.

1.4 Intégration des données dans l'environnement géo-localisé

Depuis une dizaine d'années environ, l'utilisation des informations géo-localisées s'est considérablement accrue dans notre environnement quotidien. Ce phénomène est concomitant avec l'évolution des usages de l'informatique, en particulier sa miniaturisation. Outre l'utilisation courante du GPS en voiture, les consommateurs se voient aujourd'hui proposer des offres commerciales en fonction de leur localisation géographique. Ce secteur est en plein développement, et de nombreux sites Web fournissent des données géo-localisées, tel que Google, Bing, Mappy, OpenStreetMap et Géoportail (la cartographie de l'IGN depuis les 200 dernières années).

Bien connaître l'espace géographique est primordial pour le déploiement des réseaux de capteurs sans fil : cela permet de les positionner de façon idéale pour recueillir les informations souhaitées, par exemple des relevés de température à des endroits clés. Les capteurs ayant été déployés à des endroits identifiés, cela permet de recevoir d'eux une information géo-localisée. Il est alors possible d'alimenter le système d'information avec les données collectées.

La connaissance de l'environnement est permise par la cartographie et les systèmes d'information géographiques (SIG).

1.4.1 La cartographie, une base pour la connaissance de l'environnement

Les bases de données géographiques rassemblent des données associées à une position sur Terre. Les coordonnées de cette position sont définies par un système de localisation géographique, le plus connu étant le GPS (Global Positioning Sytem, mis au point par les États-Unis). D'autres systèmes de géo-localisation existent, comme le système russe GLONASS, le système indien IRNSS et le système européen Galileo.

Les bases de données renferment des informations de différentes natures. Il peut s'agir :

- de données cartographiques de base : elles incluent les routes et autoroutes, les limites administratives, les noms de communes, les cours d'eau, les forêts, bois et d'une façon générale toutes les informations habituelles disponibles sur une carte.
- de données sur le climat, la topographie, mais aussi la démographie, les transports, les habitudes de consommation, les entreprises…

Les SIG permettent de rassembler, d'organiser et d'analyser ces différentes informations disposant d'une localisation spatiale, sous forme de cartes. Leur impact est actuellement très important.

Les données géométriques contenues dans les bases de données sont présentées sous la forme de coordonnées x, y, z.

On distingue trois types d'objets :

- les points d'intérêt, représentés par un simple triplet ;
- les segments (routes, rivières), sont représentés par une succession de coordonnées x, y, z ;
- les polygones (contour de bâtiments, limite de territoire), sont représentés par une succession de coordonnées délimitant une surface fermée.

Une donnée est caractérisée par plusieurs champs d'informations :

- la forme géométrique de l'objet (point, segment, polygone) ;
- la nature de l'objet : bâtiment, route, cours d'eau ;
- la représentation de l'objet sur la carte : couleur, épaisseur du trait ;
- la date d'acquisition de la donnée et l'auteur.

Les données sont présentées par couches : il est ainsi possible de superposer sur une carte des informations climatiques avec des limites de pays, par exemple. Cela permet d'obtenir une carte adaptée à la demande, en y faisant figurer uniquement les informations souhaitées et en supprimant celles qui sont inutiles.

Ces objets sont exportés dans un format de fichier qui peut être relu par les autres logiciels de SIG.

1.4.2 Une base de donnée géographique participative : OpenStreetMap

OpenStreetMap est un projet qui rassemble des données géographiques du monde entier et les diffuse, libres de droit. Ce sont les contributions des utilisateurs qui enrichissent le fonds, en ajoutant des données ou par des mises à jour. Ces données sont de différentes natures : routes, bâtiments, forêts, cours d'eau, … Il est possible de consulter la carte produite par OpenStreetMap, ou de récupérer des données pour construire sa propre carte.

Nous nous sommes tournés vers la cartographie numérique participative d'OpenStreetMap, car d'une part ces données sont en accès libre, et d'autre part chacun peut contribuer à enrichir la carte, ce qui augmente la précision et la nature des données y figurant. Enfin, les données des cartes étant périssables, le nombre important de contributeurs permet aussi leur mise à jour régulière et rapide. Depuis le démarrage de ce projet en 2004, le nombre d'utilisateurs enregistrés progresse de

manière exponentielle : en 2009, on compte 100 000 utilisateurs, en 2012, 500 000 utilisateurs, en 2013, 1 050 000 inscrits et 2 200 000 en 2015. Le nombre croissant d'utilisateurs de ce projet est le reflet de l'intérêt porté aux données cartographiques. De même, les données collectées augmentent considérablement : de 500 millions de points GPS en 2009, le projet totalise 5 000 millions en 2015. Les administrations utilisent également ce moyen pour rendre publiques leurs bases de données : par exemple, la communauté urbaine Brest Métropole y a ajouté les contours des 80 000 bâtiments de son territoire.

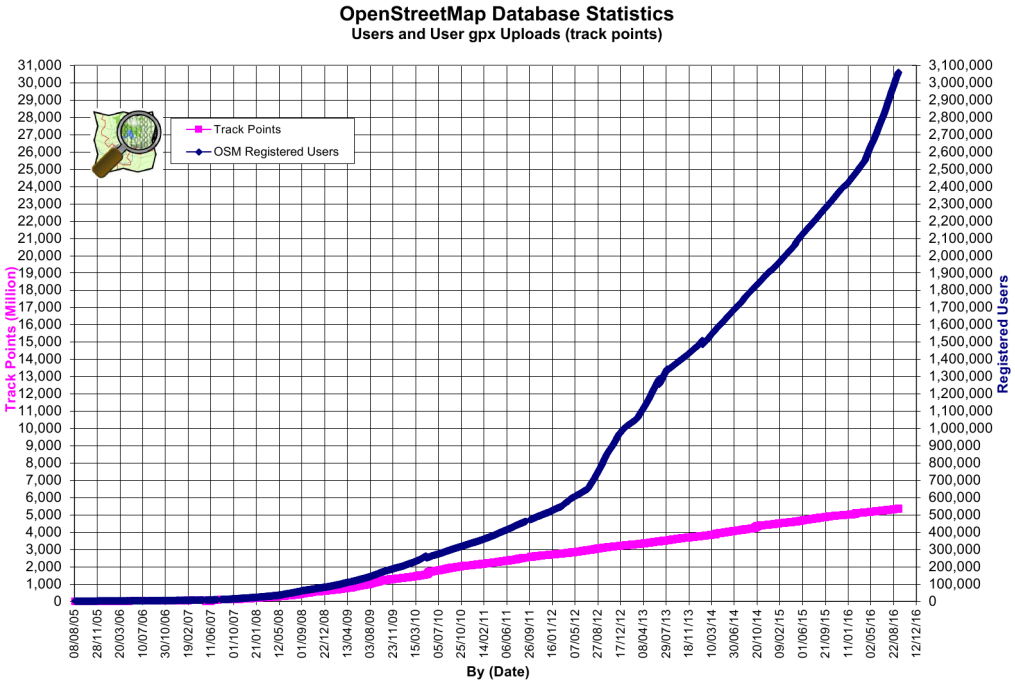


FIGURE 1.4 – Depuis 2008, le nombre d'utilisateurs enregistrés d'OpenStreetMap est en forte progression *(courbe bleue)*. Les contributions *(en rose)* augmentent de manière régulière. *(source : Statistiques des bases de données d'OpenStreetMap [6].)*

1.4.3 Usages pour les mobiles et réseaux de capteurs

L'informatique se miniaturise, elle se trouve dans un grand nombre d'appareils, disséminés dans notre environnement. On peut prendre comme exemple les téléphones intelligents (*smartphones*), qui sont de plus en plus répandus. Ces téléphones comportent plusieurs capteurs, par exemple : GPS, accéléromètre, compas. En les utilisant, les applications proposent de nouveaux services tels que la navigation routière, l'orientation d'antennes paraboles.

Ainsi, l'usage de l'informatique n'est plus limité au calcul de données : elle devient capable de percevoir l'environnement dans lequel elle se trouve, et d'agir physiquement sur lui. La géolocalisation est le point de rencontre sémantique entre le mobile et la perception fixe, réalisée par le système de capteurs.

1.5 Structure des contributions et du mémoire

Le travail que nous avons effectué et qui est décrit dans ce mémoire peut se répartir en découverte technologique, et en apports scientifiques. Ces travaux se sont situés dans un projet interne de l'Université de Brest, dans une composante du LabSTICC créée en 2008 pour couvrir le domaine des réseaux de capteurs sans fil (Wireless Sensor Networks, WSN).

La première année de la thèse (2010-2011) a été une année de découverte durant laquelle nous avons pris en main des outillages de base permettant la réalisation d'applications. Les petits systèmes créés à partir de micro-contrôleurs basse consommation, provenant des constructeurs Texas Instrument (TI, MSP430), à partir des architectures modulaires reconfigurables de Cypress, puis les circuits de AtMel utilisés dans les composants Arduino, et plus tard les Raspberry PI, proposant un grain plus élevé et supportant Linux.

Avec ces composants, nous avons pu créer des réseaux radio, en utilisant les services de circuits radio présents sur les cartes (CC2420 et MSP430) ou ajoutés aux systèmes (modules Digi XBee et Arduino). Ceci a permis d'explorer quelques réseaux radio, dont 802.15.4.

Enfin la partie capteur a été également étudiée, en ajoutant des modules discrets permettant de mesurer des données atmosphériques, les accélérations, le positionnement géographique, ceci dans un contexte de mobilité.

Ces aspects de nos travaux sont décrits dans le chapitre 2 de ce document. Ils témoignent des investissements et des compétences acquises en amont des développements suivants. L'intégration logicielle des WSN a également été prise en charge lors de ces premières années, avec, par exemple, les suivis de mobiles et les transcriptions géographiques de ces suivis.

Nous avons beaucoup travaillé sur ces aspects intégration logicielle, et en particulier la géo-localisation, et le calcul des placements de capteurs radio a servi à alimenter les synthétiseurs de code de NetGen. La conception des outils géographiques en amont a été effectuée en deux temps, avec l'outil GMap décrit en annexe C, puis dans une évolution en *brosseur* de carte appuyé sur les serveurs de tuiles, QuickMap, décrit en 3.

L'intégration de ces outils dans le flot NetGen est présentée sur la figure 4.1. On voit comment le géo-positionnement est important pour la description des modèles objets Smalltalk qui sont ensuite lus et transcrits par les générateurs de simulateurs. L'intégration récente (2016) dans le système d'information de l'équipe WSN est présentée sur la figure 3.12. Cette évolution re-dessine le fonctionnement des outils autour d'une base de données supportant OpenStreetMap et d'autres schémas d'informations. Nous avons développé QuickMap qui est un navigateur spécifique pour les réseaux de capteurs, en amont d'autres outils d'analyse cellulaire et des simulateurs.

Le chapitre 4 présente le flot général de NetGen, il sert d'articulation entre les aspects frontaux, géo-localisés, et deux développement que nous proposons :

- en chapitre 5 pour la synchronisation entre des visiteurs mobiles, par exemple un satellite LEO [2] et les réseaux de capteurs [7]. Ces travaux ont été réalisés en 2015. L'exemple de la récolte aérienne est intéressant pour des zones distantes, toutefois les méthodes utilisées peuvent s'appliquer à des récoltes à moindre échelle, par exemple en utilisant les circuits de transports en commun montrés figures 3.8 et 3.9.
- en chapitre 6, en ce qui concerne l'intégration de contributions locales, au sens de la capture physique de l'information, et de contributions distribuées, telles qu'elles sont présentées en chapitre 5. Il s'agit ici de s'appuyer sur le modèle des processus communicants pour décrire

2. Low Earth Orbiter

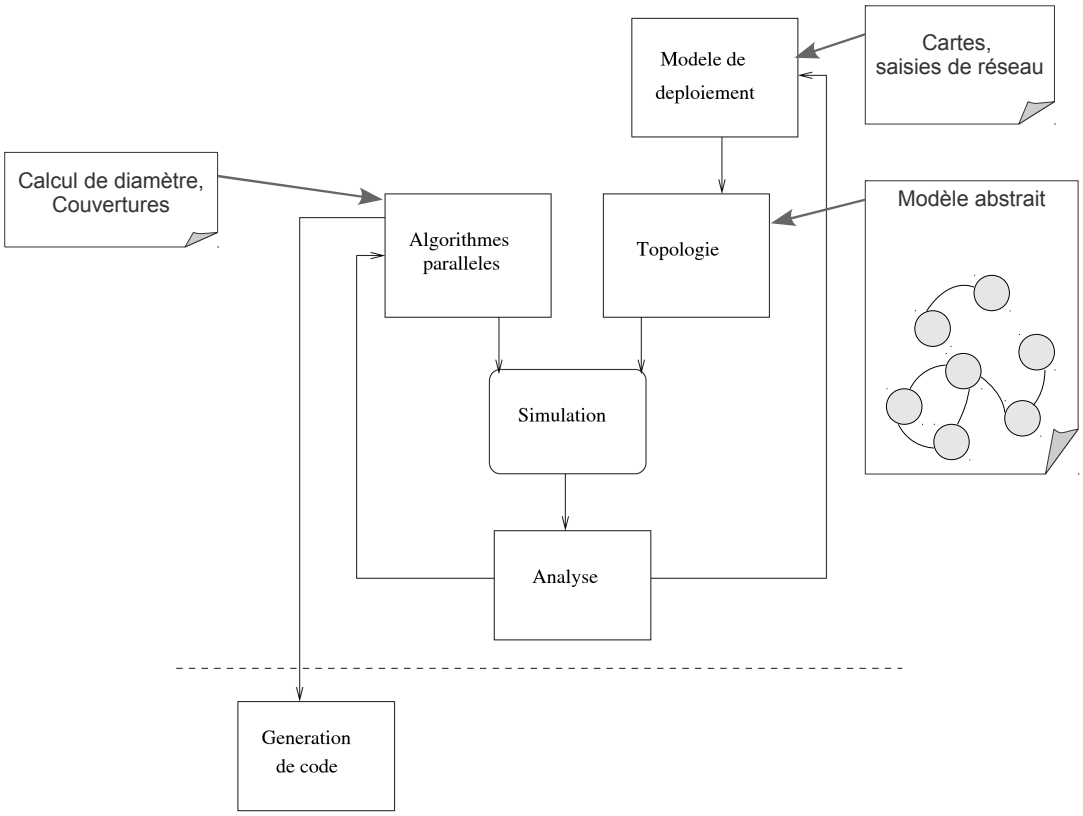


FIGURE 1.5 – Le flot de conception NetGen simplifié pour les réseaux de capteurs : présentations graphiques et géographiques en haut à droite (QuickMap, chapitre 3), modèle structurel généré (Topologie en NetGen, chapitre 4) et comportements spécifiés (Algorithmes, chapitre 5) au centre, intégration de code embarqué, en bas (chapitre 6).

à la fois le code du capteur et le système distribué formé par les WSN. L'idée présentée est d'utiliser des machines virtuelles pour effacer la complexité et l'hétérogénéité des outils de développement.

À l'issue de cette thèse, nous pouvons revendiquer des compétences techniques avérées dans le chapitre 2, et une série d'apports originaux, en géo-localisation pour les WSN, intégré à un flot de conception complet, des contributions originales en mobilité et interactions aériennes, et en intégration logicielle portable.

2

État de l'art technologique et réalisations de référence

Ce chapitre décrit plusieurs réalisations qui ont permis de comprendre le domaine des réseaux de capteurs, en largeur, du capteur physique au système distribué, et en profondeur, en s'intéressant de près à plusieurs volets techniques interdépendants.

On y trouvera :

1. les descriptions de plates-formes de développement utilisées : Arduino+XBee, MSP430+CC2420,
2. l'intégration de capteurs (accéléromètres), l'acquisition, le traitement et le conditionnement des données,
3. le géo-positionnement, avec la saisie de points d'intérêts, et de traces de parcours de mobiles,
4. l'intégration et l'analyse de l'accès au lien radio dans un micro-contrôleur, et pour une machine virtuelle embarquée,
5. l'analyse de la consommation d'énergie en émission sur le lien radio,
6. l'analyse de puissance des signaux radio sur une zone géographique,
7. l'analyse du protocole radio.

Ces réalisations d'outils ou d'applications ont donné lieu à des développements logiciels et matériels. Les résultats ont permis de prendre pied dans des contextes complexes : prise en charge de visiteurs mobiles (chapitre 5), capteurs mobiles (section 2.2.3), systèmes de développements évolués intégrant les mesures de consommation (chapitre 6), appréhension de la géo-localisation dans les applications (chapitre 3).

2.1 Trois plates-formes pour le prototypage

Nous allons décrire ici les matériels que nous avons utilisés pour élaborer des applications ou des outillages. Ce sont par exemple le capteur-bouée « Mersea », un système embarqué qui permet de

réaliser la mesure de couverture radio d'un émetteur, ou un outil permettant de déporter la mesure de température et d'humidité sur un interface internet.

L'appréhension et la compréhension technique des fonctionnements locaux ont été obtenus dans ces réalisations, qui ont aussi permis de produire des données géo-localisées, grâce aux GPS interfacés.

2.1.1 La carte SoftBaugh

Cette carte de prototypage est commercialisée par la société SoftBaugh. Elle regroupe trois éléments d'un capteur sans fil : le micro-contrôleur, le module radio et l'alimentation, qui se fait par deux batteries situées dans un logement sous la carte (voir figure 2.1). Elle permet de mettre au point des programmes pour les capteurs sans fil.

Architecture matérielle

L'organisation de la carte est centrée autour d'un micro-contrôleur Texas Instruments MSP430 très basse consommation. Il est relié à un transmetteur radio Chipcon CC2420, lui aussi basse consommation, par un bus de communication. L'ajout des capteurs physiques est aisé grâce aux broches qui permettent de relier directement le capteur aux connecteurs du micro-contrôleur. Ces capteurs peuvent être analogiques (thermomètre, hygromètre, accéléromètre), ou numériques (module GPS, boussole électronique). Un régulateur d'alimentation permet d'utiliser deux batteries de taille AAA, pour alimenter le système. Le chargement d'un programme sur la carte est fait en utilisant le port JTAG, à l'aide d'un boîtier de programmation.

Cette carte est produite d'après une note d'application de Texas Instruments. Une autre version a été réalisée par l'équipe du CAIRN de Lannion pour la plate-forme matérielle PowWow.

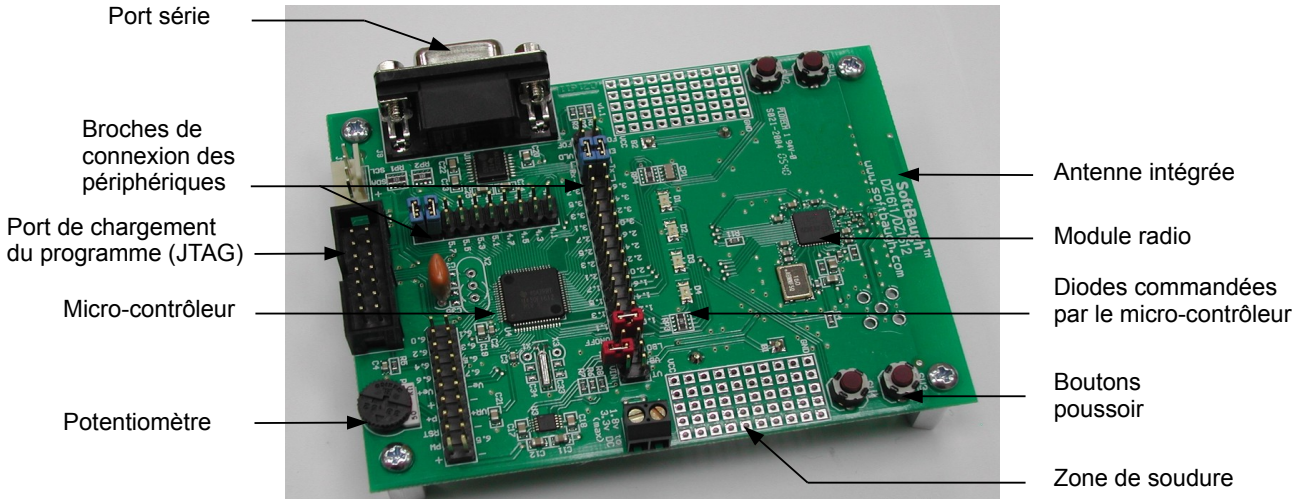


FIGURE 2.1 – Vue d'ensemble de la carte de prototypage SoftBaugh DZ1612.

Outils de développement

Les applications conçues pour ce micro-contrôleur sont programmées en langage C, avec des extensions spécifiques pour utiliser certaines ressources, par exemple les interruptions. Celles-ci sont propres aux micro-contrôleurs : elles sont un moyen pratique d'être averti de l'arrivée d'un

signal à l'entrée du micro-contrôleur. La tâche en cours est stoppée (interrompue) pour effectuer le traitement de l'information reçue, puis elle reprend une fois le traitement terminé.

Le compilateur utilisé est MSPGCC, il est spécifique à ce micro-contrôleur. À partir du programme, il produit le code binaire en veillant à ce que les différentes parties du code soient chargées aux bons emplacements de la mémoire.

Ensuite, ce code est copié dans la mémoire flash du micro-contrôleur à travers une interface spécifique, appelée JTAG (Joint Test Action Group), grâce au logiciel MSPDEBUG. Ce dernier est capable de contrôler l'exécution du programme et de relire les données en mémoire pour aider à la mise au point de l'application.

2.1.2 Arduino, une famille de cartes sous licence libre

Les cartes de développement Arduino ont un plan de fabrication publié sous licence libre. Elles se composent d'un micro-contrôleur et de composants complémentaires, pour programmer la carte sans matériel spécifique. Les connecteurs des cartes Arduino respectent tous une même disposition. Cela permet d'adapter des cartes d'extension universelles qui apportent des nouvelles fonctionnalités : par exemple un émetteur radio, ou des platines d'expérimentation qui supportent les montages des capteurs supplémentaires. Une communauté active conçoit de nouvelles cartes et met leur patron de conception à disposition. Plusieurs étages de cartes additionnelles peuvent se superposer à la carte initiale.

Architecture matérielle

Le composant principal est un micro-contrôleur Atmel AtMega2560 basse consommation. Les broches du circuit sont accessibles sur des connecteurs placés en périphérie de la carte. L'alimentation se fait par un adaptateur secteur ou par le port USB. La programmation est faite par le port USB.

La disposition des connecteurs permet d'empiler des cartes filles, appelées « shield », pour ajouter des extensions.

Par exemple, on ajoute la fonctionnalité de transmission radio avec un shield XBee. C'est un module radio autonome, qui utilise un micro-logiciel de type propriétaire pour fonctionner.

Deux versions du logiciel sont disponibles :

- un protocole de transmission compatible 802.15.4,
- un protocole propriétaire, appelé Digimesh, pour construire des réseaux maillés.

Les transmissions radio sont faites sur la même bande de fréquence que le circuit CC2420, si bien que ce dernier peut être utilisé pour capturer les trames émises par les modules.

Ce module est commercialisé par Digi.com.

Outils de développement

L'environnement de développement intégré (IDE) Arduino permet de concevoir le programme, de le compiler puis de le charger sur la carte depuis la même interface graphique. Il est basé sur le langage Processing. Le code minimal se compose de deux fonctions :

- une fonction setup, qui initialise le matériel,
- une fonction loop, qui exécute en boucle la même séquence de code.

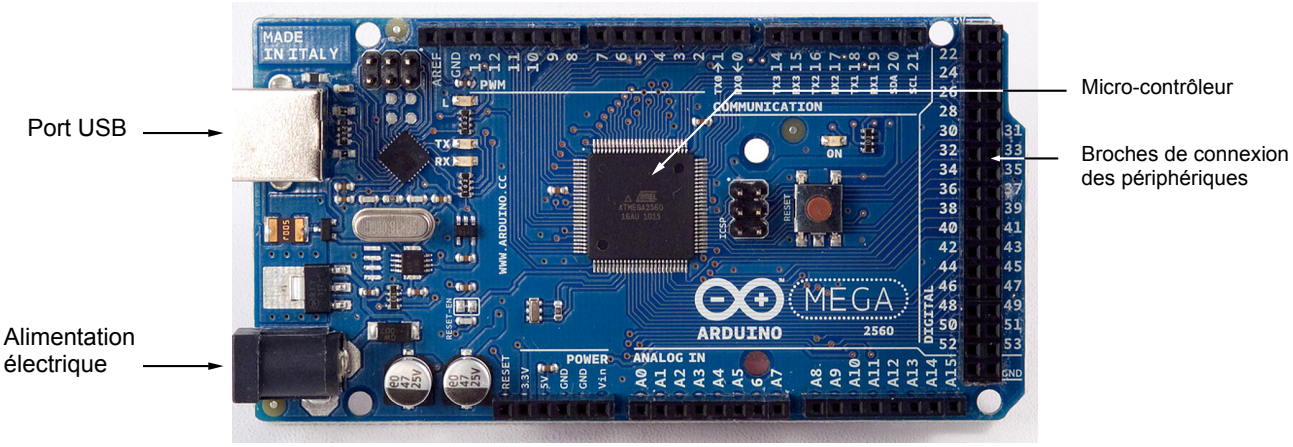


FIGURE 2.2 – Vue d'ensemble de la carte Arduino Mega 2560.

2.1.3 Le système Waspmote de Libelium

C'est un système complet pour développer des applications de capteurs sans fil. Il comporte une carte principale, de petite taille (7 cm × 5 cm), avec le micro-contrôleur, un émetteur-récepteur radio, un large choix de capteurs physiques compatibles, que l'on place sur une plaque adaptée à la carte principale. Une batterie de grande capacité (2 400 mAh) alimente le système (figure 2.3).

Architecture matérielle

Le composant principal est un micro-contrôleur Atmel AtMega1281 basse consommation. Deux capteurs sont déjà montés sur la carte : un capteur de température et un accéléromètre. Un emplacement sur la carte est prévu pour ajouter un émetteur radio, ainsi qu'une carte d'extension qui accueille les capteurs physiques. L'alimentation se fait par la batterie intégrée. Celle-ci se recharge via le port USB ou bien un panneau solaire. La carte se programme par le port USB.

Outils de développement

L'environnement de développement est également fourni, comme pour la famille de cartes Arduino. Des bibliothèques de codes sont disponibles pour utiliser facilement les ressources matérielles. Le principe de programmation est identique aux cartes Arduino.

2.2 Applications en acquisition et géo-positionnement

Nous avons utilisé la carte SoftBaugh pour réaliser d'abord un outil de saisie géographique interactive pour des positions de capteurs (section 2.2.1), puis un prototype d'une bouée-capteur dérivant appelée Mersea, destinée à mesurer les courants et la hauteur des vagues (section 2.2.3).

Une autre réalisation est la mesure de couverture radio d'un émetteur (section 2.2.2). Les applications réalisées sur les cartes Arduino et Libelium sont la mesure de température et d'humidité qui ont servi à des démonstrations publiques.

S'agissant de capteurs sans fil, les performances en transmission doivent être évaluées et chiffrées en terme de consommation d'énergie. La section 2.3 donnera des indications sur les analyses qui ont été effectuées dans le cas du système Arduino+Xbee programmé en Occam interprété, et dans le cas

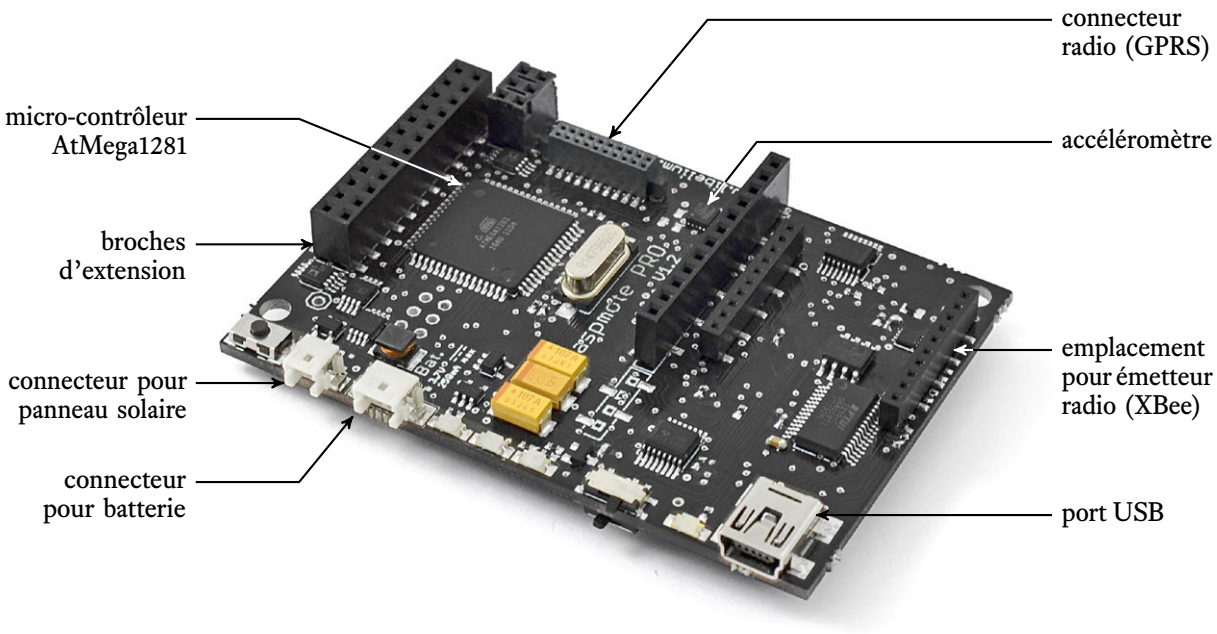


FIGURE 2.3 – Description de la carte Waspmote (documentation technique de Libelium [8]). Les platines d'expérimentation (capteurs, émetteur radio) sont ajoutées en mezzanine. On remarque qu'un panneau solaire peut alimenter le système.

de la plate-forme MSP430+CC2420, avec la décomposition énergétique du protocole radio. Cette section permet aussi d'introduire deux techniques de programmation que nous avons utilisées : le codage concurrent avec Occam, et le codage de bas niveau sur les interfaces de micro-contrôleurs basse consommation.

Cette analyse quantitative a été complétée par un outil permettant d'observer le lien radio de manière non-intrusive. Cet outil a été développé en appui sur un logiciel libre *Wireshark*, nous l'avons donc baptisé *wireless-shark* (section 2.4).

2.2.1 Outil de saisie d'un réseau de capteurs à l'aide d'un GPS

Un système MSP430 est relié à un récepteur GPS, de façon à enregistrer en continu sa position. Celle-ci est enregistrée sous forme d'un quadruplet de coordonnées longitude, latitude, altitude et date. Des points d'intérêt peuvent être mémorisés via un bouton. Nous avons testé ce système en parcourant une ligne de bus, et en sauvegardant les positions des arrêts. De retour au laboratoire, la mémoire de la carte est relue sur une station de travail, et un fichier contenant les positions géographiques est produit et procure des facilités de saisie pour les réseaux.

Le parcours peut être représenté par trois moyens différents :

- sur Google Earth,
- dans un logiciel de système d'information géographique, QGis,
- dans NetGen.

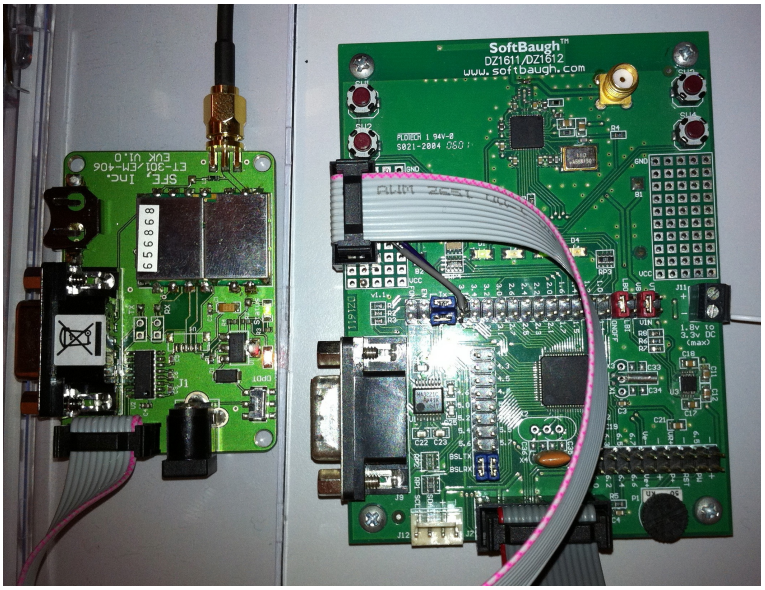


FIGURE 2.4 – Un assemblage sur table du MSP430 et de son GPS (à gauche). La communication est opérée par un port série implanté via le fil en nappe, le protocole est textuel et proche de la norme NMEA. Ce système dispose de deux communications radio, l'une pour les signaux satellites, à gauche, et l'autre pour les communications entre capteurs, à droite et en haut.

2.2.2 Répartition spatiale de puissance radio

L'idée derrière cette investigation est d'effectuer des relevés de portée que l'on utilisera pour extrapoler des couvertures, pour ensuite construire des modèles théoriques de couverture. Deux systèmes embarqués MSP430 ont servi à produire un outil de mesure de la couverture d'un émetteur radio à l'aide d'un GPS. Le dispositif se constitue d'une partie fixe, reliée à une station de travail (un ordinateur portable) qui reçoit les informations de la partie mobile. Cette dernière est connectée à un récepteur GPS : elle émet sa position, répétitivement. Le système fixe extrait la puissance et la qualité du signal reçu, qu'elle cartographie. La figure 2.5 présente la courbe de puissance lue sur le chemin parcouru, alors que l'afficheur de droite présente le déplacement 2D du système mobile.

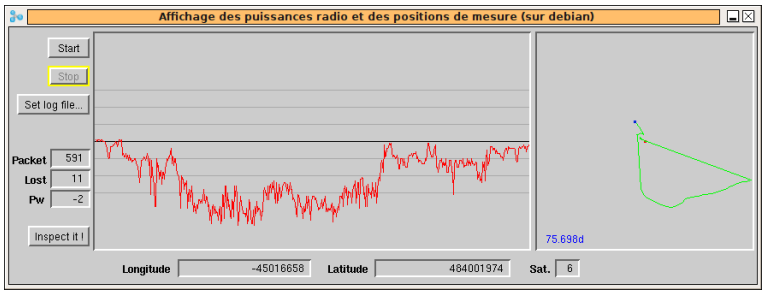


FIGURE 2.5 – L'interface de mesure des puissances radio : à gauche les niveaux de puissance sur le chemin, à droite la carte 2D des déplacements.

K. Ammouche [9], a montré en appui sur ce travail, qu'une interpolation des mesures effectuées permet d'extrapoler la couverture grâce à des calculs d'enveloppe convexe. L'échantillonnage discret réalisé peut permettre de retrouver des valeurs intermédiaires avec une certaine fiabilité.

2.2.3 Mersea : un capteur-bouée

L'objectif de cette réalisation est de disposer d'une maquette multi-capteurs, afin de pouvoir étudier des relevés de fréquence variable, dans le cadre d'une synthèse système (ordonnancement des acquisitions et des communications). Le projet Mersea est une bouée équipée d'un système embarqué avec les capteurs suivants :

- un récepteur GPS, qui permet de connaître le déplacement de la bouée et de mesurer la vitesse de sa dérive.
- un accéléromètre, qui détecte les mouvements de plus faible amplitude, non décelables par GPS, comme la houle, ce qui permet de mesurer la hauteur des vagues ;
- un compas, qui renseigne sur l'orientation de la bouée.

La bouée est laissée à la dérive ; équipée d'un émetteur radio, cela lui permet de communiquer avec un opérateur situé dans un bateau, éventuellement par l'intermédiaire d'autres bouées. Sa batterie assure une autonomie suffisante pour quelques heures de fonctionnement.

Les données collectées nous permettent de connaître l'état de la mer, comme la vitesse du courant ou la force des vagues. Le module GPS est lu via un UART et permet de recevoir des positions géographiques périodiquement.

L'accéléromètre

Nous disposons d'un accéléromètre trois axes : il mesure la valeur de l'accélération dans trois directions orthogonales de l'espace. Cet accéléromètre est un capteur analogique : la tension à ses bornes est proportionnelle à l'accélération qu'il subit. C'est un capteur miniaturisé de type MEMS (*microelectromechanical system*).

Les bornes de sortie du capteur sont reliées au micro-contrôleur en utilisant les convertisseurs de signal analogique en signal numérique. Trois convertisseurs sont ainsi utilisés. La numérisation du signal analogique est faite par échantillonnage : périodiquement, une mesure de la tension est prise et convertie en un nombre entier, avec une précision de 12 bits. La fréquence d'échantillonnage est réglée à 45 Hz environ : les trois convertisseurs sont déclenchés simultanément, ensuite les résultats sont lus dans le registre de chaque convertisseur, puis stockés en mémoire. À la fin de l'expérience, les données sont relues à partir de la mémoire flash pour tracer la courbe des accélérations enregistrées. Le résultat est présenté ci-dessous.

On configure les périphériques du MSP430 à travers des registres de contrôle. La lecture du registre indique l'état dans lequel il se trouve. Ainsi, les périphériques à configurer sont :

- le timer,
- l'ADC.

Configuration de l'ADC. Une interruption est un signal déclenché par un périphérique pour annoncer un événement au processeur. Celui-ci arrête le traitement en cours et exécute un autre programme : le service d'interruption.

Le code ci-dessous programme le service d'interruption du timer afin de déclencher la conversion analogique vers numérique. L'accès aux périphériques est fait au travers de registres, en lecture ou écriture. Ainsi, le registre `ADC12CTL0` contrôle le convertisseur 12 bits n° 0 et la valeur `ADC12SC` déclenche une conversion.

```
1 // Timer A0 interrupt service routine
2 interrupt (TIMERA0_VECTOR) TimerA0InterruptServiceRoutine(void)
```

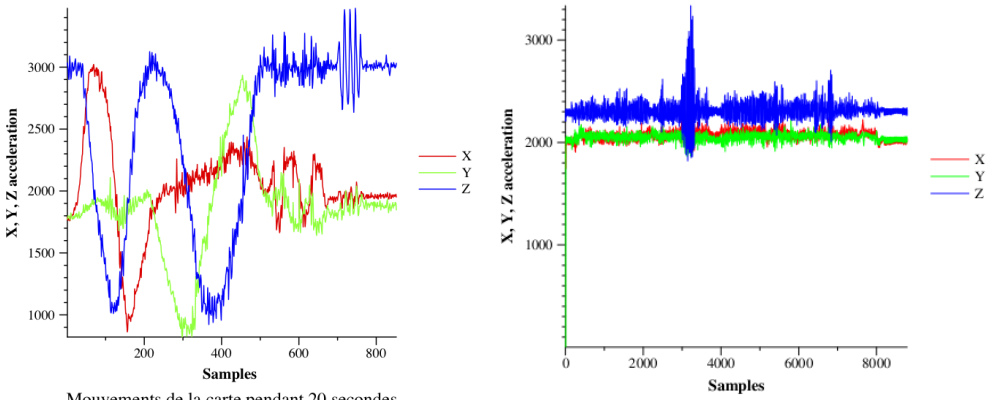


FIGURE 2.6 – Deux mesures de l'accéléromètre et deux cas très différents pour les outils de synthèse et le système. À droite, le système est porté par un marcheur qui descend un escalier, puis fait quelques pas en courant. L'unité de l'axe des ordonnées est arbitraire.

```
3 {
4             ADC12CTL0 |= ADC12SC ;   // Start conversion
5 }
```

L'intégration système est donc particulièrement scabreuse et se prête bien à des styles de programmation concurrente, comme Occam, qui sera discuté en chapitre 6.

Lecture du résultat. À la fin de l'expérience, on relit la mémoire flash, et on trace la courbe des accélérations enregistrées. Le résultat est présenté figure 2.6.

Le compas

C'est un détecteur de champ magnétique. Deux capteurs disposés orthogonalement mesurent le champ magnétique terrestre. Un micro-contrôleur calcule la direction du nord à partir des deux mesures et produit en sortie un signal PWM (*pulse width modulation*, modulation codée par largeur d'impulsion). La durée de l'impulsion positive est proportionnelle à l'angle (le cap).

On utilise la carte SoftBaugh pour interpréter ce signal et stocker les valeurs du cap en mémoire flash. La mémoire est ensuite relue sur un ordinateur pour afficher une courbe.

La valeur positive du signal dure entre 1 ms et 36.99 ms. Les acquisitions sont conduites par les interruptions.

2.3 Analyse des transmissions radio sur Arduino/TVM et MSP430

La motivation est le contrôle temporel et énergétique lors de transmissions radio 2.4 GHz sur des matériels Arduino/XBee et MSP430/CC2420. On montre une approche analytique du coût et des performances de transmission radio sur une machine virtuelle Occam, puis une analyse de l'énergie consommée lors de l'exécution du protocole CSMA-CA sur la seconde plate-forme.

2.3.1 Protocole de mesure

Le matériel

On utilise une carte Arduino ATMEGA 2560, comportant un microcontrôleur Atmel AVR 2560, et un module Xbee. Le module Xbee se place sur une carte d'alimentation intermédiaire (shield), couplée sur la carte Arduino : la communication entre le MCU et XBee est faite par un lien série (UART).

Le montage est alimenté par des accumulateurs en série, qui fournissent une tension continue de 6 V. Une résistance de 1 ohm est placée en série dans le montage. En mesurant la tension à ses bornes avec un oscilloscope, on obtient la valeur instantanée de l'intensité du courant. Ce montage permet de visualiser les variations de consommation d'énergie du microcontrôleur et du module Xbee. De cette façon, on peut connaître quelles sont les périodes d'activité et de veille du MCU, et les charges induites par la radio.

La mesure de l'intensité s'écrit :

$$U = R \cdot I, \text{ avec } R = 1\,\text{ohm}.$$

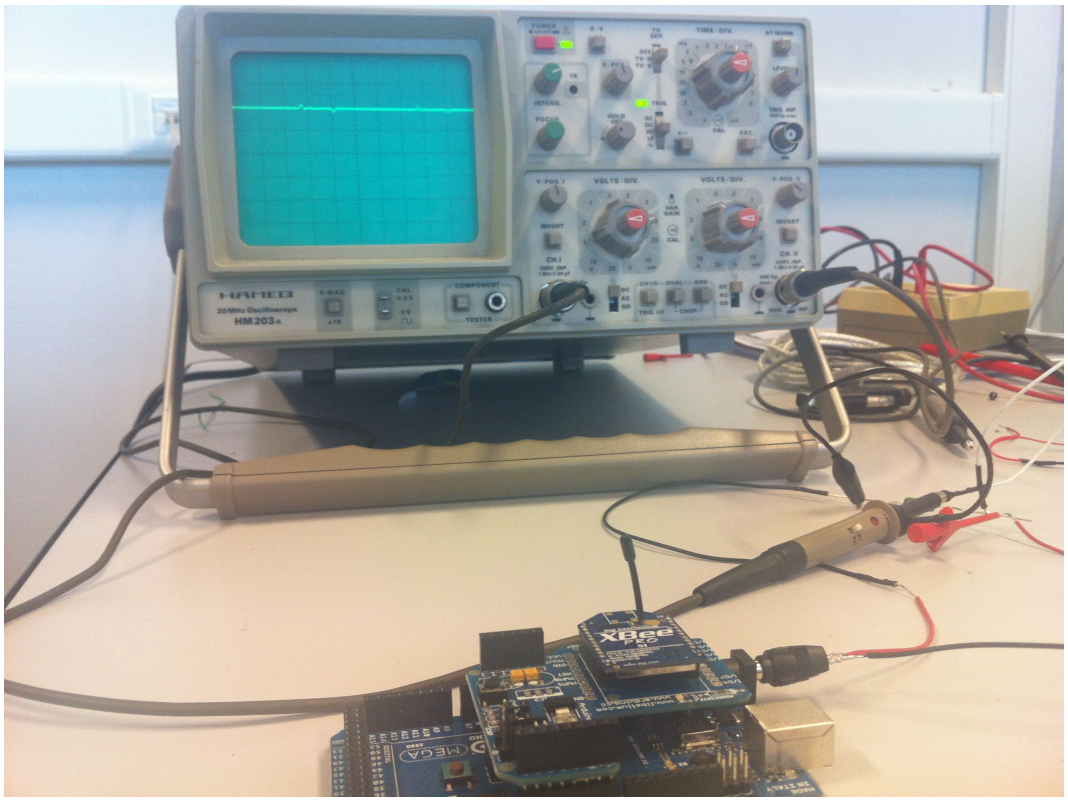


FIGURE 2.7 – Le montage expérimental côté Arduino : on repère le module XBee placé sur une carte en mezzanine et la sonde de l'oscilloscope branchée aux bornes de la résistance shunt de 1 ohm dans la ligne d'alimentation. La courbe de la tension est le reflet de la consommation de courant.

Le logiciel

Un interpréteur Transputer Virtual Machine (TVM) [1] exécute un programme Occam chargé sur la carte. Le programme exécute en boucle une commande d'émission de paquets au module Xbee.

Le code chargé d'écrire le paquet sur l'UART est une primitive C que l'on a développée dans l'interpréteur. Elle assure actuellement seulement l'émission des paquets.

On a fait varier les paramètres suivants :

1. la durée entre deux itérations (11 ms minimum),
2. la taille du paquet émis (100 octets maximum),
3. le débit de données de l'UART (250 000 bauds maxi).

Dans le programme Occam, un chronomètre capture les estampilles temporelles encadrant l'appel de la procédure d'émission (Timer Occam).

Les estampilles récupérées à l'itération i sont copiées dans le paquet suivant qui est transmis à l'itération $i + 1$.

On récupère les paquets radio avec un deuxième module Xbee, installé sur un module USB sur une station Linux. Les estampilles sont stockées dans un fichier, et l'on calcule les valeurs suivantes, pour chaque itération :

1. la durée de l'écriture sur le lien série (transmission radio)
2. l'intervalle entre deux itérations.

Pendant l'exécution du programme, on surveille la courbe de l'intensité affichée à l'oscilloscope.

2.3.2 Les résultats

Deux séries de mesures sont faites :

1. pour une vitesse de l'UART constante (9600 bauds), on fait varier la taille du paquet (21, 32, 64, 96, 100 octets de *payload*),
2. pour une taille de paquet constante, on fait varier la vitesse de l'UART entre l'Arduino et le Xbee.

Il n'y a pas de pause dans le programme entre chaque itération.

Les mesures sont données dans le tableau suivant :

Paquet (octets)	Vitesse (bauds)	Copie UART (ms)	Attentes (ms)	Paquet par (s)	Débit UART (o/s)	Débit réel (o/s)
21	9600	32	11	31,25	656,25	488,37
32	9600	45	11	22,22	711,11	571,43
64	9600	81	11	12,35	790,12	696,65
96	9600	118	11	8,47	813,56	744,19
100	9600	123	11	8,13	813,01	746,27

1. `http://concurrency.cc`

Paquet (octets)	Vitesse (bauds)	Copie UART (ms)	Attentes (ms)	Paquet par (s)	Débit UART (o/s)	Débit réel (o/s)
96	9600	118	11	8,47	813,56	744,19
96	19200	60	11	16,67	1600	1352,11
96	38400	30	11	33,33	3200	2341,46
96	57600	20	11	50,00	4800	3096,77
96	115200	10	11	100,00	9600	4571,43
96	250000	5	11	200,00	19200	6000,00

Consommation

Tension (V)	Sans Xbee	Avec XBee	En émission
7,29		150	340
7,34	80		
6,16	55		
6,09		120	260

Le débit de l'UART est de 9600 bds, chaque paquet fait 32 octets. Les valeurs indiquées sont l'intensité, exprimée en milliampères.

2.3.3 Interprétation

Délai entre chaque transmission

On observe que le délai entre deux émissions consécutives est constant, indépendant de la taille du paquet et de la vitesse de l'UART. Cela correspond à l'exécution du code Occam qui prépare le paquet suivant avec les estampilles.

Durée de copie vers l'UART et débit effectif

En connaissant la durée de la copie d'un paquet, on déduit le nombre de paquets émis par seconde (colonne 5). On calcule également le débit réel de l'UART, qui est inférieur au débit attendu.

Débit net du lien radio

Finalement, la dernière colonne montre le débit net du lien radio. L'ordre de grandeur est de 6 ko/s (48 kbits/s), avec une grande taille de paquet et la vitesse de l'UART la plus élevée. On compare cela au débit du protocole 802.15.4 : 250 kbits/s.

2.3.4 Mesures sur MSP430 et CC2420

Matériel utilisé

On réalise les mesures de consommation d'un micro-contrôleur MSP430 et un émetteur-récepteur CC2420 installé sur une mezzanine. La carte provient de l'ENSSAT de Lannion, elle reproduit une note d'application de Texas Instrument. Un résistance de 1 ohm est branchée en série dans le montage. L'oscilloscope, relié aux bornes de la résistance, affiche la tension : on déduit l'intensité du courant consommé.

Consommation pour un paquet

Le programme réalise une émission régulière d'une trame radio, sous le contrôle d'un minuteur.

Le micro-contrôleur (MCU) est relié au module radio par un bus SPI (Serial peripheral interface, 400 kbits/s). L'émission d'un paquet est déclenchée par une interruption, puis le MCU revient à l'état de veille. Le mode basse consommation choisi est celui où seul le processeur est arrêté. Il existe quatre autres modes basse consommation qui désactivent des horloges secondaires.

La table montre les consommations observées sur la résistance série, et les durées de quelques opérations : veille, transfert de données sur SPI, écoute du canal avant la transmission (phase clear channel assessment CCA), transfert d'un paquet radio, écoute et réception.

	Veille (PM0)	Activité MCU	Écoute (CCA)	Transfert données	Écoute	Réception
Intensité (mA)	2 à 12	16	68	44 à 48	48 à 52	52
Durée (ms)	-	2	< 1	10	-	10

Mesure des débits de données

On réalise en continu l'émission de trames de 32 octets, contenant un numéro d'identification. Chaque commande d'émission démarre dès que le module CC2420 est prêt à émettre, il n'y a pas de pause dans le programme (front montant, en E).

On contrôle la réception avec le module XBee, branché en USB sur la station Linux.

On distingue sur la courbe chaque phase pour émettre un paquet.

A Écriture de la trame sur le bus SPI

B Prise de contrôle du canal radio

C Silence avant l'émission (vérifier le protocole)

D Émission de la trame

E Écriture de la trame suivante

T Période de 20 ms pour émettre 32 octets : débit de 1600 octets/s

2.4 Wireless Shark : analyse des protocoles de communications radio

À côté de l'outillage quantitatif présenté section 2.3, il s'est avéré nécessaire de réaliser un outil qualitatif permettant de déboguer les protocoles radios, et d'obtenir des estampilles de séquencement temporel. *Wireless Shark* [10] a été développé (2012) afin de :

- vérifier le bon fonctionnement des programmes
- aider à comprendre les problèmes de communication et les corriger plus rapidement.

C'est un outillage important pour les réseaux de communication sans fil, car il réalise les opérations suivantes :

- mise au point et vérification des algorithmes : déverminage, analyse des transactions, traces de fonctionnement ;
- inspection des réseaux en opération : détection des intrusions, vérification du bon fonctionnement.

En développant notre propre outillage, nous pouvons contrôler la précision de la capture des données :

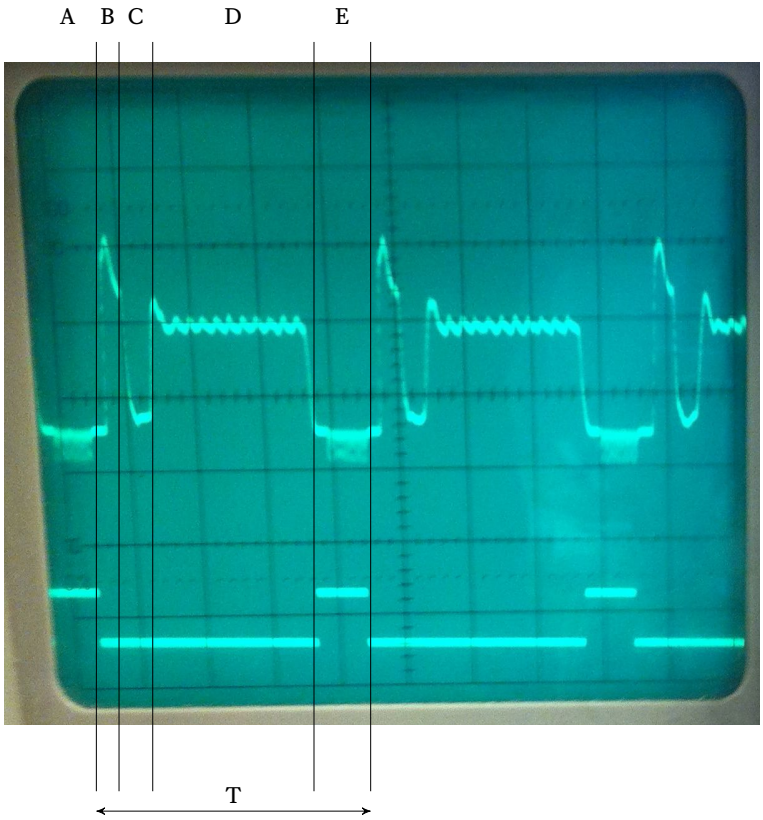


FIGURE 2.8 – Chronogramme d'expédition intensive sur MSP430. On décompose une période en montrant : un pic pour la réservation du lien en CSMA, puis l'écoute des éventuelles réponses, l'émission des données avec un niveau de consommation élevé, et le transfert du paquet suivant du MCU vers le circuit radio. Ce transfert est tracé sur le second chronogramme.

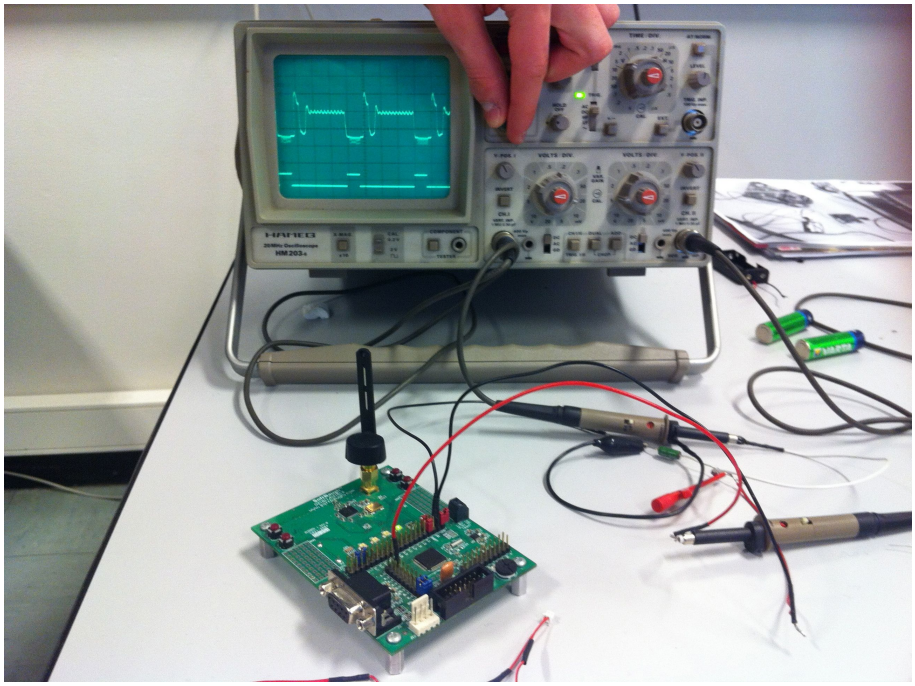


FIGURE 2.9 – Un système MSP430 - CC2420 en observation. La résistance de mesure (vert) se situe sous la sonde du haut.

- détermination des estampilles temporelles de réception,
- récupération possible d'informations provenant du récepteur et associées aux données (puissance et qualité de réception, taux d'erreurs, etc.).

Nous avons développé notre propre système d'analyse des communications radio (figure 2.10), en utilisant :

- une carte Softbaugh (MSP430 et CC2420) qui capture les transmissions radio (les paquets) puis les transmet à un ordinateur,
- sur cet ordinateur, Wireshark affiche ensuite les données capturées, puis les analyse.

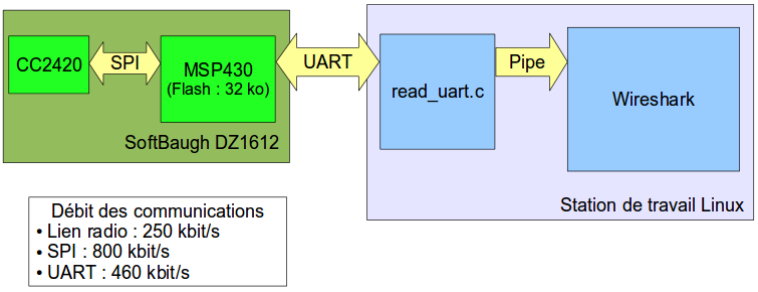


FIGURE 2.10 – Synoptique de l'intégration d'un récepteur radio dans le logiciel Wireshark.

L'analyse peut se faire simultanément, ou hors ligne, après avoir sauvegardé les résultats dans un fichier.

Wireshark est un logiciel libre qui analyse et dépanne les réseaux informatiques. De nombreux protocoles sont pris en charge, notamment le protocole IEEE 802.15.4, le protocole radio utilisé

par les systèmes de capteurs sans fil, par exemple XBee. Ces fonctionnalités sont étendues à des nouveaux protocoles en programmant des modules « disséqueurs » supplémentaires. La figure 2.11 montre ainsi les paquets capturés sur l'interface radio.

Le débit de capture atteint la valeur maximale de la norme IEEE 802.15.4, soit 250 kbit/s. La date de réception d'un paquet est obtenue avec une précision de 10 ms environ. Le système peut donc analyser une séquence de petits paquets émis en rafale, comme un débit soutenu de données.

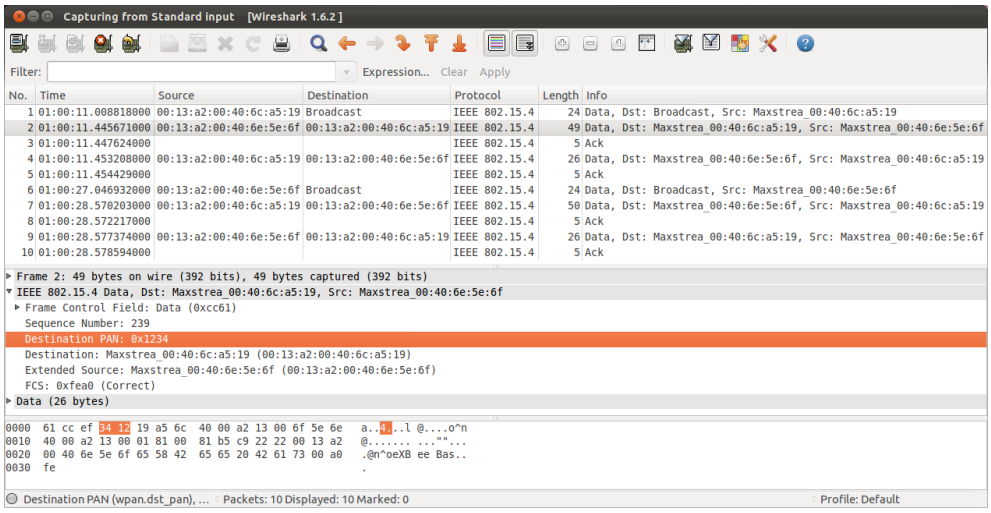


FIGURE 2.11 – Trace de paquets radio 802.15.4 produite par Wireless-shark, utilisant les services du système MSP430+CC2420

2.5 Bilan des expérimentations

Ces réalisations ont permis d'entrer pleinement dans le domaine des réseaux de capteurs sans fil, en suivant une approche réaliste basée sur l'état de l'art technique des premières années des travaux doctoraux. Bien entendu, il s'agit d'un contact, indispensable, mais dépendant de la maturité du domaine de 2010 à 2012.

Les micro-contrôleurs, les protocoles radio, les capteurs, et les applications évoluent à grande vitesse car les enjeux sont de taille, économie de ressources et changement climatique obligent.

Nous allons maintenant nous intéresser à des sujets où les contributions seront plus stables : l'appréhension des dimensions géographiques, où nous contribuons avec Quickmap (chapitre 3), collecte distribuée par un mobile (chapitre 5), et premiers pas vers un environnement cyber-physique, où il sera possible de raisonner sur les contraintes temporelles et spatiales de la collecte (chapitre 6).

3

QuickMap : présentation géographique et capteurs

3.1 Historique et motivations

Les réseaux de capteurs sans fil constituent potentiellement une interface entre l'environnement et les systèmes d'information. Les capteurs sont au contact direct des phénomènes physiques qu'ils perçoivent. Ils mesurent, coopèrent, et assurent un suivi de l'évolution de leur environnement qui est exploité dans le système d'information.

Il est donc primordial d'avoir une connaissance précise de l'environnement que l'on observe. Cette connaissance intègre la distribution spatiale, mais aussi la qualification du phénomène observé. Il peut s'agir d'une ressource, telle qu'une place de stationnement, une hauteur d'eau, une pollution, le bruit, l'élévation, etc.

Cette thèse a côtoyé NetGen, le logiciel phare du groupe WSN qui intègre à la fois un modèle simulable pour les réseaux de capteurs apparu en 2007 ([11]) et une composante géographique. Celle-ci est apparue en 2010 avec un interface de saisie permettant de localiser des points sur un fond de carte. Les points avaient alors une simple sémantique géométrique dans le plan (x,y) de l'interface de saisie.

Il est apparu que cette notion était simpliste, en particulier parce que les données géographiques sont référencées sur la sphère terrestre, en terme de longitude et de latitude. Il s'en est suivi un historique de travaux dont la finalité a été l'aisance de la manipulation géographique *Anywhere, any scale*, la mobilité et la possibilité de lire et écrire des données *géo-localisées*.

Les figures 3.1 et 3.2 montrent des évolutions de l'articulation entre les représentations graphiques géométriques, puis géo-localisées, et les réseaux de capteurs, lors de la planification de déploiement, puis sur un site de déploiement. Aujourd'hui ces outils interagissent autour d'une base de données Postgis (figure 3.12). L'annexe B fournit un historique des évolutions de la plateforme que l'on nomme *NetGen* avec nos contributions.

FIGURE 3.1 – Saisie sur fond de carte, positionnement géométrique, chemins de mobiles et calcul des rencontres.

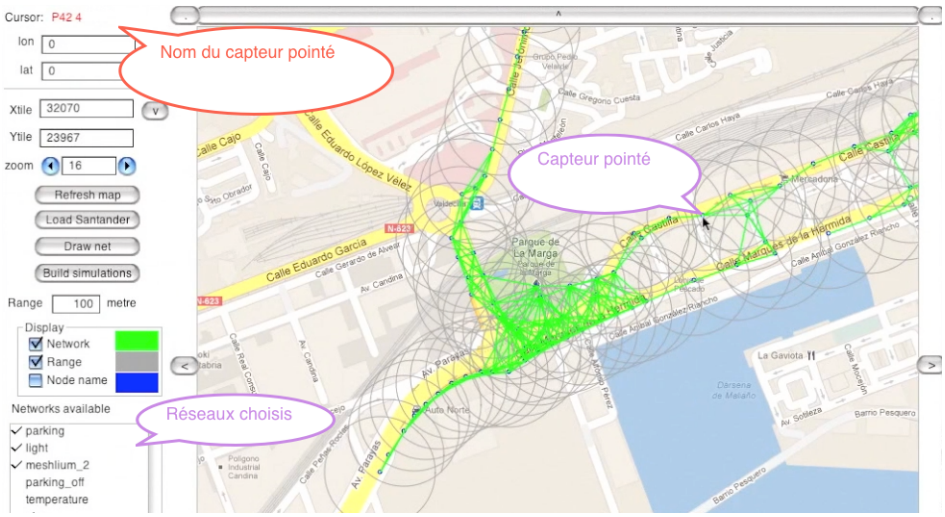


FIGURE 3.2 – Saisie GMap, positionnement géographique, lien sémantique en temps réel avec certains des capteurs de la *Smart city* de Santander. Construction d'un réseau simulable et activation possible de ce réseau. La marge gauche montre les réseaux collectés sur le site et les couleurs attribuées pour leur présentation.

3.2 Applications « rasterisées » et géo-localisation

La modélisation et l'observation de l'environnement amènent une très grande quantité d'informations à recueillir, manipuler et stocker.

Les informations géographiques sont caractérisées par deux grandeurs : l'espace et le temps. L'espace, c'est la position de la donnée, et le temps, c'est la date ou la durée pendant laquelle la donnée est valable.

Les objets géographiques sont représentés sur la base de formes telles que :

- des points,
- des segments (les routes, les cours d'eau…),
- des polygones (bâtiments, zones remarquables).

3.2.1 Du système d'information géographique aux cartes

D'où viennent les informations géographiques ? Ce sont des mesures sur le terrain, des observations, des résultats d'analyse d'images et des relevés topographiques qui permettent d'énoncer une composition géographique, puis de créer une carte.

Les informations géographiques élémentaires sont stockées dans des bases de données sous forme symbolique ou numérique. Ces informations sont finalisées dynamiquement pour produire des rendus spécifiques. On peut ici mentionner les grands fournisseurs de cartes en ligne : Google Maps, Bing Maps, OpenStreetMap, MapQuest, ESRI et MapBox.

Dans le cas d'OpenStreetMap (OSM), les services de présentation sont rendus comme le montre la figure 3.3.

- La base de données initiale est un Postgres spécialisé avec des extensions GIS (Postgis). Sur ces extensions, on charge des fichiers de cartes pour des villes, des régions, voire la planète entière. OSM propose ses données en usage libre.
- La seconde étape de la *rasterisation* consiste à sélectionner des éléments dans une zone géographique et à composer une image pour cette zone. Ce travail est fait par un processus spécialisé conformément à un style. Le procédé est générique, il est donc possible de produire des rendus différents à partir d'une certaine base. Les images sont découpées en *tuiles* de taille fixe (typiquement un carré de 256 pixels de côté) que l'on peut télécharger individuellement.
- La troisième étape consiste à loger la tuile dans un cache, en prévision d'une réutilisation possible. Un serveur Web peut être configuré pour servir ces tuiles en utilisant ce cache. La table 3.1 présente la structure d'un répertoire cache pour une variété de cartes.

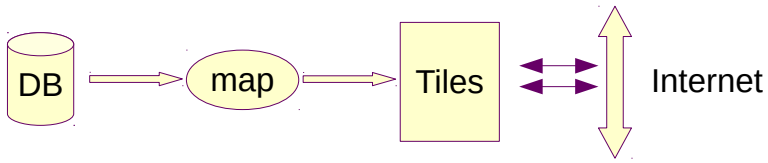


FIGURE 3.3 – Organisation d'un service de production de cartes sur *Linux* : base de données Postgres (DB), rendu OSM par un service *renderd* (map), cache (tiles), serveur Apache2 interprétant les requêtes en s'appuyant sur le cache et le processus de rendu.

taille Ko	zone
9156	centralamerica
16420	default
16468	france
16336	indonesia
4904	madagascar
17116	vietnam

TABLE 3.1 – Capacités pour quelques caches de cartes pour OSM. Ces répertoires sont structurés en niveau de zoom, de 0 à 19. Les niveaux faibles peuvent être cachés, les niveaux hauts sont générés dynamiquement, compte tenu de leur nombre et de la taille disque nécessaire. Les serveurs de tuiles *industriels* doivent être puissants, parallèles et bien dotés en disque.

3.2.2 Accès aux tuiles

Il existe plusieurs techniques permettant d'adresser les tuiles sur un serveur. La syntaxe utilisée pour OSM est TMS (Tile Map Service), qui conduit à décrire les tuiles sous la forme $URL/zoom/x/y.png$, par exemple `http://sames.univ-brest.fr/osm_tiles/2/1/1.png` rend l'atlantique nord.

zoom : résolution, de 0 pour la terre complète à 19 pour les tuiles les plus précises,

x : index horizontal de la colonne de la tuile,

y : index vertical de la tuile qui est un fichier PNG.

Cette syntaxe permet de retrouver ou générer une image correspondant à la zone recherchée. Elle permet aussi à des brosseurs externes de naviguer sur le serveur, en profondeur en jouant sur le zoom, et à un niveau de zoom fixe, à l'horizontale, en calculant les valeurs de x et y. Voir figure 3.5. Ces brosseurs externes font partie de l'usage courant et public des cartes. Il peut s'agir de modules d'extension disposés dans des pages servies (figure 3.4), ou de programmes spécifiques tels que QGis ou QuickMap.

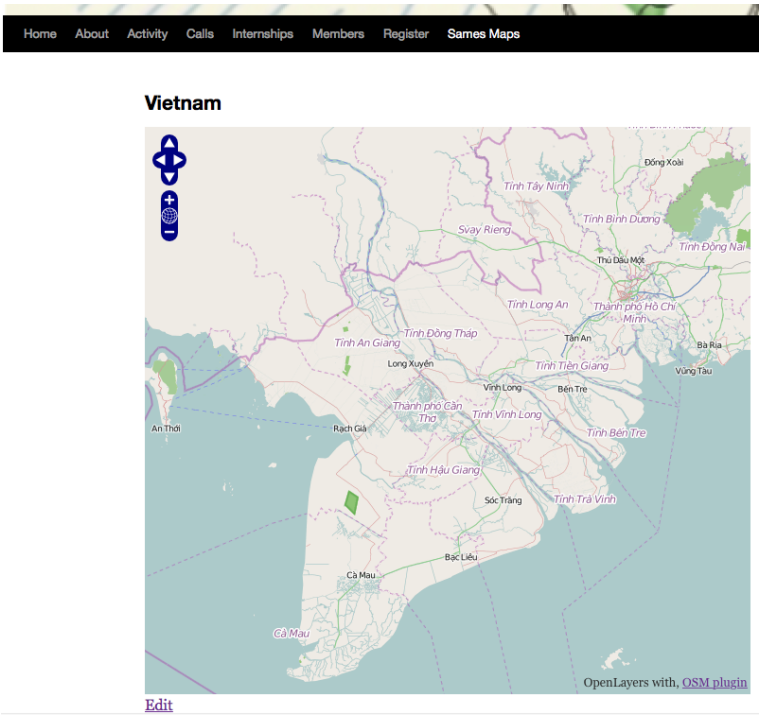


FIGURE 3.4 – Brosseur de cartes OSM, extension disponible dans WordPress. Les icônes de navigation permettent des déplacements à facteur de zoom constant, ou en profondeur. Ces fonctions sont disponibles de manière commode par manipulation de la souris.

Les cartes au format raster sont donc consultables en ligne, à travers un format qui en permet un affichage pratique. Les tuiles sont assemblées dynamiquement pour reconstituer la carte. Plusieurs échelles sont disponibles : la tuile d'une échelle donnée est découpée en quatre nouvelles tuiles à l'échelle suivante. Cela forme une pyramide avec au sommet, une seule tuile à l'échelle la plus grande et, à la base, les tuiles à l'échelle la plus petite. Les dimensions de la carte sont doublées lors du passage à l'échelle supérieure, selon la formule suivante : $dimension_n = dimension_0 \times 2^{n-1}$, où $dimension_n$ est la dimension totale de la carte à l'échelle n°n.

FIGURE 3.5 – Tuile rendue par le serveur du laboratoire pour l'URL `http://sames.univ-brest.fr/osm_tiles/5/15/10.png`. Le niveau de zoom est 5, la tuile est produite à partir de la base OSM greatbritain.

Ce mode de représentation utilise la projection dite de Mercator, avec une troncature aux pôles pour produire une carte carrée. À chaque point, ou pixel, correspond une position géographique (figure 3.6).

Les coordonnées géographiques longitude et latitude sont des valeurs angulaires en degrés. Les coordonnées métriques sur la projection Mercator sont calculées par la formule suivante :

$$x = \lambda - \lambda_0$$

$$y = \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right]$$

En pratique, la projection de Mercator est représentée par des fonctions de traduction. Dans l'atelier NetGen, on les trouve en syntaxe Smalltalk dans la classe GlobalMercator [12]), dans l'environnement de développement des outils, en syntaxe Occam, et en syntaxe CUDA pour un usage dynamique dans les algorithmes. Ces fonctions sont reproduites de codes sources largement documentés dans le domaine public.

Les positions géographiques sont présentées sous forme de couples longitude (x), latitude (y) en précision numérique élevée. Les projections sur les cartes à deux dimensions sont données par des points x, y. Les distances et les positions relatives sur une carte sont obtenues par les fonctions de la librairie Mercator.

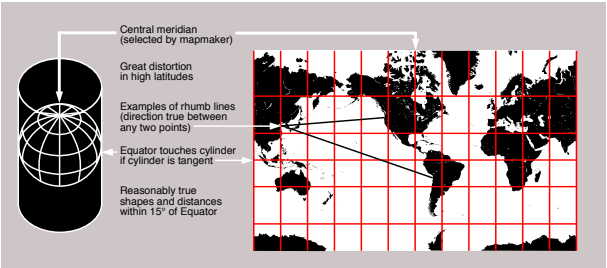


FIGURE 3.6 – Projection de Mercator (source [13])

3.3 Applications en environnement objet

On a vu que les constructions algorithmiques opérant sur la réalité physique ont besoin de métriques, un élément évident étant l'appréciation de la position. Nous avons vu que la sphère terrestre peut être modélisée à partir de coordonnées à deux dimensions. La troisième dimension, z représente l'élévation de points, positivement ou négativement par rapport à la sphère. En plus de cette dimension, le temps est un facteur qui permet de décrire des processus dynamiques, qu'il s'agisse de mobiles discrets, ou d'évolution physiques telles que les courants ou les phénomènes météorologiques. Ces questions permettent d'opérer sur des modélisations d'interactions cyber-physiques telles que Edward Lee les présente [14].

Les informations sur les réalités de terrain sont disponibles sur les bases de données géographiques, ou des systèmes complémentaires.

3.3.1 Objets remarquables et GIS

Les bases de données géographiques sont une source d'informations critiques. Elles décrivent les routes, l'organisation des bâtiments, les rivières, ainsi que des « points d'intérêt ». On a vu que l'usage de OpenStreetMap compte un nombre d'utilisateurs enregistrés qui croît de manière exponentielle (figure 1.4). Actuellement la base planétaire de OpenStreetMap tient dans un fichier comprimé de 50 giga-octets. La base concernant la France occupe plus de 3 giga-octets.

À côté des présentations « *raster* », il existe des logiciels libres de manipulation de données géographiques, telles que Quantum Gis (QGis). QGis permet de sélectionner, charger et présenter des données provenant d'OpenStreetMap, par exemple.

Les données géographiques sont organisées par tables primaires ou secondaires :

planet_osm_line : des lignes,

planet_osm_point : des nœuds,

planet_osm_polygon : des contours de zones,

planet_osm_roads : des définitions de voies logiques.

Des données graphiques paramètrent l'affichage des objets : couleurs, traits, et des méta-données permettent d'enrichir par des données associées : ce sont par exemple la date d'acquisition et le propriétaire.

Les données sont affichées par couche. On peut créer une nouvelle couche à partir de certains objets sélectionnés, ou en créer de nouveaux. Ces objets sont exportables dans un format de fichier qui peut être relu par les autres logiciels de SIG (figure 3.7).

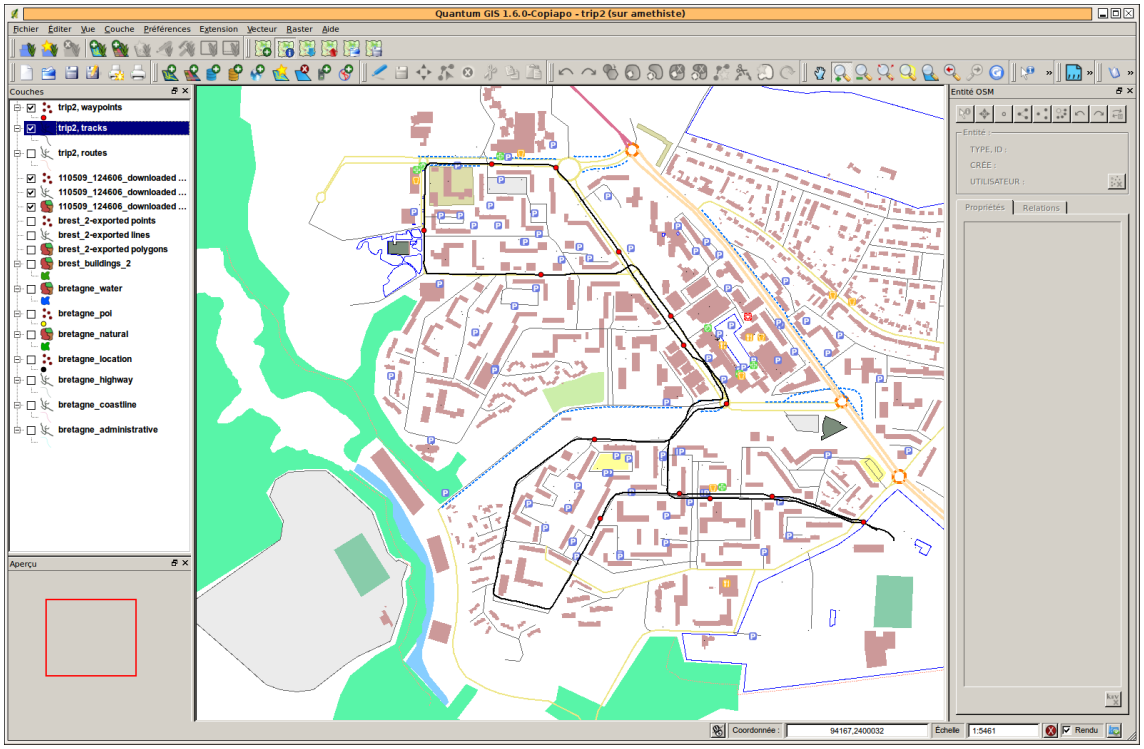


FIGURE 3.7 – Fenêtre de QGis présentant le quartier de Bellevue à Brest, d'après les informations recueillies sur OpenStreetMap standard. On y reconnaît aussi un chemin suivi par une voiture, saisi à partir de l'outil MSP+GPS décrit en figure 2.4, avec une disposition de stations de bus présentés en rouge. Ce trajet est estampillé temporellement.

3.3.2 Applications du modèle géographique dans NetGen

Le modèle amont à la simulation se comprend très bien à partir d'explications menées sur une image d'IHM (interface homme-machine). Dans la fenêtre 3.8, on repère la présentation graphique d'une image satellite, les contours de bâtiments, un parcours de mobile, les capteurs et leur portée, le réseau, les rencontres entre le mobile et le réseau.

L'interface de saisie sur carte possède deux référencements géographiques. Les capteurs peuvent être positionnés pour des considérations telles que la proximité d'un arrêt de bus ou d'un éclairage. La portée représente la puissance d'émission et influence la connectivité dans le modèle. Nous avons établi plusieurs interfaces permettant l'intégration de données publiques (SIG), ou la production de représentations familières (QGIS, SHP pour les données de Brest, GPX, KML pour les parcours). Ce réseau est simulable, en fonction de la portée des émetteurs, y compris les mouvements de mobiles sur leurs chemins. *L'étude des interactions avec les données publiques, la géo-localisation des objets a été menée au premier trimestre 2011.*

3.3.3 Affinement du calcul de couverture radio par lancer de rayon

Grâce aux SIG, nous pouvons obtenir l'emplacement des constructions pour simuler la couverture d'un émetteur radio. Les émissions radio sont fortement atténuées par les bâtiments, qui font obstacle et qui réfléchissent le signal.

Ceci provoque des zones d'ombre où aucun signal n'est reçu, et des interférences qui perturbent la réception. Ces informations de terrain permettent de calculer plus précisément la couverture des

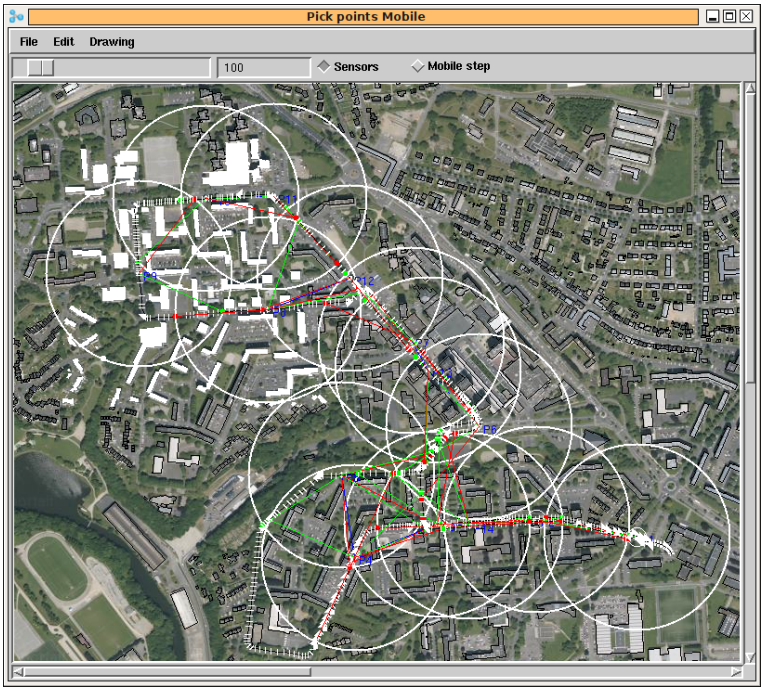


FIGURE 3.8 – Résultat de la simulation dans l'interface de NetGen. Les capteurs saisis sont placés sur une image satellite du quartier de Bellevue, à Brest. Leur portée attendue (théorique) est délimitée par un cercle. L'emplacement des bâtiments a été ajouté d'après les informations provenant de fichiers de type *shapefile* rendus publics par Brest Métropole Océane (BMO) [15]. Des animations montrant l'éclairage et les zones atteignables sur le réseau ont été réalisées.

émetteurs, pour obtenir une simulation réaliste.

Il est concrètement nécessaire d'obtenir des informations 3D venant du terrain afin d'avoir un modèle plus proche de la réalité. Cela conduit à une estimation plus précise de la couverture des émetteurs, pour obtenir une simulation réaliste. Ces travaux sont en cours en ce qui concerne les reliefs et leur appréciation provenant de mesures par satellites [16, 17].

Les étudiants de Master ont implémenté un calcul de couverture radio en utilisant la technique de lancer de rayon (*ray tracing*). Ils ont importé une carte des bâtiments de la ville de Brest dans le simulateur, puis ils ont représenté la surface où un récepteur est situé dans la ligne de vue de l'émetteur (voir fenêtre 3.9). La carte des bâtiments est produite à partir des données venant d'OpenStreetMap *(c'est une de mes contributions)*.

Les surfaces calculées peuvent être exportées dans un fichier *shapefile*, puis remises à disposition du public si elles présentent un intérêt (éclairage public, réseau radio, etc.).

3.4 Organisation du brosseur QuickMap

Nous avons développé ce brosseur avec les objectifs suivants :

- permettre une promenade planétaire en x,y,z ;
- interfacer des serveurs de tuiles variés : cartes (OSM…) ou images de satellites (ARCGis…), présenter les images sur un interface local ;
- permettre des transformations entre points, nuages de points, coupes, distances géométriques et valeurs vraies sur la sphère terrestre ;

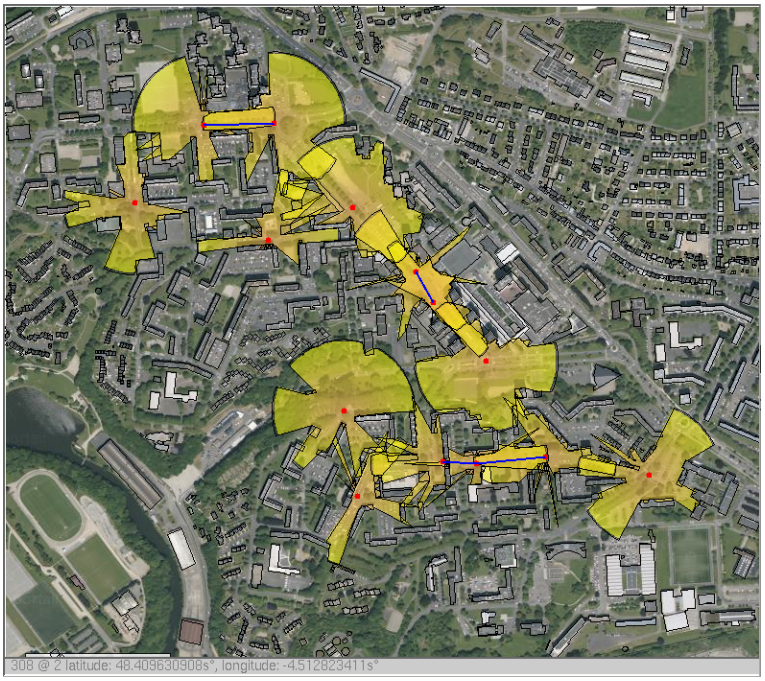


FIGURE 3.9 – Résultat du calcul de lancer de rayon (Florent Chateau, Yohann Le Gall et Romain Herry [18])

- permettre la consultation et la création de données relatives aux réseaux de capteurs ;
- permettre la spécification de chemins de mobiles.

QuickMap est la première étape du flot de conception NetGen, qui peut conduire à d'autres fonctions, notamment PickCell pour la synthèse de systèmes cellulaires (figure 4.4), et l'interface de GPredict (figure 5.1) qui permet de connecter les nuages de capteurs à des visites par satellites [7].

3.4.1 Système MVC interne

QuickMap repose sur un système Modèle-Vue-Contrôleur (MVC) implémenté en environnement Visualworks. Les quatre classes principales sont :

l'application : **UIQuickMap**, qui hérite de la classe support ApplicationModel. Plusieurs interfaces sont proposées afin de recouper des besoins expérimentaux. Le démarrage de l'application est assuré à partir d'un menu principal *Tools*.

le modèle : une instance de **QuickTileMap** qui représente les données à afficher et d'autres caractéristiques. L'essentiel du modèle est accessible dans une variable de la classe **QuickTileModel**. Le modèle gère la position dans le tuilage, et des outillages permettant de saisir des nuages de points, ou des chemins de mobiles. Cette classe définit les fonctions d'affichages dans ses méthodes displayOn :. L'arbre d'héritage inclut QuickTileComposite et QuickTile qui est un composant visuel.

la vue : un afficheur qui est une instance de la classe **ScrollWrapper** encapsulant le modèle. Cette classe permet de gérer automatiquement et efficacement une vue glissante dans une fenêtre de taille donnée.

le contrôleur : une instance de **QuickTileController**, sous-classe de Controller, qui gère les évènements utilisateurs. Ce contrôleur interprète l'état des touches *méta* du clavier pour

décider si il est nécessaire d'accomplir un mouvement plan, ou au contraire saisir un point, ou une forme géométrique.

L'application reçoit d'autres composants classiques, menus, boutons permettant de spécialiser l'interface. Les figures 3.10 et 3.11 présentent deux vues de QuickMap configurées pour accéder aux images satellites ArcGis et à un serveur interne proposant la carte OSM actuelle pour la France.

On note qu'un des boutons à gauche présente le choix de plusieurs serveurs, ce qui permet de commuter les présentations très rapidement. La présence d'un serveur GIS local est un autre atout, si on considère la possibilité d'en extraire des informations calculées par simulation, et d'y déposer des objets graphiques saisis.

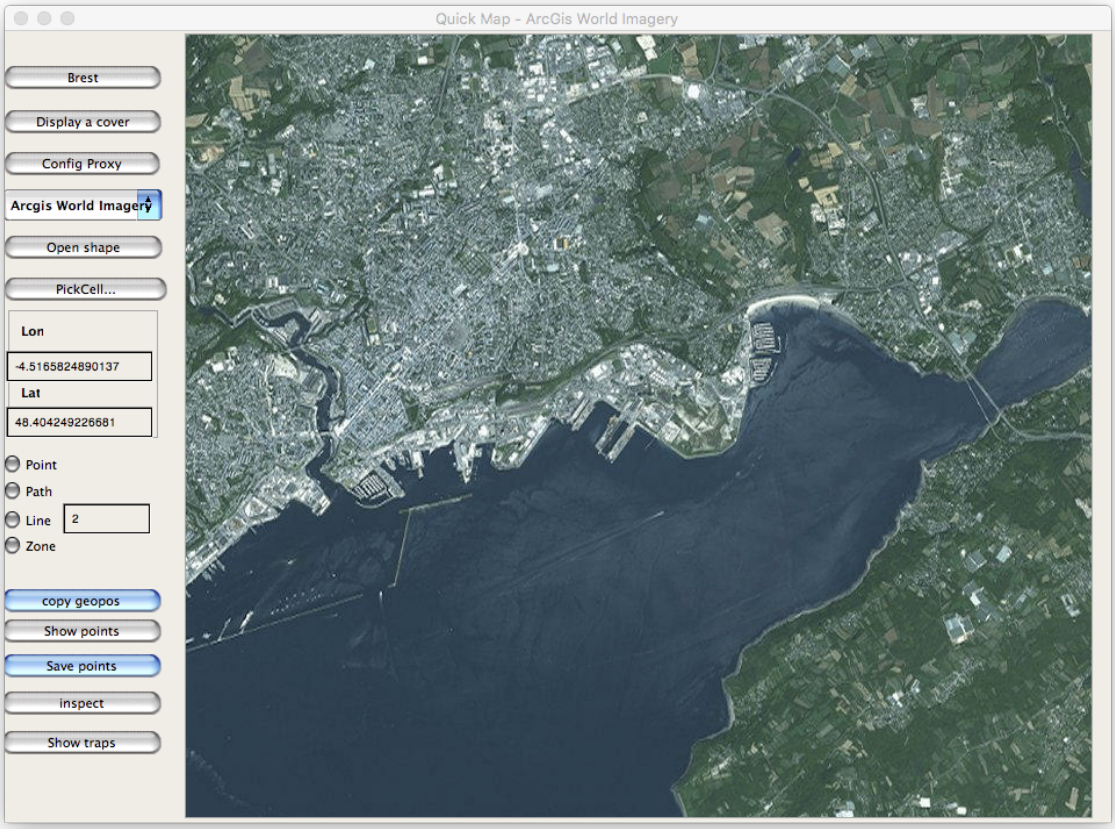


FIGURE 3.10 – Photo satellite ArcGis, Rade de Brest. Il s'agit de la position préférentielle proposée par le premier bouton. On note en dessous les fonctions de projection de zones de couverture, en général résultats de simulation, puis la fonction de configuration de proxy pour les accès à l'Internet. Deux autres boutons permettent de charger des fichiers *shapefile* tels que les contours de bâtiments, et de transférer l'image sélectionnée dans l'outil PickCell.

3.4.2 Paramétrage d'accès aux serveurs de tuiles : QuickTileProxy

Le support d'accès à ces serveurs est un ensemble de classes qui se chargent d'adapter une adresse en forme canonique OSM pour les caractéristiques d'un serveur particulier. Il s'agit de l'adresse URL du serveur, et des paramètres de désignation d'une tuile ou d'un service particulier.

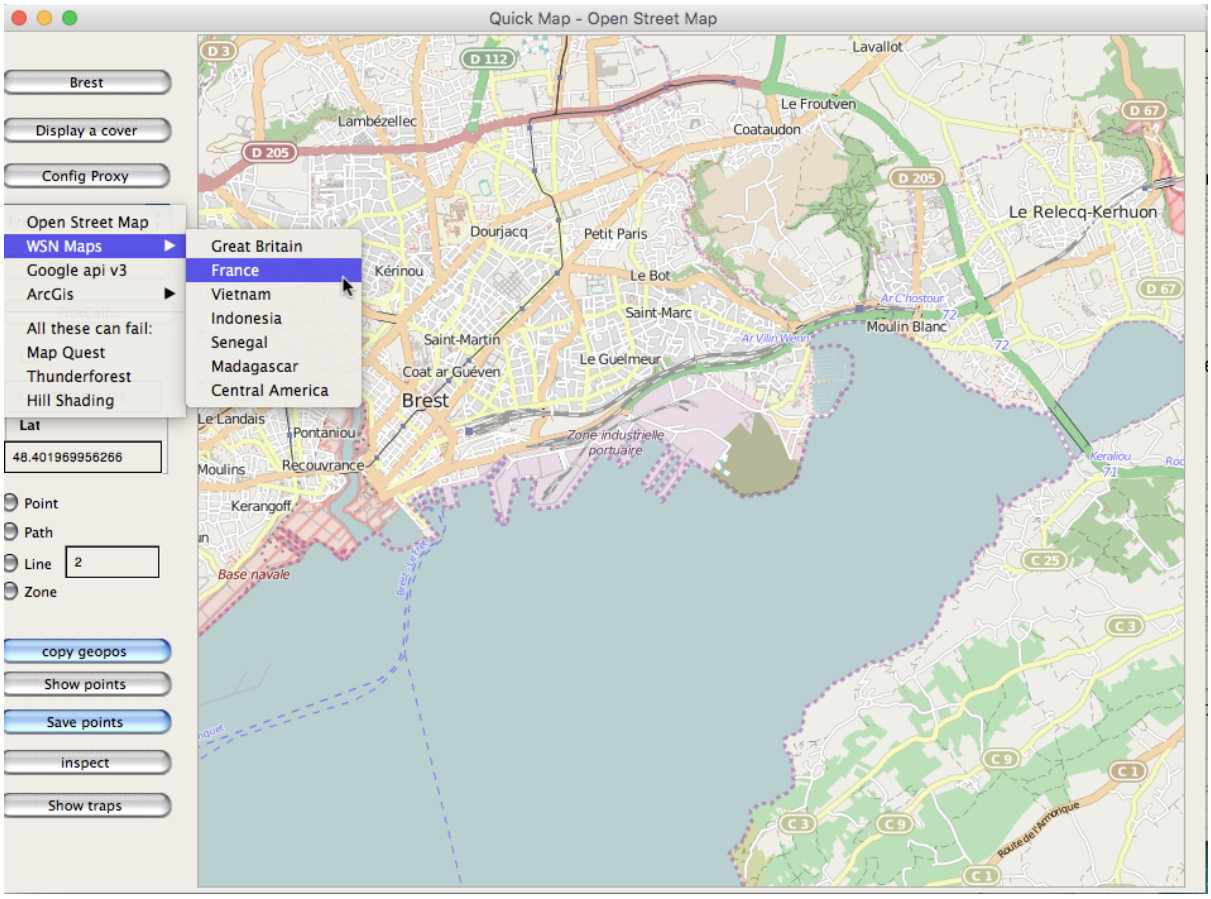


FIGURE 3.11 – Image extraite d'une des bases locales en rendu OSM. Il faut noter le menu hiérarchique de sélection des serveurs de tuiles et de cartes en GIS. WSN est un serveur de cartes de l'équipe accessible sur l'Internet, et qui présente les mêmes cartes à l'adresse `http://sames.univ-brest.fr/sameswp/sames-maps/brittany/`. On peut aussi noter les coordonnées géographiques du curseur en milieu de page, et les fonctions de saisie de géométries spécifiques aux capteurs.

La forme des fichiers peut être un élément de variation, les images présentées dans QuickMap sont des fichiers qui nécessitent parfois une adaptation de format. Ces classes peuvent donc avoir à télécharger la tuile, la convertir, pour ensuite la présenter en environnement Visualworks.

Une dizaine de classes héritant de QuickTileProxy remplissent ces fonctions. Le système peut facilement être étendu ou adapté. Un point d'entrée unique `getUri: anInteger x: x y: y zoom: zoom` permet d'accéder aux serveurs distants et de masquer les caractéristiques de ces services.

3.4.3 Fonctionnement dynamique

La classe **QuickTile** représente une tuile intégrée dans l'image présentée. Cette tuile est localisée en un point de l'afficheur. Pour gérer les tuiles, la classe **QuickTileProxy** propose un cache dans lequel il stocke les images provenant du serveur. On procède en émettant une requête HTTP. À la fin de cette requête, l'image est lue et transformée en objet **QuickTile** pour servir à la composition du modèle.

Les requêtes sont asynchrones, les résultats étant protégés par des sémaphores.

Le système MVC présenté peut être piloté par des mouvements extérieurs, un mobile se déplaçant peut asservir l'afficheur à sa position.

3.5 Bilan de ces travaux

Les aspects géographiques sont prépondérants dans la conception des systèmes de perception et de contrôle pour la surveillance de l'environnement.

Ce chapitre a donné un aperçu des outils permettant d'appréhender les informations connues et mises à disposition dans les systèmes d'information géographiques. Nous avons montré que ces informations sont disponibles directement sur les bases GIS, ou indirectement via des représentations graphiques. Ces deux types d'accès ont été pratiqués pour définir les placements de capteurs en regard des objets et des obstacles physiques (manipulation et intégration dans QGis), dispositions géographiques des capteurs et synthèse des réseaux logiques obtenus dans le flot NetGen. Le chapitre 4 va détailler comment la transition entre les informations géographiques et les descriptions de systèmes s'effectuent. Le chapitre 5 montrera ensuite comment des objets dynamiques peuvent interagir avec des objets statiques dans ce cadre. Le modèle choisi est celui de satellites basses orbites, mais d'autres systèmes mobiles pourraient parfaitement s'appliquer (lignes régulières ou visites programmées).

Au niveau des réalisations, *nous pouvons revendiquer la conception et la réalisation de QuickMap*, brosseur de cartes, mais aussi *couteau suisse* permettant d'intégrer des objets relatifs aux systèmes de capteurs, et notamment de passer à des descriptions cellulaires de grande taille sur lesquelles les simulations physiques peuvent opérer. QuickMap alimente Pickcell simplement et commodément, en permettant aux réalisateurs de positionner leurs systèmes puis de passer à une analyse cellulaire des comportements physiques.

Cette intégration est réalisée en environnement objet, mais les générateurs construits en amont permettent de produire des simulateurs performants au niveau physique et au niveau des réseaux de capteurs qui interfacent ce niveau. Une organisation du système d'information actuel est présentée en figure 3.12. On y voit la base de données centrale, la branche simulation, la branche d'injection des données capteurs, et celle qui concerne l'élaboration des connaissances. Toutes les interactions dans ce système sont géo-localisées et grandement facilitées par le brosseur de carte intégré.

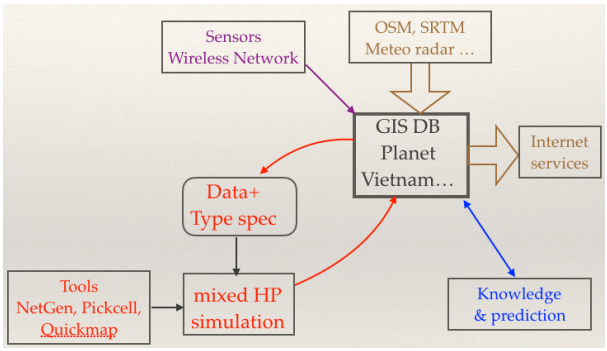


FIGURE 3.12 – Organisation d'un système d'information pour les réseaux de capteurs géo-localisés. La base de données est lue et écrite par les outils spécialisés internes. La base est alimentée et mise à jour par des données externes, et rend des services publics via des services cartographiques.

4

Un flot de conception orienté processus

4.1 Structure et objectifs

Ces travaux de thèse se sont appuyés sur les outils NetGen développés à l'Université de Brest. L'objectif de ces outils est d'apporter des solutions pour la simulation et ensuite, pour le déploiement de réseaux de capteurs distribués.

4.1.1 Simulation du réseau et simulation physique

Actuellement deux objectifs principaux sont visés. Le premier est l'exécution distribuée et sa simulation, et dans une moindre mesure l'exécution locale, au sens de l'acquisition et des pré-traitements. Le second est la simulation cellulaire physique pressentie comme indispensable à la validation de l'instrument de mesure distribué qu'est le réseau de capteurs.

Le premier objectif est couvert par des outils de saisie tels que ceux décrits au chapitre 3, le noyau de NetGen, et des générateurs de code pour les processus communicants et pour les accélérateurs graphiques GPUs. Le second objectif n'est pas discuté dans cette thèse, et concerne la représentation cellulaire physique, que l'on confronte au réseau d'observation. Cet objectif est couvert par les outils *PickCell* qui utilisent les mêmes modèles et générateurs que la spécification de réseau. Il concerne les vrais problèmes traités par les réseaux de capteurs : évolutions de l'environnement, ou artefacts humains. L'intégration des flots est présenté dans la figure 4.2, l'interface de Pickcell est présentée dans les figures 4.4 et 4.5.

Un troisième objectif sera considéré avec la gestion et l'analyse de données collectées. Les données peuvent être géo-localisées et distribuées. Il s'agit ici de travaux de modélisation de données, en élaboration des connaissances que l'on peut rapprocher de considérations centrales dans le développement des systèmes d'information et de gestion de la connaissance. Cette dimension existe déjà dans les outils actuels, avec les modèles cellulaires embarquant des données variées, et avec les modèles de données circulant dans les réseaux de capteurs. Il n'y a pas eu d'initiative concernant la modélisation des données échangées lors de nos travaux. Voir cependant l'ébauche pratique réalisée pour la présentation des données de Santander, déjà mentionnée (figure 3.2), voir aussi l'accès aux

élévations SRTM, et la production de zones de visibilité lors de la simulation LineOfSight de [17, 16]. Le flot général interagit avec des bases d'informations géo-localisées à plusieurs étapes (figure 4.3).

En première approche, la figure 4.1 présente un flot simplifié conduisant à la conception de réseaux de capteurs géo-localisés. On reconnaît en haut à droite l'interaction géographique décrite au chapitre 3. Le *modèle de déploiement* montré sur la figure conduit à des annotations sur la partie présentation sous forme d'annotations graphiques.

Cette organisation permet ainsi de contrôler la répartition des capteurs sur le terrain en réalisant un échantillonnage spatial des processus observés. La fréquence des mesures réalise un échantillonnage temporel de ces processus. Le couplage entre l'observation montrée ici, et les processus physiques doit être validé, et la simulation conjointe est une solution pour obtenir cette validation [19].

L'estimation de la portée des transmissions radio nécessaires pour collecter les données et les acheminer jusqu'à une passerelle vers l'Internet est ici grossière et déduite des caractéristiques des adaptateurs radio. On sait que des calculs plus précis de la propagation peuvent être menés par simulation cellulaire [17, 16], en tenant compte des obstacles et en utilisant la puissance de calcul des GPUs.

4.1.2 Caractéristiques méthodologiques

NetGen propose un flot de conception *descendant*, inspiré de la conception matérielle, telle qu'elle est pratiquée dans les langages de description matérielle (HDL). Il s'agit d'un effort que l'on peut situer dans un cadre de *Conception Assistée par Ordinateur* (CAD), qui place au premier plan la finalité de l'application, et qui permet de tester et utiliser plusieurs variantes de mises en œuvre, en particulier pour les réseaux sans fil.

Dans le domaine des HDL, il a été prouvé que cette approche est la seule qui permette d'augmenter les ambitions sans risque. Elle permet en effet d'isoler des paramètres de la solution les uns des autres, par exemple le choix du réseau peut varier en gardant la couverture physique fixe, et les capteurs fixes. La fonction de coût à optimiser peut inclure le coût économique, mais aussi des caractéristiques liés aux risques. Une référence méthodologique est à l'évidence celle proposée par D.D.Gajski pour la conception et la synthèse des circuits intégrés en se basant sur les systèmes plutôt que les aspects locaux [20] [1].

Un autre avantage majeur de cette approche est la possibilité de garder les outils de prototypage lors de l'exploitation du réseau. Les aspects graphiques, les données intermédiaires, les simulateurs restent disponibles si un accident ou une modification physique sont constatés. La vitesse de prise en charge des applications est ici primordiale et acquise grâce à la méthodologie *descendante* utilisée. Une preuve en est donnée par les interfaces et les simulations réalisées à distance sur le réseau de Santander (figure 3.2).

4.1.3 Étapes hautes du flot

Le flot est organisé ainsi :

1. **La description physique du réseau.** C'est une spécification du déploiement des capteurs, qui inclut leur positionnement et peut-être des annotations fonctionnelles permettant de les caractériser (cas de Santander).

1. *"C++ as presently practiced in EDA industry is creating more chaos than progress and if the path is followed it will slow down progress in EDA for 10 years to come."* [21]

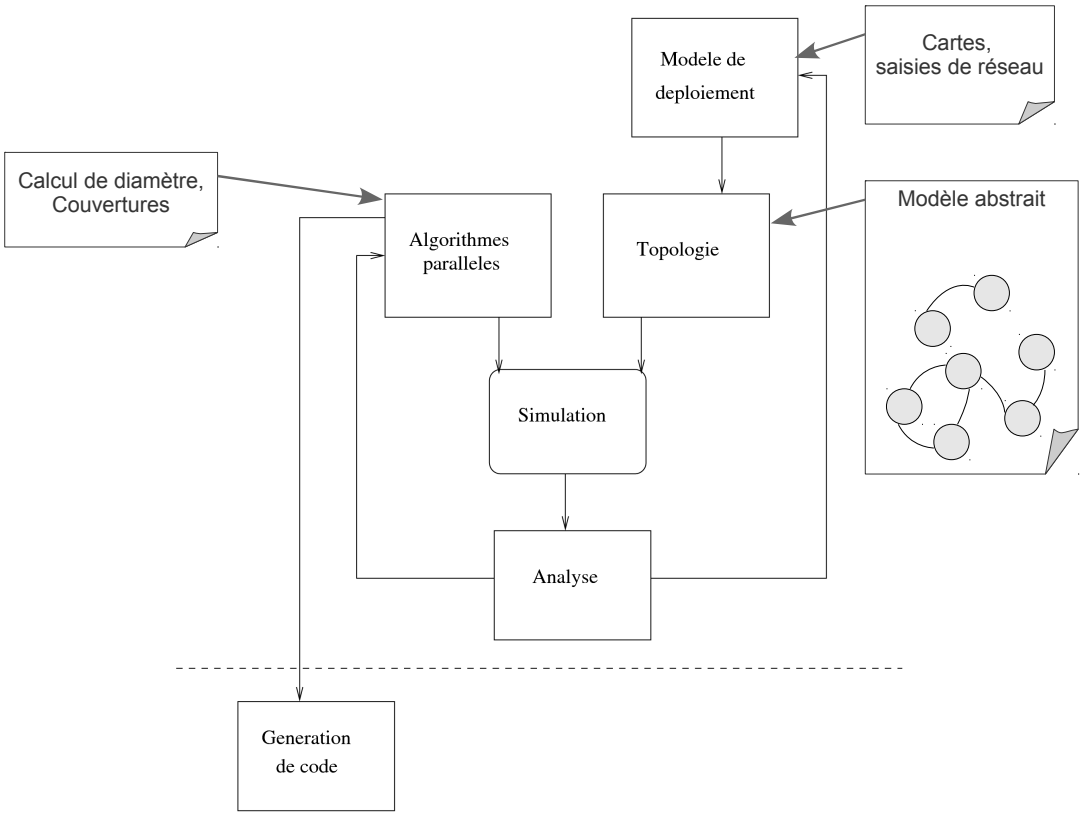


FIGURE 4.1 – Le flot de conception NetGen simplifié pour les réseaux de capteurs : présentations graphiques et géographiques en haut à droite, modèle structurel généré (Topologie) et comportements spécifiés (Algorithmes) au centre, production de code embarqué, en bas.

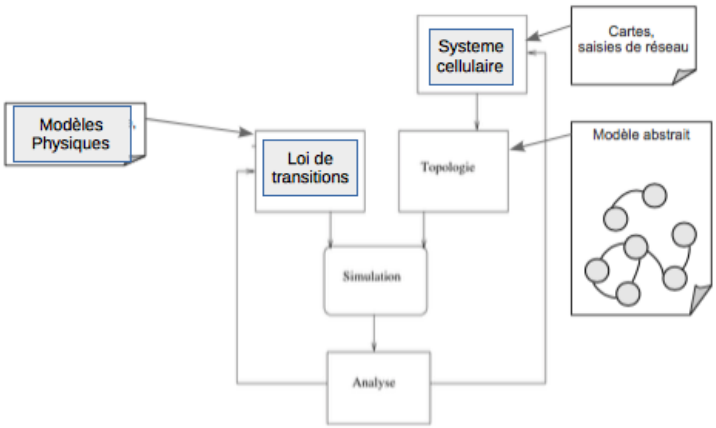


FIGURE 4.2 – Le flot de conception NetGen permet aussi d'intégrer des systèmes physiques cellulaires géographiques en haut à droite. Le modèle structurel généré (Topologie) est un système de processus maillé selon un voisinage. Les comportements sont cette fois des fonctions de transitions. La finalité est la simulation.

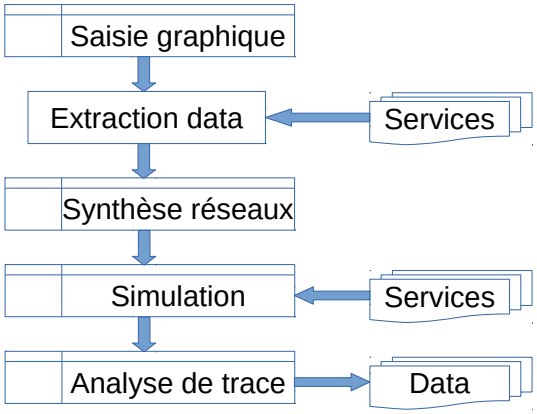


FIGURE 4.3 – Interactions du flot de simulation avec des bases externes : extraction des tuiles de cartes, ou de photos, extraction d'élévations ou autres informations géo-localisées, production de résultats géo-localisés, par exemple, zones impactées ou éclairées.

2. **La structure logique du réseau.** Le placement des nœuds, permet de déduire la connectivité entre chaque entité du réseau. Le réseau est ainsi conceptualisé en un système de processus concurrents.
3. **Le comportement du réseau.** Le modèle d'exécution du système est « synchrone ». Le fonctionnement collectif conduit le comportement local, et donc les acquisitions et les calculs à réaliser sur chaque nœud. Le comportement global est un programme distribué déduit de la composition des programmes locaux et des communications entre les nœuds.
4. **La simulation du déploiement.** Les algorithmes collectifs sont indépendants de la topologie, qui est découverte de manière dynamique. On peut faire varier les algorithmes et la topologie indépendamment les uns des autres. On peut ainsi exploiter des topologies arbitraires. L'objectif est de réaliser rapidement une simulation réaliste d'un déploiement conforme à une méthodologie.
5. **La mise au point.** On analyse les résultats de simulation pour ajuster les deux degrés de liberté dont on dispose : (a) les algorithmes et (b) la topologie.

4.2 Représentation abstraite du réseau et générateurs

4.2.1 Classes supports du modèle

La représentation d'un réseau est exprimée sous la forme d'une structure de données dans l'environnement objet Smalltalk. Cette structure est une collection de nœuds représentant des processus. Pour fixer les idées, une classe NetworkNode se définit comme suit :

```
1 Smalltalk.AlgoDis defineClass: #NetworkNode
2         superclass: #{Core.Object}
3         instanceVariableNames: 'name␣outputLinks␣inputLinks␣processName
              ␣networkGraph␣'
4         classInstanceVariableNames: ''
```

Les collections de liens entrants et sortants sont conservées ou construites, et sont décrites comme suit :

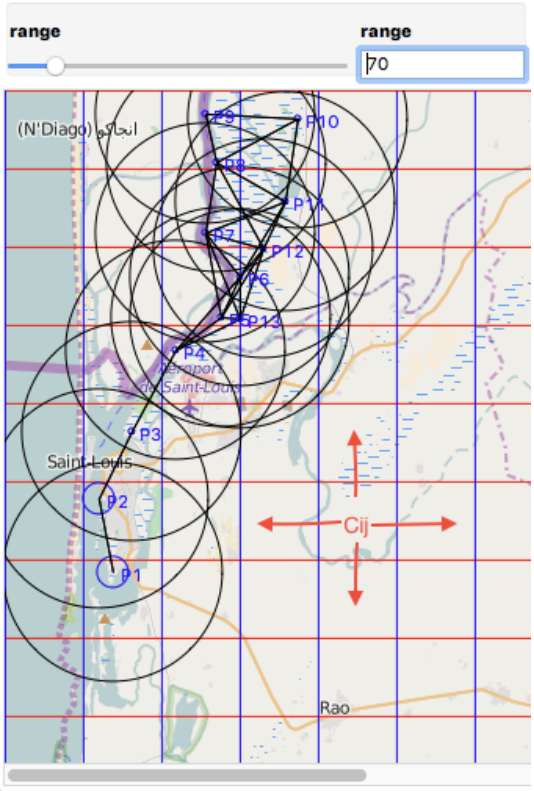


FIGURE 4.4 – Réseau de capteurs et réseaux cellulaires présentés dans l'interface PickCell : synthèse par calcul de portées uniformes à gauche, voisinage de Von Neumann à droite. Voir aussi les portées calculées en Line-of-Sight dans [17, 16]

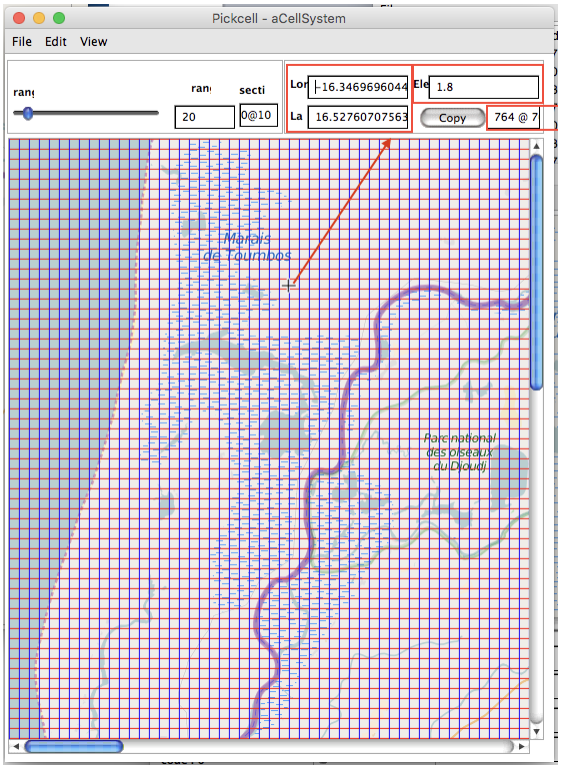


FIGURE 4.5 – Vue complète de l'interface PickCell présentant la géo-localisation, les élévations au niveau du pointeur, et les dimensions réelles des cellules. Ces informations sont produites par extrapolation des informations de QuickMap (projection Mercator), et par extraction des données NASA SRTM[22].

```
Smalltalk.AlgoDis defineClass: #NetworkLink
        superclass: #{Core.Object}
        instanceVariableNames: 'source target '
        classInstanceVariableNames: ''
```

Les variables désignées par *source* et *target* sont des objets de classe NetworkNode.

Le graphe du réseau est représenté dans une troisième classe NetworkGraph qui intègre une collection de nœuds (*nodes*) et des informations de gestion :

```
Smalltalk.AlgoDis defineClass: #NetworkGraph
        superclass: #{Core.Object}
        instanceVariableNames: 'initialConnectivity nodes roots
            networkName table maxProcessName statistics nodeDictionary
            sharedVariables includeFilename includeData '
```

Les outils produisent leurs descriptions en construisant sur les structures de données. Ce sont principalement des outils graphiques et en particulier les contrôleurs, mais l'usage est bien plus général. Il est en effet possible de traduire quantité de structures vers ce modèle et de profiter ainsi des générateurs.

4.2.2 Formes textuelles simplifiées

La forme textuelle la plus simple associe les noms des nœuds à leur sortance (fan-out), le nom d'un programme associé et des attributs géométriques variables.

Un cycle associant deux processus P1 et P9 s'écrit ainsi (Code 4.1) :

Code 4.1 – Forme textuelle du système P1-P9

```
biPoint
messages none  defined.
P1 { P9 } Node (531 @ 355) (100)
P9 { P1  } Node (553 @ 429) (100)
```

La représentation textuelle reflète l'organisation du réseau :

- un nom donné au réseau,
- une liste de messages qui circulent à travers les liens de communication,
- une ligne pour chaque nœud du réseau

Chaque ligne regroupe les informations suivantes :

- le nom du processus
- les voisins directement accessibles par les liens sortants
- le nom du programme, ou de la procédure exécutée par les nœuds
- les attributs du nœud (position sur la carte, portée de l'émetteur radio)

À partir de ce programme, un petit outil de compilation construit un modèle, qui peut ensuite être traduit en description graphique *graphviz* (figure 4.6)[23], ou en description en processus concurrent Occam (Code 4.2).

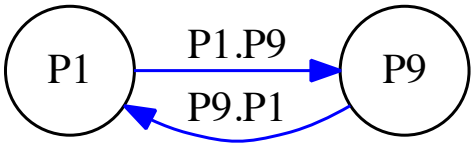


FIGURE 4.6 – Diagramme montrant un NetworkGraph de deux processus P1, et P9

Code 4.2 – Description du système P1-P9 en Occam

```
VAL [2][2]BYTE NetProcess IS [   "P1", -- id: 1
  "P9"]: -- Table des noms
VAL [2][4]BYTE NetProcedure IS [   "Node", -- id: 1
  "Node"]: -- Table des procédures
DATA TYPE Location -- structure geometrique
  RECORD
    INT xLoc:
    INT yLoc:
    INT elevation:
    INT range:
:
VAL [2] Location  NetLocation IS [   [531 , 355 , 0 , 100 ] , -- id: 1
  [553 , 429 , 0 , 100 ] ]: -- Table des données géométriques
#INCLUDE "nodes-test-include.occ"
PROC biPoint(CHAN OF BYTE stdin, stdout, stderr)
   -- Channel declarations
  CHAN OF diam.proto P1.P9:
  CHAN OF diam.proto P9.P1:
  -- Channel table declaration for nodes
  P1.out IS [ P1.P9 ]:
  P1.in IS [ P9.P1 ]:
  P9.out IS [ P9.P1 ]:
  P9.in IS [ P1.P9 ]:
  -- Program Body
  [MaxNodes]CHAN OF BYTE toMux:
  PAR -- description du systeme concurrent
    Node(P1.in, P1.out,0, toMux [0])
    Node(P9.in, P9.out,1, toMux [1])
    Mux(toMux,stdout)
     -- End of program body
:
```

4.2.3 Description Occam de l'architecture

La description Occam de l'architecture est générée automatiquement par les outils NetGen. Le petit système de l'exemple est ainsi représenté par deux processus qui communiquent entre eux par des canaux directionnels.

Le code 4.2 présente successivement des tables conservant les noms des processus, et le code qu'ils exécutent (lignes 1 et 3), la structure de données initiales (ligne 5), puis les valeurs de cette structure pour chaque processus (ligne 12).

La ligne 14 permet d'intégrer un fragment de programme décrivant le comportement des processus. Ici ce fichier nommé *nodes-test-include.occ* contiendra la procédure Node fixant les contributions des processus.

La construction parallèle démarre ligne 15, avec à suivre les déclarations de canaux de communication (ligne 17), puis les groupes de canaux entrants et sortants. Le PAR ligne 26 active les processus dans lesquels on reconnaît un processus traceur Mux. Les sorties en graphes facilitent la lecture de l'architecture si celle-ci n'est pas trop complexe (figure 4.7).

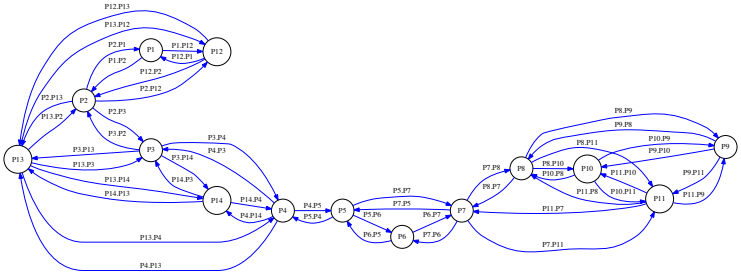


FIGURE 4.7 – Diagramme montrant le graphe correspondant au réseau de la figure 4.11

4.3 Simulation : le comportement du système

4.3.1 Organisation des programmes de simulation

Le modèle interne permet de composer des capteurs au sein d'un réseau, avec des liens pair-à-pair qui représentent les communications sans fil. Les outils de NetGen peuvent générer des réseaux aléatoires, avec certains paramètres fixés par l'utilisateur, ou bien des réseaux maillés. Les générateurs de code produisent des fichiers de code Occam ou CUDA, en plus des formes graphiques Graphviz. En partant des interfaces graphiques, on parvient très rapidement à produire des organisations concurrentes dans ces deux syntaxes.

Un programme complet, pour un système cellulaire ou un système de capteurs, requiert cependant deux spécifications complémentaires :

- le comportement algorithmique. Il s'agit typiquement d'automates déroulant un cycle de capteur : acquisitions, communications, changements d'états. Ce comportement peut s'exprimer dans un des deux langages utilisés actuellement.
  Les contributions algorithmiques distribuées et les règles de transition des automates cellulaires peuvent s'exprimer dans ce cadre.
- des données associées à chaque processus. Elles intègrent des spécifications de types structurés, et des valeurs associées à ces types.

Dans le cas des réseaux de capteurs, il est ainsi possible de caractériser des traitements ou des fonctions d'acquisition. Dans le cas des systèmes cellulaires on trouve des données géo-localisées dont les représentations graphiques des cellules.

Les générateurs de code permettent d'intégrer les comportements et les données dans les fichiers architectures qui constituent le point d'entrée de la simulation.

On peut noter que les trois composantes sont indépendantes, favorisant une réutilisation du code de comportement sous forme de librairies ou de données, permettant de faire varier les circonstances des simulations.

4.3.2 Sémantique synchrone

Les programmes obéissent à une sémantique synchrone. Celle-ci peut se justifier par une logique d'échantillonage distribué appliquée à un processus physique discret ou continu. Cette logique de séquencement est aussi présente dans les logiciels de programmation comme Arduino ou TinyOS, et elle s'intègre aux réseaux radio dédiés aux capteurs qui forcent des communications périodiques sur rendez-vous.

Enfin, lorsque l'on considère les simulations physiques, il existe beaucoup de travaux utilisant les automates cellulaires pour induire des comportements globaux. Les automates cellulaires progressent en discrétisant le temps et en procédant à des échanges locaux qui représentent les influences physiques.

La présence d'un temps de référence a une autre propriété qui est la possibilité d'effectuer des transferts entre plusieurs simulations concurrentes, ou de séquencer ces simulations sur la base de ces échanges. Cette dimension de composition a été explorée par Hoang [19] en combinant réseau et simulation physique dans un cadre *High Level Architecture* (HLA) [24].

L'approche synchrone permet donc de corréler les évolutions physiques et leurs dispositifs d'observation, permettant aussi d'optimiser les efforts énergétiques en fonction des objectifs de l'observation.

4.3.3 Simulation synchrone

Les deux cadres de simulation proposés sont les systèmes de processus communicants et les accélérateurs graphiques apparentés à des exécutions *Single Instruction Multiple Data* de CUDA.

Dans le premier cadre les progrès de l'exécution sont liés à des stimuli externes ou internes au système, qui progresse à la demande. Concurrent Sequential Processes (CSP) est le cadre dans lequel le langage Occam a été créé [25] [2]. CSP a pu être utilisé pour programmer des systèmes matériels formés de circuits asynchrones [27], et peut donc se passer de la notion de temps. À l'inverse, des auteurs s'en sont aussi servi pour décrire des circuits utilisant une ou plusieurs horloges, par exemple pour les circuits reconfigurables [28].

La compilation est effectuée par la chaîne KRoC de l'université de Kent conçue par l'équipe de Peter Welch au *Computing laboratory* [29] [30].

Cette chaîne intègre un noyau d'exécution concurrente permettant d'utiliser les architectures multiprocesseurs sur des programmes qui peuvent contenir des centaines de processus légers. Cette chaîne permet aussi d'utiliser des fermes de machines en passage de messages, ou de cibler une machine virtuelle embarquée nommée TVM. La programmation en Occam est bien structurée, le compilateur vérifie la correction des architectures proposées, le *runtime* détecte des erreurs et

2. C.A.R. Hoare [26]

signale leur localisation. C'est donc un point d'entrée très commode pour la prise en main des outils NetGen. À l'inverse, les architectures sont plutôt statiques que dynamiques, ce qui pose problème pour la mobilité, par exemple. Il est cependant possible de faire évoluer les connexions en utilisant les services étendus de Occam-PI [31].

Les simulations sur accélérateur graphique bénéficient du séquencement implicite donné par l'architecture SIMD. Il suffit simplement de *compter les tours* dans les boucles parallèles (kernels au sens de CUDA). Les programmes sont structurés par des appels de fonctions archivables et réutilisables. On peut en attendre des performances très importantes pour plusieurs raisons :

- le grand nombre de processeurs (plus de 2 000 dans les dernières générations),
- la localité des échanges menés sur mémoire partagée (plusieurs Giga-octets).

On trouve des éléments chiffrés dans [32] et [16].

Les outils utilisés sont la chaîne CUDA proposée par NVidia pour ses accélérateurs. Le cycle d'exécution implique le transport des données en mémoire du GPU, l'activation d'étapes de calcul, et le retour des données ou résultats vers la mémoire centrale. La syntaxe de *nvcc* est proche de celle de C, les structures de données et les tables étant tout à fait similaires à celles produites pour Occam.

Ces architectures sont flexibles, il est en effet possible de lire et écrire les données en mémoire graphique par des opérations de transfert avec la mémoire centrale. On peut donc suivre l'évolution des programmes d'autant plus facilement que les modèles de données sont connus et interprétables dans les outils.

Les simulations Occam de réseaux d'observation impliquent d'introduire un séquencement que l'on peut associer aux rendez-vous de communications dans le système d'observation. Les communications Occam étant bloquantes, les échanges implémentent une synchronisation des processus par les barrières qu'elles construisent. On rejoint ici la conception d'algorithmes distribués par échanges de messages synchrones [33]. Cette méthode consiste à décrire les programmes distribués sous forme d'automates qui exécutent des cycles selon un motif systématique :

- **initialisation**
  initialisation des variables d'état, préparation des messages sortants.
- **bouclage**

  - **communications** C_i messages entrants et sortants échangés avec les voisins
  - **changement d'état** E_i l'état évolue en fonction des données reçues et acquises. On produit les données sortantes pour C_{i+1}

Occam permet de décrire ce code de manière générique par rapport à l'architecture du réseau (voir Code 4.3 qui correspond au fichier nodes-test-include.occ dans l'architecture, Code 4.2, ligne 14).

Code 4.3 – Comportement applicable au système P1-P9 en Occam. Calcul du maximum des id sur un rayon 1

```
1 PROTOCOL diam.proto IS INT:
2 PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
3   [MaxFanOut] INT inBuf,outBuf:
4   INT MaxId:
5   SEQ
6     MaxId := id -- initialisation
7     SEQ link=0 FOR SIZE out
8       outBuf[link] := MaxId
9     SEQ turns = 0 FOR 1 -- bouclage
```

```
10        PAR -- communications
11          PAR inLink = 0 FOR SIZE in
12            in[inLink] ? inBuf[inLink]
13          PAR outLink = 0 FOR SIZE out
14            out[outLink] ! outBuf[outLink]
15      SEQ inLink = 0 FOR SIZE in -- changement etat
16        IF
17          inBuf[inLink]>MaxId
18            MaxId := inBuf[inLink]
19          TRUE
20            SKIP
21      out.number(MaxId,4,toMux) -- trace
22      toMux ! '*n'
23  :
24
25  PROC Mux([] CHAN OF BYTE fromProc, CHAN OF BYTE out)
26    BYTE ch:
27    SEQ
28      SEQ turns=0 FOR SIZE fromProc
29        ALT link=0 FOR SIZE fromProc
30          fromProc[link] ? ch
31            SEQ
32              out ! ch
33              WHILE ch<>'*n'
34                SEQ
35                  fromProc[link] ? ch
36                  out ! ch
37  :
```

4.3.4 Annotation des interfaces en exécution concurrente

La figure 4.8 rappelle le flot de conception, de la prise en charge applicative à la simulation et l'exécution.

Nous avons vu qu'un générateur de code produit un réseau simulable de deux manières :

- un code Occam [34], exécutable sur des processeurs multi-cœurs, avec un parallélisme à grain fin (papier de A. Iqbal et B. Pottier [35]) ;
- un code Cuda [36], exécutable sur des processeurs graphiques, dont les communications se font à travers la mémoire partagée (papier de DYROS [32] et master de T.Failler [37]).

Dans les deux cas, il est nécessaire de définir comment s'effectuent les interactions avec la simulation. La première de ces interactions est l'observation des évolutions distribuées. Celle-ci peut s'effectuer via une trace d'exécution (voir la table 4.1).

La figure 4.9 montre une construction où la sortie de la simulation est passée sur un *pipe* vers l'interface de haut niveau qui a permis de générer le simulateur. Dans un simulateur Occam, le comportement concurrent est renvoyé par des canaux liant chaque processus à un multiplexeur chargé d'évacuer les informations vers un *pipe*. Le programme Code 4.3 montre d'une part la construction de la trace, lignes 21 et 22, et d'autre part la structure du multiplexeur, lignes 25 et suivantes.

L'analyse de la trace et la présentation qui en découle est alors effectuée à l'intérieur d'une interface de simulation (PickCell, GMap, QuickMap), à l'aide d'un processus de haut niveau qui

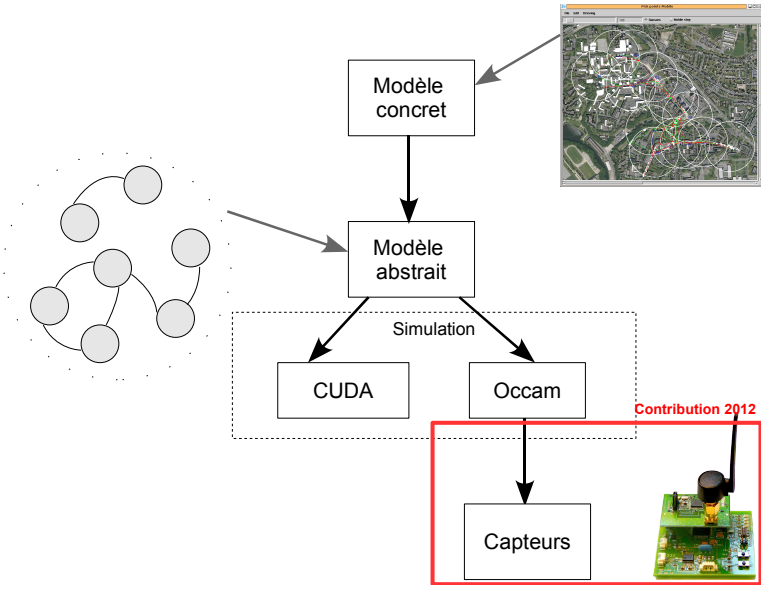


FIGURE 4.8 – Le flot méthodologique de NetGen. En haut, la maquette de déploiement : photos, contours des bâtiments, portée grossière, réseau. Au milieu, le modèle abstrait. En bas, la synthèse de code pour les capteurs.

décode la trace, et provoque des actions de rafraîchissement visuel.

4.3.5 Modèles et contrôles pour l'exécution massivement parallèle

Dans le cas de CUDA, le couplage entre l'environnement et le simulateur peut être bien plus fort que le couplage par *pipe*. Les structures de données de l'application sont connues, elles font partie du comportement de la simulation. En pratique il s'agit de déclarations C que l'on sait traduire en langage objet, et utiliser à partir du langage objet [3]. Créer des variables dynamiques correspondant aux structures CUDA, les copier vers l'accélérateur, les relire, interpréter leurs contenus, n'est donc pas un problème. De même des fonctions C implantées dans une librairie dynamique sont utilisables

3. Smalltalk/Visualworks propose des outils d'intégration C dans le paquetage DLL&C Connect

identité	diamètre	leader
13	7	13
0	7	13
1	7	13
...	...	...
10	7	13
11	7	13
12	7	13

TABLE 4.1 – Trace d'exécution du réseau figuré en 4.11. La première colonne est l'identité du processus, la seconde le diamètre calculé dans le réseau, la troisième l'identité du leader. Ces calculs sont menés à bien au démarrage du réseau et permettent le dimensionnement des algorithmes : nombre de pas, nombre de réseaux.

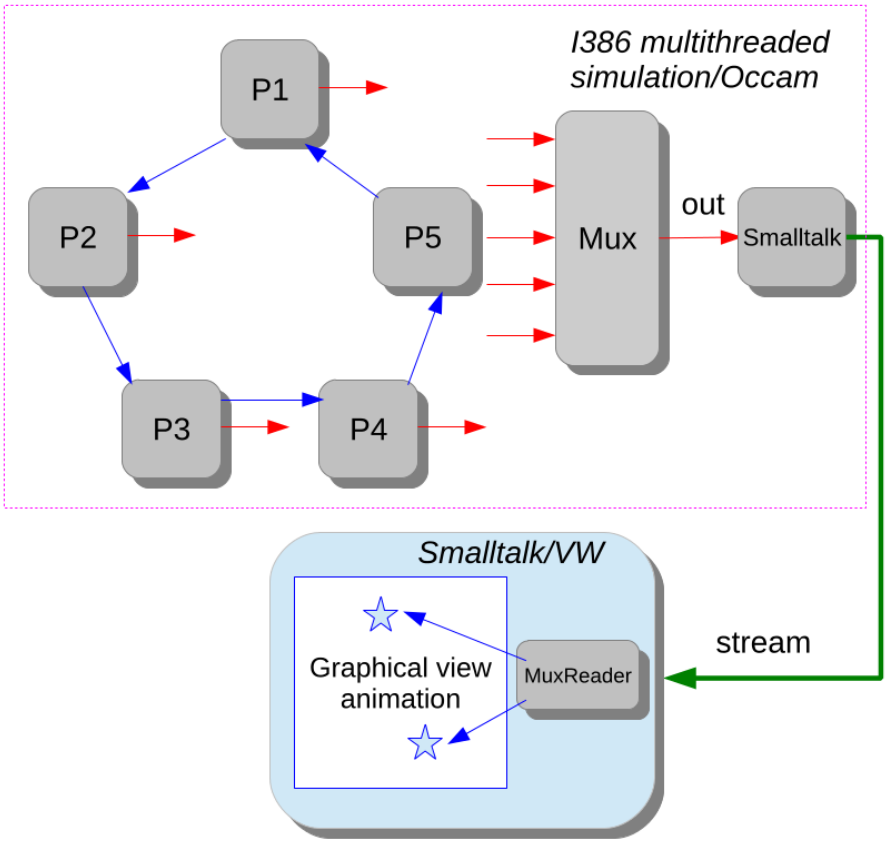


FIGURE 4.9 – Renvoi des traces d'exécution vers l'interface graphique

dans l'environnement objet qui peut importer ou exporter des données, démarrer des tâches, ou modifier les structures définissant le réseau.

Si on se réfère à la figure 4.9, l'application Smalltalk du bas, voit donc les processus de la simulation, peut lire et modifier leurs états. La figure 4.10 montre une interface réalisée dans le cadre de l'étude de mobilité du chapitre 5. Chaque nœud est présenté sur une ligne, avec quelques états notables présentés en colonne. Cette interface interagit directement avec la mémoire de l'accélérateur, et elle est bien entendu programmable pour poursuivre des conditions complexes.

4.3.6 Mobilité et contrôle de la simulation

Un mobile peut se déplacer dans le réseau et entrer en communication avec les nœuds fixes. Les chemins de mobiles sont décrits dans des classes permettant de gérer une succession de positions avec d'autres attributs. La figure 4.11 présente une facilité de saisie définie par dessus une fenêtre PickCell.

L'intersection d'un chemin de mobilité avec un graphe de réseau intègre des évènements de rencontre et de rupture du mobile avec un réseau existant. Ces points de rencontre peuvent être calculés dynamiquement ou statiquement. La figure 4.11 présente un réseau sans fil à terre, et le graphe des rencontres (figurées en vert) et ruptures du chemin avec ce réseau (figurées en rouge) [31].

La simulation du mobile procède en parcourant le chemin. La classe *ExecutionManager* permet

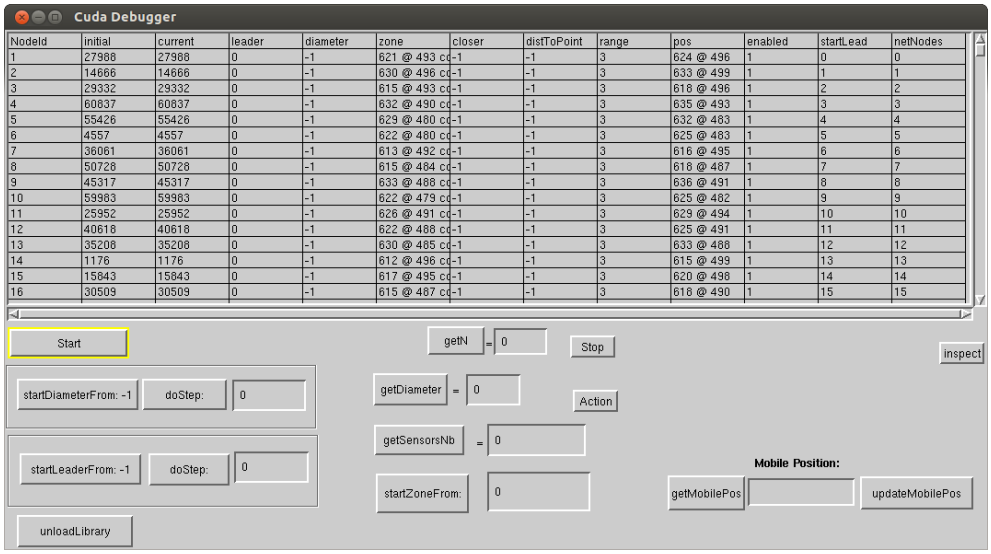


FIGURE 4.10 – Accès aux données sur l'accélérateur

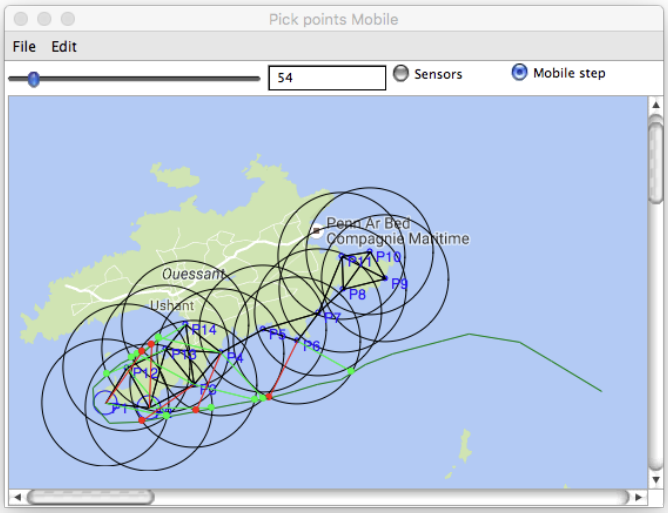


FIGURE 4.11 – Fenêtre PickCell de saisie de chemin de mobiles montrant les rencontres entre le chemin de mobiles, et le réseau de capteurs statique. Le chemin du mobile est représenté en vert sombre.

de piloter les progrès de la mobilité en déclenchant des opérations. Dans le cas d'une simulation de type CUDA, on va synchroniser l'état de la mémoire sur l'accélérateur en fonction des changements d'états impliqués par les progrès : établir des canaux de communication (LinkCreation), ou supprimer ces canaux lors des ruptures (LinkDestruction).

Entre deux opérations, le réseau est considéré comme statique et peut fonctionner normalement.

4.4 Programmation des capteurs

La partie basse du flot consiste à assembler deux éléments : (i) la contribution distribuée, (ii) le contrôle et l'acquisition sur les capteurs locaux (figure 4.12). Le premier élément a été validé par les simulations décrites dans ce chapitre. Les algorithmes qui y sont implantés sont :

- des fonctions d'ajustement aux caractéristiques architecturales, comme les calculs de diamètres, l'étiquetage des différents réseaux (leaders),
- des fonctions de transferts de données, vers des drains aux marges du réseau, par exemple,
- des fonctions d'intégration des données dans une localité spatiale, calculs de maximums, calculs de moyennes, seuillages…
- les déclenchements d'alertes, la sécurité.

Dans la structure actuelle du flot, ces algorithmes sont décrits en C lorsque l'on simule sur CUDA, en Occam, lorsque l'on simule des systèmes de processus. En conformité avec les simulations de la partie réseau, partie haute, l'expression des contributions d'acquisition locale la plus naturelle serait de procéder également en Occam ou C.

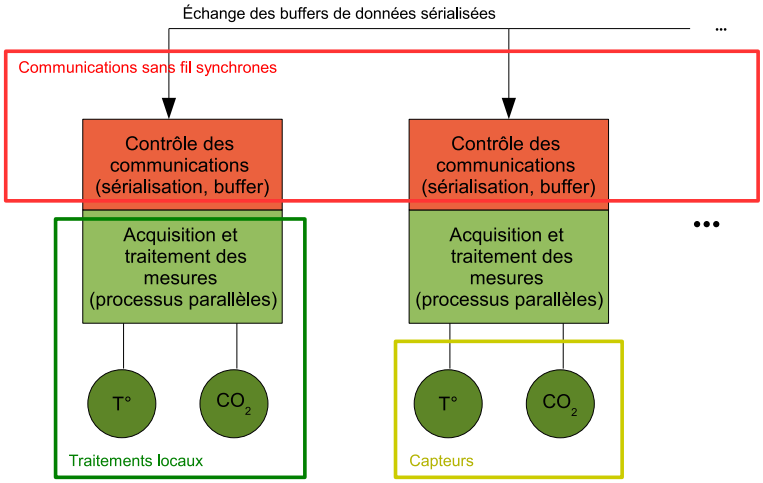


FIGURE 4.12 – La figure montre l'assemblage de deux nœuds via un système radio. Les contributions locales (acquisition et contrôle en vert) sont disjointes des contributions distribuées (en rouge).

Ces capteurs sont souvent réalisés à base de micro-contrôleurs typiquement basse consommation. Nous avons pratiqué les MSP430, AtMega 2560, Jennic, ARM, les systèmes reconfigurables de Cypress. Ces architectures comportent une variété de mémoires internes, des interfaces d'acquisition et de contrôle, des bus périphériques. La figure 4.13 montre l'architecture d'un MSP430 avec ses mémoires et ses unités périphériques. La réduction de la consommation est obtenue par gradation des activités en éteignant les ressources logiques inutiles.

Programmer signifie produire des logiciels embarqués pour des environnements contraignants et qui requièrent un apprentissage qui peut être long et complexe. Si le nombre des capteurs disponibles ne cesse d'augmenter, les micro-contrôleurs sont aussi très nombreux et hétérogènes. Dans une application environnementale on interrogerait, par exemple, des capteurs physiques pour la température, l'hygrométrie, la concentration de gaz, ou la détection de présence par infrarouge. Ces tâches peuvent être menées en parallèle, et séquencées avec des fréquences variables. Les pré-traitements peuvent être très simples, ou très complexes et étalés dans le temps (reconnaissance de signaux, d'images). Les besoins de la programmation locale sont donc très spécifiques, concurrents, et le séquencement des opérations y joue un rôle important.

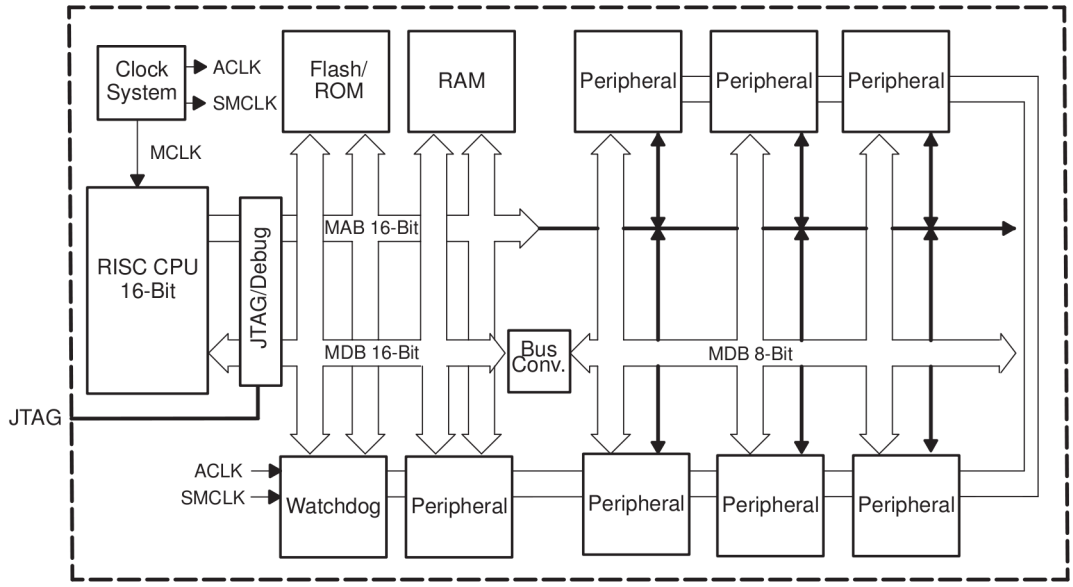


FIGURE 4.13 – Architecture d'un micro-contrôleur de type MSP430. (source Texas Instruments)

TABLE 4.2 – Nomenclature des périphériques du MSP430 et de Atmel utilisés dans les développements présentés en chapitre 2

Périphériques de communication	MSP430 F1612	ATMEGA 2560
Périphériques génériques d'entrée/sortie	48	54
Convertisseurs Numérique/Analogique (CNA)	2	0
Convertisseurs Analogique/Numérique (CAN)	8 (12 bits)	16 (10 bits)
Canal *Pulse Width Modulation* (PWM)	16 (16 bits)	16 (16 bits)
Inter Integrated Chip (I2C)	2	1
Serial Peripheral Interface (SPI)	1	1
Universal Asynchronous Receiver Transmitter (UART)	2	4

Plusieurs modèles de programmation sont connus tel que la programmation à bas niveau en C (MSP430), l'usage d'un micro-système d'exploitation (TinyOS [38]), ou des Environnements de Développement Intégrés (IDE), tels qu'Arduino et sa version commerciale Libelium (figure 4.14). Arduino s'appuie sur une syntaxe proche du langage C++, avec une organisation de programme représentant un automate qui, après une initialisation, exécute un cycle d'actions de communication et de capture. Le programme ne se termine pas, il est cyclique. Arduino propose un modèle de programmation proche des besoins des capteurs dans lequel il est possible d'insérer des accès au réseau.

Nous avons étudié et pratiqué plusieurs de ces plates-formes, en concluant qu'elles sont perfectibles en terme de *sécurité* et de *fiabilité* de la programmation. L'idée est cependant venue d'étudier Occam pour plusieurs raisons :

- les capacités à modéliser la concurrence locale, inhérente aux capteurs,
- la vérification possible de programmes concurrents donc complexes,
- la miscibilité avec les activités distribuées, permettant des vérifications au niveau système,
- la robustesse des concepts appuyés sur *Communicating Sequential Processes*, et ses nombreuses variantes d'implémentation, y compris matérielle [28].

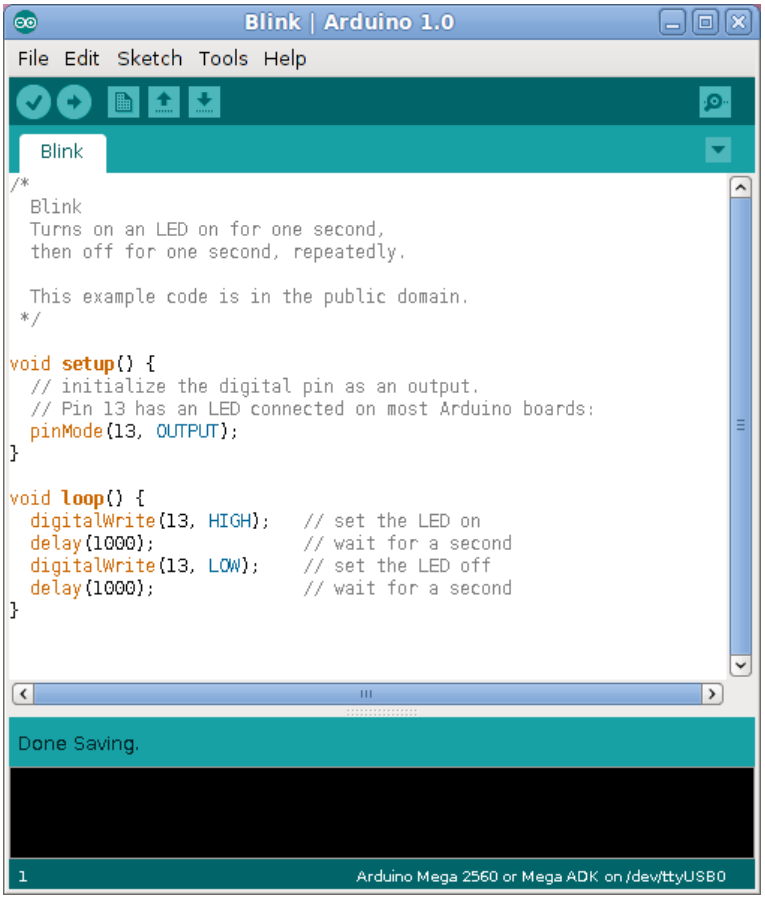


FIGURE 4.14 – L'environnement de développement intégré Arduino permet de compiler le code et programmer un système embarqué. Deux fonctions structurent le code : la fonction `setup()` pour l'initialisation, et la fonction `loop()` pour l'exécution.

- la disponibilité d'une machine virtuelle, à même de faciliter les portages sur une grande variété de matériels.

En comparaison des environnements basés sur C, les aspects de programmation à haut niveau d'Occam facilitent le développement des algorithmes : la concision de l'écriture et la vérification lors de la compilation rendent leur exécution plus robuste.

4.4.1 Occam : le langage idéal pour la programmation des capteurs ?

Le langage Occam privilégie les constructions concurrentes (PAR), la gestion du temps (TIMER), le non-déterminisme (ALT), les synchronisations (CHAN). Tout ceci renforce la fiabilité et la lisibilité de la programmation des systèmes embarqués. En couplant le modèle synchrone d'exécution de Netgen et la programmation concurrente des capteurs, on obtient une maîtrise analytique du déroulement temporel sur le capteur, et dans le système distribué. Ceci permet de se rapprocher de l'expression de solution à des problèmes physiques qui font intervenir la notion d'espace et de temps. Edward Lee (Berkeley) s'est fait l'avocat de ce domaine : « *In the physical world, the passage of time is inexorable and concurrency is intrinsic.* [14] »

TABLE 4.3 – Squelette de programme Occam. Le programme principal `CapteurSysteme` appelle deux procédures, `Acquerir` et `Communiquer`. Entre chaque séquence d'appel, le programme se met en veille. Les mots-clés `SEQ` et `PAR` spécifient les actions à exécuter en séquence ou en parallèle respectivement.

```
1  PROC CapteurSysteme                1  PROC Acquerir                      1  PROC CommuniquerTDMA
2    WHILE TRUE                       2    PAR i FOR nCapteurs              2    PAR
3      SEQ                            3      SEQ                            3      Diffuser ()
4        Communiquer ()               4        Lire (i)                     4      SEQ i FOR mVoisins
5        Acquerir ()                  5        Traiter (i)                  5        Recevoir (i)
6        Pause ()                     6                                     6
7  :                                  7  :                                  7  :
```

4.4.2 La chaîne de programmation KRoC

L'Université de Kent a développé la chaîne de compilation KRoC pour le langage Occam[29]. Cette chaîne produit un code intermédiaire *occ21* pour une machine abstraite orientée processus. Ce code peut être exécuté par une machine virtuelle *Transputer Virtual Machine* (TVM) [39] ou traduit en code microprocesseur natif. La figure 4.15 montre un programme formé de bytecodes pris en charge par TVM ou traduit en code x86 multi-tâches.

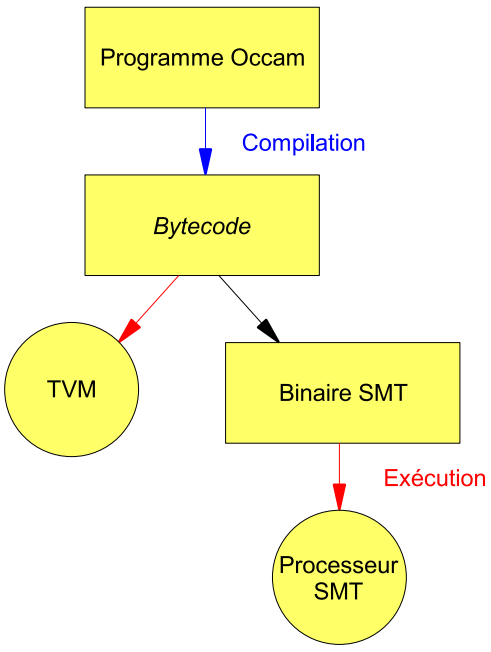


FIGURE 4.15 – Flot de compilation d'un programme Occam

NetGen utilise le code x86 pour simuler le réseau de capteurs, les bytecodes peuvent eux supporter une exécution sur un système embarqué.

4.4.3 Un flot de développement orienté processus et portable

Utiliser un programme Occam sur les capteurs sans fil rend homogène le flot de développement : un algorithme écrit pour le simulateur peut fonctionner sur le réseau de capteurs sans traduction vers un autre langage. C'est une garantie de fiabilité de fonctionnement.

La machine virtuelle TVM est décrite en C et est portable, elle est associée à des librairies Occam et peut être étendue via des primitives en langage C. Ainsi, elle ne requiert pas de composants matériels spécifiques comme le *Transputer* : elle peut notamment s'adapter à un grand nombre de systèmes embarqués. Un seul programme Occam peut donc cibler plusieurs plates-formes, il suffit de disposer de la machine virtuelle pour le système choisi.

La machine fonctionne sur le matériel cible en utilisant un adaptateur (*wrapper*) qui lui permet d'utiliser les ressources physiques, grâce à la cartographie des adresses mémoire. Quand le matériel change, seul l'adaptateur réclame un nouveau développement.

4.4.4 Le typage fort et la mise en paquet automatique

Occam est un langage fortement typé. La vérification de la cohérence des types des variables est faite lors de la phase de compilation. Ainsi, de nombreuses erreurs de programmation peuvent être détectées avant l'exécution du programme. Cela accroît la sécurité et la productivité du développement.

La notion de protocole (PROTOCOL) permet de spécifier les variantes de types circulant sur les canaux et de contrôler la production et le construction de ces données. En utilisant les protocoles, il est aisé de produire les paquets qui vont effectivement circuler sur le réseau.

4.4.5 Le comportement lors d'une erreur d'exécution

Le système est paramétrable pour décider du comportement à adopter en cas de défaillance d'un processus lors de l'exécution [40] :

- le processus en cause est arrêté : le reste du système peut continuer à fonctionner tant qu'il n'est pas dépendant du processus en cause. Dans cette perspective, on peut construire un système redondant, où plusieurs processus réalisent une même tâche. Le système conserve un fonctionnement normal après qu'un de ces processus soit arrêté.
- le système entier s'arrête : ceci est utile pour la mise au point des programmes.
- l'erreur est ignorée : le compilateur peut optimiser les programmes qui ont été prouvés corrects.

4.5 Conclusion

Ce chapitre a présenté un flot de conception complet, de l'application à sa réalisation concrète. La quasi-totalité de ce flot a été implémentée, avec des caractéristiques de développement rapide. Nous avons contribué à plusieurs éléments de la plate-forme comme cela a été montré au chapitre 2, et c'est particulièrement le cas pour la programmation sur la plate-forme TVM.

Les chapitres qui suivent vont développer les aspects distribués et la mobilité d'une part (chapitre 5), l'intégration matérielle et logicielle sur TVM d'autre part (chapitre 6).

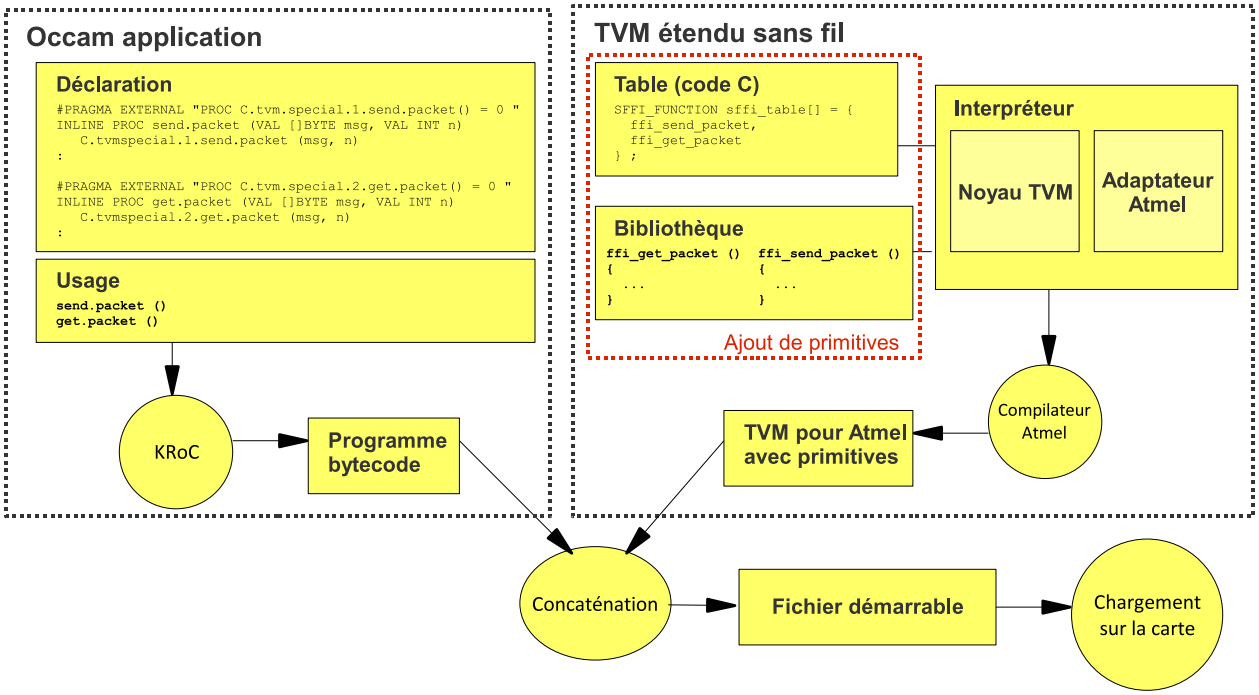


FIGURE 4.16 – Flot de développement d'un programme Occam sur Arduino. La machine virtuelle TVM est extensible : à droite, on ajoute les primitives écrites en langage C pour piloter un matériel spécifique de la plate-forme. Par exemple, les primitives de communication radio `ffi_get_packet()` et `ffi_send_packet()` utilisent le modem XBee. À gauche, les points d'entrée dans le programme Occam grâce aux directives `#PRAGMA` analysées par le compilateur KRoC. L'appel d'une primitive de TVM se fait ensuite de la même manière qu'une procédure Occam classique. Un seul fichier, contenant la machine virtuelle et le programme compilé, est chargé sur la carte.

5

Capteurs, mobiles et algorithmes distribués

Les systèmes de communication les plus développés sont les réseaux de données Internet, les réseaux téléphoniques avec leurs variantes numériques et en particulier les réseaux cellulaires. En ce qui concerne les terminaux mobiles, la banque mondiale donnait pour la France le chiffre de 73.7 millions d'unités.

Les réseaux sans fil occupent donc une place très importante, car ils nécessitent seulement une infrastructure légère. Certains d'entre eux peuvent de plus être déployés et déplacés facilement, c'est le cas des réseaux de capteurs.

L'acquisition de données dans l'environnement va poser d'autres problèmes. C'est en particulier le cas des dispositifs de surveillance de zones difficiles d'accès : zones isolées ou dangereuses, déserts, océans, zones volcaniques. Dans ces cas un recours est de procéder par le haut, avec l'utilisation de satellites, de ballons ou de drones variés (*Unmanned Aerial Vehicle* ou UAV). De part leur trajectoire prédictible, dans le cas des satellites, ou contrôlée, dans le cas des drones, les mobiles survolent les champs de capteurs et permettent ainsi de collecter de façon automatique des données localisées.

Pour cette raison, nous nous sommes intéressés aux interactions entre les mobiles et les réseaux de capteurs. Ici les liens de communication sont nécessairement dynamiques, mais ils peuvent être considérablement plus efficaces en raison de l'absence d'obstacle. Les métriques les plus importantes sont les calendriers de visite, la durée possible des interactions et les distances sol-sol et sol-air. La planification de ces réseaux est de toute évidence un problème difficile, mais les bénéfices peuvent être très importants. Ce sera l'objet de l'étude présentée dans ce chapitre, étude coopérative publiée avec des acteurs à l'Université de Brest[7, 41].

Nous allons nous intéresser plus particulièrement aux satellites, et en particulier ceux qui circulent en orbite basse en balayant la surface de la Terre.

Il existe actuellement plusieurs systèmes de satellites à usage commercial utilisés pour l'environnement, ce sont Argos et Iridium. Le système Argos permet le positionnement de balises, la surveillance, quelquefois à des fins d'étude et de protection de l'environnement et de la faune. Le réseau Iridium propose un service de télécommunication utilisable par les agences gouvernementales, comme les particuliers. Les deux systèmes forment chacun une constellation de satellites en orbite basse, entre 700 et 900 km d'altitude.

Les satellites ont souvent une fonction *d'observation* : état de la mer, couvertures nuageuses ou autres. Ils ont aussi la capacité d'appréhender de l'information numérique et de déposer des informations au sol. Les deux types de relations, physique et numérique, constituent des *couvertures* tout à fait analogues à celles définies pour les réseaux de capteurs : couverture radio et couverture observatoire. L'empreinte radio au sol peut atteindre plusieurs milliers de kilomètres de diamètre, utilisant des liaisons montantes, descendantes et croisées, et fournissant une haute disponibilité de communication.

Les liens radio utilisés sur les satellites basse orbite sont en évolution et peuvent être résumés comme suit :

- Ces systèmes utilisent des antennes sur la bande L (fréquences radio de 1 à 2 GHz), pour répondre aux exigences de haute performance, basse consommation, taille compacte et faible coût.

5.1 Modèles pour la simulation des UAV et satellites

Les mobiles aériens ont deux types de comportement. Ils peuvent suivre des trajectoires prédéfinies, c'est le cas des satellites, ou ils peuvent être pilotés, voire auto-pilotés, c'est le cas des drones. On associe à ces trajectoires la notion de *chemin*. Un chemin est une séquence de positions géométriques ou géographiques associées à des estampilles temporelles.

Un chemin, ainsi défini d'une manière discrète, peut aussi dériver de lois physiques, balistiques par exemple. Les satellites ont leurs mouvements définis par un système de paramètres et des équations dont on connaît une expression algorithmique : "*Keplerian elements*" [42].

Dans le cas des UAV, on peut simplement se référer à une ligne brisée dont les points sont à 4 dimensions (x, y, z, t) et représentent une mission prédéfinie.

La composition des chemins de mobiles avec les champs de capteurs procède de calculs géométriques basés sur la visibilité radio réciproque. Ces calculs définissent des réseaux dynamiques intégrant le mobile lors de certains intervalles de temps. La figure 5.1 présente ainsi les points de contact et de rupture du chemin d'un satellite avec un réseau fictif déployé sur Madagascar [1].

5.2 Chemins discrétisés, statiques et dynamiques

Les paramètres de beaucoup de satellites sont connus publiquement. Une base de données publique permet ainsi de suivre et prédire les mouvements [43]. Plusieurs applications interprètent cette base de données. C'est le cas de GPredict que nous avons utilisé en association aux outils du laboratoire présentés au chapitre 3.

La figure 5.2 est une vue de l'interface, cet assemblage présentant une révolution complète d'un satellite LEO. Le décalage de positions sur cette révolution correspond à la rotation de la Terre durant la période du satellite : quelques milliers de kilomètres pour une période d'environ une heure et demie.

Les chemins d'UAV peuvent eux être spécifiés interactivement d'une manière similaire à la saisie des positions de capteurs avec les conversions géométriques nécessaires. Ils peuvent aussi être calculés de manière à optimiser un parcours de collecte, sur un champ de capteurs par exemple, ou ils peuvent être saisis lors d'un voyage contrôlé à l'aide d'un GPS (figure 5.3).

1. Le logiciel de calcul que nous avons réutilisé est décrit dans le rapport [31].

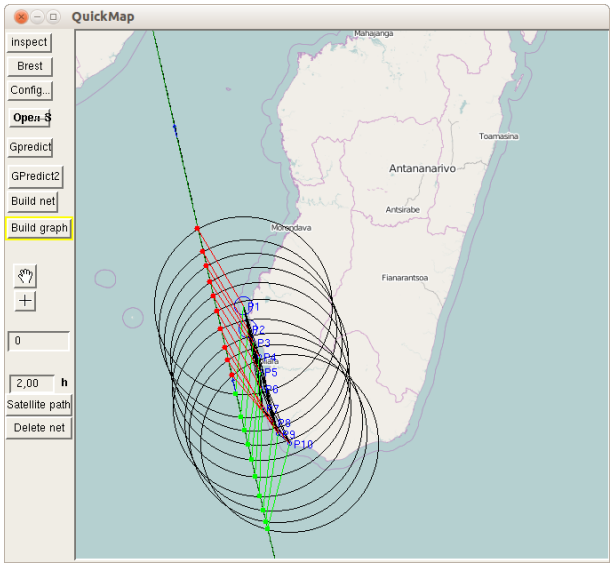


FIGURE 5.1 – Chemin satellite sud-nord et champ de capteurs dans QuickMap : les lignes et points verts représentent l'établissement de liens radio, les lignes et points rouges représentent les ruptures de communication.

Dans la suite de ce chapitre, on suppose l'existence d'un mécanisme de simulation permettant d'interpréter les mouvements du mobile en regard des champs de capteurs. Un tel mécanisme peut reposer sur des techniques de simulation guidée par le temps, ou correspondre à une exécution en temps réel.

Les connexions et déconnexions du satellite avec le réseau de capteurs peuvent être calculées statiquement, en prenant en compte tous les éléments mobiles du réseau. Chaque événement est représenté par une position géographique portée sur la carte, associée à sa date.

Le flot de conception et de simulation utilise actuellement la simulation sur accélérateur GPU contrôlée à partir d'un interface de haut niveau en Smalltalk (voir chapitre 4). L'exécution finale est donc effectuée à partir d'un code CUDA. Cependant, il est bien plus facile et sûr de décrire ces algorithmes en Occam, en payant ensuite le prix d'une réécriture en CUDA. Cette réécriture pourra être rendue automatique ultérieurement.

Pour fixer les idées, le flot amenant à la situation décrite en figure 5.4 est actuellement le suivant :

1. **QuickMap** intégration d'un chemin de mobile, par exemple un satellite (figure 5.2).
2. **QuickMap** choix d'un champ de capteurs (figure 5.1).
3. **GMap** conditionnement du champ de capteurs, en particulier des portées radio.
4. **NetGen** production du réseau abstrait correspondant aux champs de capteurs.
5. **NetGen** production des simulateurs Occam et CUDA.
6. **Occam** spécification et mise au point des algorithmes distribués (ici calcul de boîte englobante).
7. **CUDA** reproduction de cette spécification, génération d'une librairie dynamique.
8. **NetGen** exécution du code CUDA associant le mouvement du mobile, sa répercussion sur le réseau installé dans l'accélérateur (connexions et déconnexions), transactions entre le mobile et les réseaux, représentation graphique des résultats (figure 5.4).

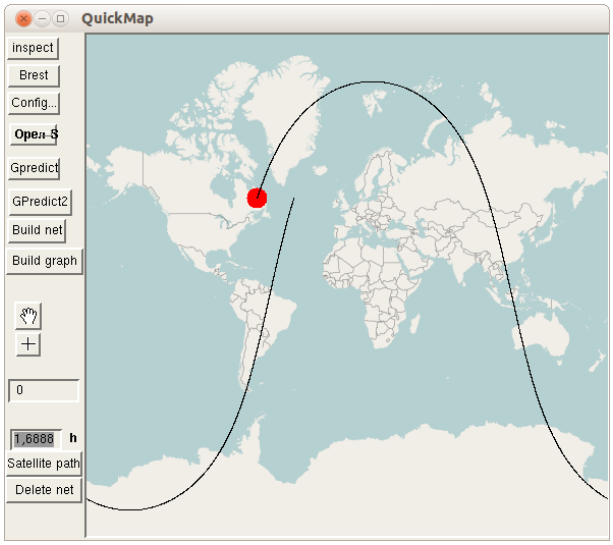


FIGURE 5.2 – QuickMap présentant un chemin de satellite LEO, avec le décalage horizontal consécutif à la rotation terrestre durant une heure et demie.

FIGURE 5.3 – Présentation d'un chemin (en bleu) saisi lors d'un parcours réel sur une ligne de bus. La tache jaune figure la couverture d'un capteur calculée par lancer de rayons à partir d'une description plane des bâtiments de la ville de Brest [18].

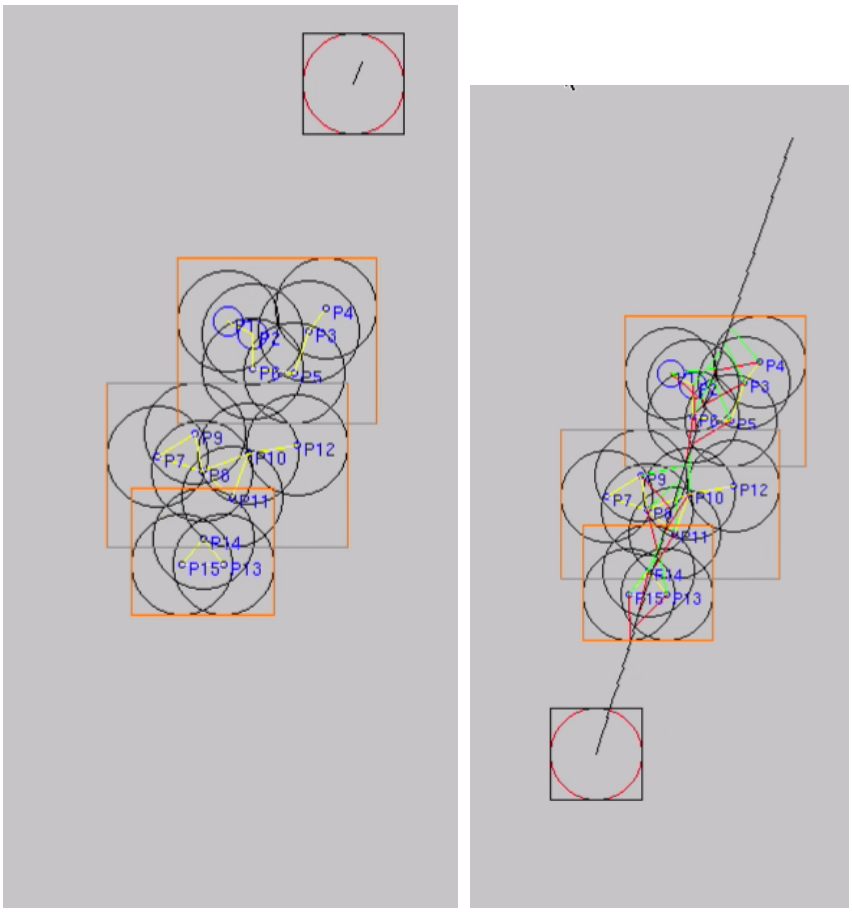


FIGURE 5.4 – Simulation concurrente d'algorithmes démarrés au vol par un mobile. À gauche, au début de la simulation, à droite, à la fin. La figure représente trois champs de capteurs disjoints sur lesquels le satellite a demandé un calcul de boîte englobante (délimités par les rectangles orange). Les résultats ont été affichés au retour des algorithmes. La boîte propre au satellite se trouve en bas de la trajectoire.

Cette partie est documentée dans le chapitre 4, elle est réalisée par des modifications opérées sur la description de réseaux en mémoire partagée de l'accélérateur.

Les descriptions d'algorithmes sont rendues indépendantes des topologies aux étapes 3 et 4 qui fournissent des organisations de processus en Occam et CUDA. Nous allons maintenant (section 5.3) préciser comment l'étape 6 procède, en admettant que les appels de ces algorithmes seront effectués aux étapes 7 et 8, par le mobile appelant un protocole chez son, ou ses, partenaires au sol.

Ce protocole intègre une interrogation du partenaire afin d'identifier le réseau, qui se sera auparavant identifié grâce à son numéro de *leader*. L'auto-configuration des réseaux est supposée accomplie, de sorte que l'on dispose d'un *leader* estampillant le réseau et du diamètre du réseau.

Le numéro du leader permet au mobile de ne pas démarrer d'activité de calcul si lors d'un contact établi avec un réseau, on constate que le traitement a déjà été effectué : dans le cas de la figure 5.4, seulement 3 calculs distribués sont effectivement démarrés, alors que les points de contact qui séquencent la simulation sont au nombre de 15.

5.3 Trois algorithmes de collecte pour les satellites LEO

Un scénario de fonctionnement d'un réseau de capteurs sans fil est donc d'utiliser un satellite pour collecter les données et établir une liaison entre plusieurs champs de capteurs distincts et des stations d'observation.

5.3.1 Évolutions relatives du mobile et du réseau

Lorsque le satellite parvient au dessus d'un champ de capteurs intéressant, une communication s'établit avec un nœud, ou une passerelle, avec lequel il partage un protocole. Ce nœud va contrôler le déroulement d'un calcul distribué sur le réseau auquel il participe. Trois cas de figures permettent de fixer des comportements couplant le mouvement du mobile et le déroulement de leur coopération :

- Dans le premier cas, la date d'arrivée du satellite est connue. Le calcul distribué est exécuté en avance, ainsi les données sont transmises au satellite dès le premier contact avec le réseau de capteurs au sol. On considère que ce calcul est effectué de manière statique, ou bien que le résultat provient de la précédente visite du satellite.
- Dans le deuxième cas, le calcul est dynamique : dès le premier contact avec le réseau de capteurs, le satellite émet une commande vers le nœud passerelle. Ce dernier propage l'information à tous les membres du réseau pour qu'ils démarrent le calcul distribué dynamiquement. Le résultat est transmis vers le premier nœud, puis vers le satellite. Le nombre de pas d'exécution est d'au moins deux fois la valeur du diamètre du réseau. Le satellite doit toujours être en contact avec le premier nœud jusqu'à ce qu'il reçoive le résultat, ce qui fait de sa vitesse de déplacement la principale contrainte.
- Dans le troisième cas, le satellite émet une commande à un premier nœud passerelle, qui est propagée à travers le réseau. Les résultats sont transmis à un deuxième nœud passerelle, diamétralement opposé au premier.

Les algorithmes sont codés en Occam, qui concernent uniquement la partie champs de capteurs après le contact satellite. Le mobile n'est pas simulé dans cette situation, mais on compte le nombre de périodes synchrones nécessaires pour terminer le calcul, en relation avec une estimation de son déplacement. Plusieurs jeux d'essais sont produits ; ce sont des réseaux quelconques, générés

aléatoirement sur des zones géographiques, et de tailles différentes : de 10 à 500 nœuds. L'exécution est tracée afin de vérifier si le fonctionnement est correct.

5.3.2 Un problème générique : le calcul de boîte englobante

On se donne à calculer la boîte englobante de chaque réseau rencontré par le mobile, ce calcul permettant de figurer un calcul distribué générique et permettant de représenter graphiquement une caractéristique géométrique proche de la couverture.

On appelle boîte englobante, le plus petit rectangle dans lequel se trouve les représentations des portées de tous les nœuds d'un même réseau.

La portée est ici figurée par un cercle dont le rayon est connu du nœud (il est fixé préalablement lors de la construction du réseau).

5.4 Algorithme *données anticipées*

Ce premier algorithme suppose que l'arrivée imminente du mobile est connue. Avant cette arrivée, on opère une diffusion de données de proche en proche jusqu'à concurrence du diamètre du réseau. Le calcul à réaliser est un calcul collectif de la boîte englobant le réseau (BB) menée par unions successives de rectangles tenant compte de la position et de la portée locale de chaque capteur.

5.4.1 Description informelle de l'algorithme

Initialement. Chaque nœud détermine sa boîte englobante initiale, à partir de sa position sur la carte et de la portée de son émetteur radio.

Transitions. À chaque itération de l'algorithme, tous les nœuds communiquent leur boîte englobante à leurs voisins immédiats. Ils calculent leur nouvelle valeur, à partir des données reçues.

$\text{BB}_{\text{locale}}(t+1) = \text{Union}(\text{BB}_{\text{locale}}(t) + \text{BB}_{\text{voisin1}}(t) + ... + \text{BB}_{\text{voisin n}}(t))$

Conclusion. Au bout d'un nombre d'itérations égal à la valeur du diamètre du réseau, toute l'information a été propagée à tous les membres du réseau, si bien que chaque nœud connaît la boîte englobante du réseau.

5.4.2 Protocole de diffusion des messages

Les messages sont diffusés aux voisins, sans adressage à un nœud particulier. Chaque message contient le statut courant du nœud, en l'occurrence la boîte englobante. Au démarrage, le nœud est initialisé avec sa propre boîte englobante.

```
1 Initialization
2         At t=0
3                 AllNodes.initialize
```

Lorsqu'un nœud reçoit un message de son voisin, il met à jour sa valeur par union avec les données reçues.

```
1 UpdateTransaction
2         while (i < Node.neigbors)
3                 Set(outMessages[i]) -- receive the message from neighbors
4                 Set(inMessages[i]) -- send the message to neighbors
5         Node.updateResult(Node.status, inMessage[i].status)
```

Le nombre de tours nécessaires est égal à la valeur du diamètre du réseau.

```
1         Initialization
2         while (i < diamter)
3                 Node[i].UpdateTransaction
```

Dans cet algorithme, le mobile ne déclenche pas de calcul en contactant le réseau, il interroge un des nœuds et récupère la valeur au tour suivant. Ce délai est fixe et ne dépend pas de la taille du réseau.

5.5 Algorithme *Transaction du mobile*

Cette fois, le mobile va déclencher le calcul au sol en appelant un protocole sur le premier nœud qu'il rencontre dans le réseau. Ce nœud agit comme un passerelle de communication, il se charge de propager la requête et collecter les résultats. Il transmet ensuite la réponse au mobile. On suppose que le mobile reste en communication avec la passerelle jusqu'à la fin du calcul.

5.5.1 Description informelle de l'algorithme

Un arbre de parcours en largeur (BFS) est construit, il propage une requête de calcul depuis la passerelle satellite (la racine) vers les nœuds feuilles. Les calculs de BB sont opérés à la remontée.

Étape descendante. À l'état initial, les nœuds restent en attente de réception de la requête. Un nœud qui reçoit une requête en mémorise la provenance et note son parent. Au tour suivant, il notifie celui-ci par un message spécifique. À la réception de ce message, les nœuds notent l'identité d'un fils dans une table. Avec la notification au parent, des requêtes de calcul sont simultanément transmises à tous les autres voisins. Après un nombre de tours égal au diamètre du réseau, un arbre a été construit.

Étape montante. La deuxième partie de l'algorithme consiste à propager les résultats depuis les feuilles jusqu'à la racine. Les nœuds se placent en attente de réception des réponses de leurs fils. Les nœuds qui n'ont pas de fils transmettent leur résultat (leur propre boite englobante) à leur nœud père. À chaque fois qu'un message d'un fils est reçu, le nœud met à jour sa valeur courante. Une fois que les réponses de tous les fils sont reçues, le nœud émet le résultat à son nœud parent. Après un nombre de tours égal au diamètre du réseau, le nœud racine a connaissance du résultat global.

Le nombre de tours total pour terminer le calcul est $diamètre \times 2 + 1$.

5.5.2 Réalisation en Occam

Protocole de diffusion des messages

L'algorithme utilise quatre messages : {*null, requête, parent, résultat*}. Le protocole de communication utilisé dans la réalisation Occam associe des données à ces messages :

```
1 PROTOCOL trans.proto
2   CASE
3     BBoxRequest; BYTE -- tag for request command with dummy data
4     BBoxParent ; BYTE -- tag for parent answer with dummy data
5     BBoxResult ; BoundingBoxArea -- tag for result with a rectangle
6     SendNull   ; BYTE -- dummy data
```

Définition des états

L'algorithme est décrit sous forme d'un automate à états. À chaque état correspond une tâche à réaliser par le nœud. L'algorithme est divisé en deux parties, l'étape descendante et l'étape montante exécutées à la suite.

Étape descendante. Au cours de l'exécution, chaque nœud se trouve successivement dans l'un de ces trois états suivants :

1. **StateInit** : attente de réception de la requête. Le nœud est à l'écoute de ses voisins et attend de recevoir le signal `BBoxRequest`. Une fois reçu, il mémorise la provenance du message et passe à l'état suivant.
2. **StateSendRequest** : propagation de la requête. Le nœud émet un accusé de réception `BBoxParent` en retour à son parent pour s'enregistrer. Il émet le signal `BBoxRequest` à tous ses autres voisins. Il passe à l'état suivant.
3. **StateReceiveParent** : enregistrement des nœuds fils. Durant un seul tour, le nœud réceptionne les messages `BBoxParent` provenant des fils. Il les enregistre dans sa mémoire locale. Il passe ensuite à l'état final.
4. **Fin de l'algorithme** : Le nœud a terminé l'exécution. Il ne fait aucune action et reste dans cet état.

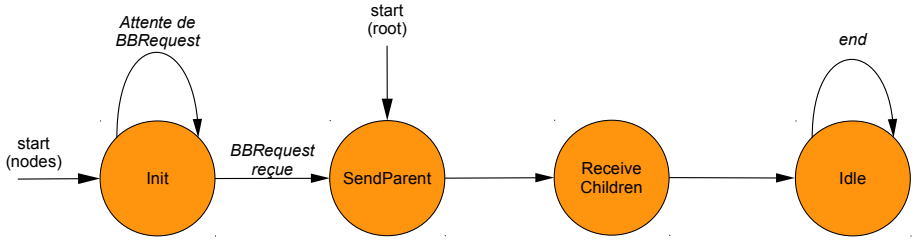


FIGURE 5.5 – Automate à états pour la propagation descendante

La mémoire est organisée ainsi :

- `currentState` contient la valeur de l'état.
- Les fils sont mémorisés dans une collection
- Le parent est mémorisé dans `indexParent`

```
1 DATA TYPE OrderedCollection
2   RECORD
3     INT NbChildren: -- reflect number of children, initially 0
4     [MaxFanOut] INT tabOfChildren: -- index of channels toward children
5 :
6 INT indexParent: -- index toward parent, initially -1
7 OrderedCollection TabOfChildren: -- initially empty
```

Étape montante. Voici la succession des états dans lesquels se trouvent les nœuds :

1. **ReceiveResult** : à l'état initial, tous les nœuds sont en attente de réception des réponses de leurs fils. Chaque réponse est traitée pour obtenir un nouveau résultat. Lorsque tous les fils ont répondu, le nœud passe à l'état suivant. Si le nœud est une feuille, il passe directement à l'état suivant.
2. **SendResult** : Le nœud transmet son résultat à son parent.
3. **End** : Le nœud a fini l'exécution, il reste dans cet état sans faire d'action.

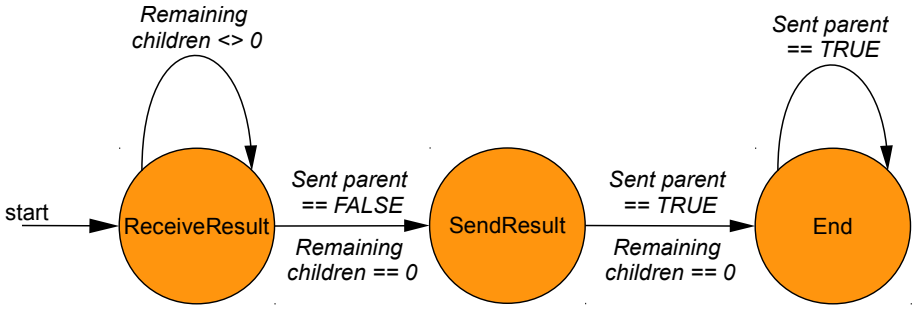


FIGURE 5.6 – Automate pour la propagation montante des résultats, depuis les feuilles jusqu'à la racine.

Détails du code Occam

Étape descendante L'automate émet des messages à chaque tour

```
StateInit :
  IF Request.received.somewhere
      Keep (indexParent)
      Set (BufOut[all], BBRequest)
      Set (BufOut[indexParent], BBoxParent)
      nextState := StateSendRequest
  ELSE
      Set (BufOut[all], SendNull)
```

```
StateSendRequest :
  Set(BufOut[all], SendNull)
  nextState := StateReceiveParent
```

Les fils sont enregistrés dans un tableau *TabOfChildren*.

```
StateReceiveParent :
  IF BufIn[i] = Parent
    AddChildren (TabOfChildren, i) -- add children to parent's collection
    TabOfChildren.NbChildren := TabOfChildren.NbChildren + 1 -- increase children
        counter
  ELSE
    Set(BufOut[all], SendNull)
    nextState := StateReceiveParent
```

```
StateEnd :
  Set (BufOut[all], SendNull)
  nextState := StateEnd
```

Étape montante L'automate de l'étape montante est présenté figure 5.6.

Le nœud se place en attente de réception jusqu'à ce que tous ses fils lui aient envoyé leur résultat. À chaque réception, il met à jour sa valeur en réalisant l'union du résultat reçu.

```
1  StateReceiveResult :
2  IF Result.received.somewhere
3    MergeResult (CurrentResult, BufIn[indexReceived]) -- update from local
         contributions
4    remainingChildren := remainingChildren - 1  -- decrease children counter
5  IF remainingChildren == 0 THEN
6    Set (BufOut[indexParent], BBResult) -- send upward node's final result
7    nextState := StateSendResult
8  ELSE
9    Set(BufOut[all],SendNull) -- otherwise, continue to wait
```

Le nœud retourne à l'état inactif une fois le résultat transmis à son parent. Le nœud *Root*, quand à lui, transmet le résultat au mobile.

```
1  StateSendResult :
2    Set(BufOut[all],SendNull)
3    nextState := StateIdle
```

```
1  StateIdle :
2    Set(BufOut[all],SendNull)
3    nextState := StateIdle
```

La figure 5.7 présente un exemple des communications échangées par un réseau de 50 nœuds, produit de manière aléatoire avec les outils NetGen.

5.6 Algorithme *Transaction et Flot vers une source*

Dans les deux algorithmes précédents, on n'a pas tenu compte du déplacement du mobile à travers le réseau. Or, dans le cas d'un satellite, la vitesse de déplacement n'est pas négligeable et le premier nœud contacté peut être hors de portée une fois le calcul terminé. Une troisième approche est donc de transmettre les résultats du calcul à un second nœud, à l'autre extrémité du réseau, que le mobile contacte en dernier avant de quitter le champ de capteurs.

Dans cet algorithme, ce sont deux arbres de recouvrement qui sont construits. Le premier arbre propage la requête depuis le nœud contacté par le mobile (la racine), jusqu'aux feuilles ; le deuxième arbre remonte les résultats vers une seconde racine.

5.6.1 Description informelle de l'algorithme

Un arbre de parcours en largeur est créé au préalable, connaissant le nœud d'entrée du mobile dans le champ de capteurs et le nœud de sortie. Chaque nœud possède donc un parent et une collection de fils pour chacun des deux arbres. Selon la topologie du réseau, les parents ne sont pas nécessairement distincts. Les nœuds transmettent la réponse dès que possible : s'ils n'ont pas de fils dans le deuxième arbre, ou dès que tous les fils ont envoyé leur résultat. Ainsi, le nombre de tours est inférieur ou égal à $2 \times$ *diamètre*.

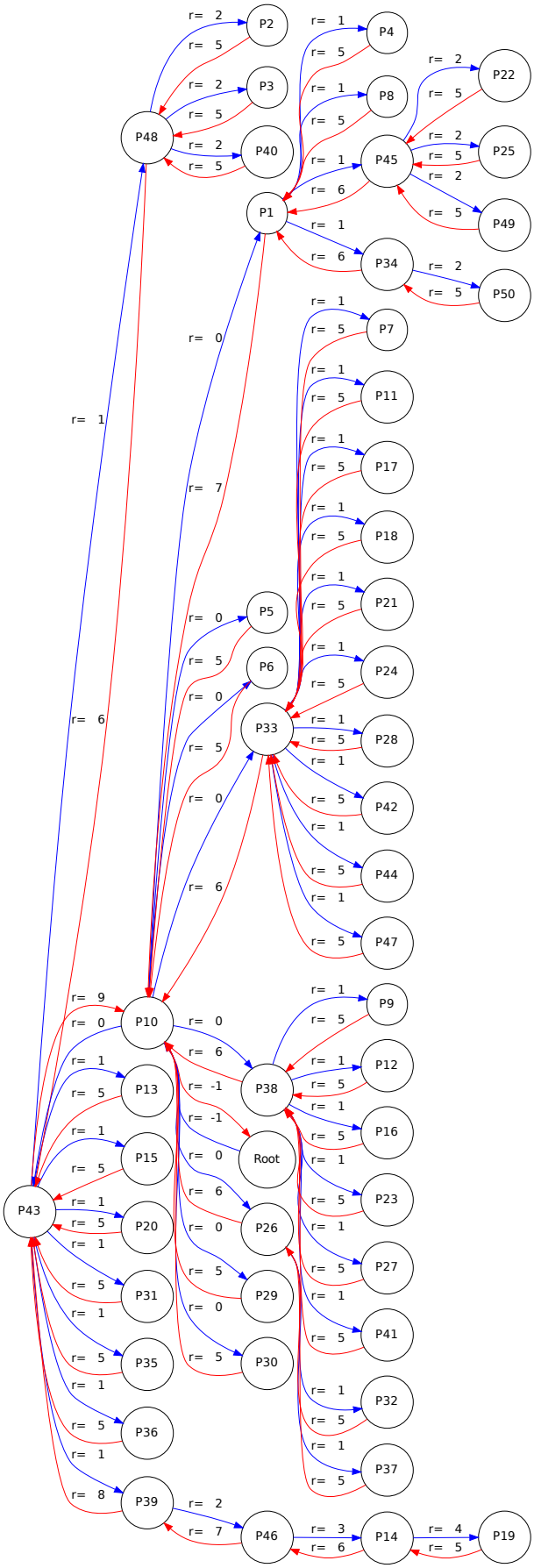


FIGURE 5.7 – Algorithme transaction, avec 50 nœuds. Les arcs en bleu montrent la propagation de la requête, ceux en rouge indiquent les résultats transmis. Le tour lors duquel se fait la transmission est numéroté.

5.6.2 Réalisation en Occam

Protocole de diffusion des messages

Les nœuds sont en attente de deux messages :

- `BBoxRequest` : démarrage du calcul sur le nœud, propagation aux voisins ;
- `BBoxResult` : réception des résultats des voisins.

L'algorithme utilise les messages suivants : *{requête, résultat, null}*. Le protocole de données utilisé dans la réalisation Occam les associe aux données suivantes :

```
1 PROTOCOL trans.proto
2   CASE
3     BBoxRequest; BYTE            -- tag for request command with dummy data
4     BBoxResult ; BoundingBoxArea -- tag for result with a rectangle
5     SendNull   ; BYTE            -- dummy data
```

Définition des états

Cet algorithme est décrit en utilisant également une machine à états. L'automate est représenté dans la figure 5.8. On suppose que les arbres de recouvrement ont déjà été construits.

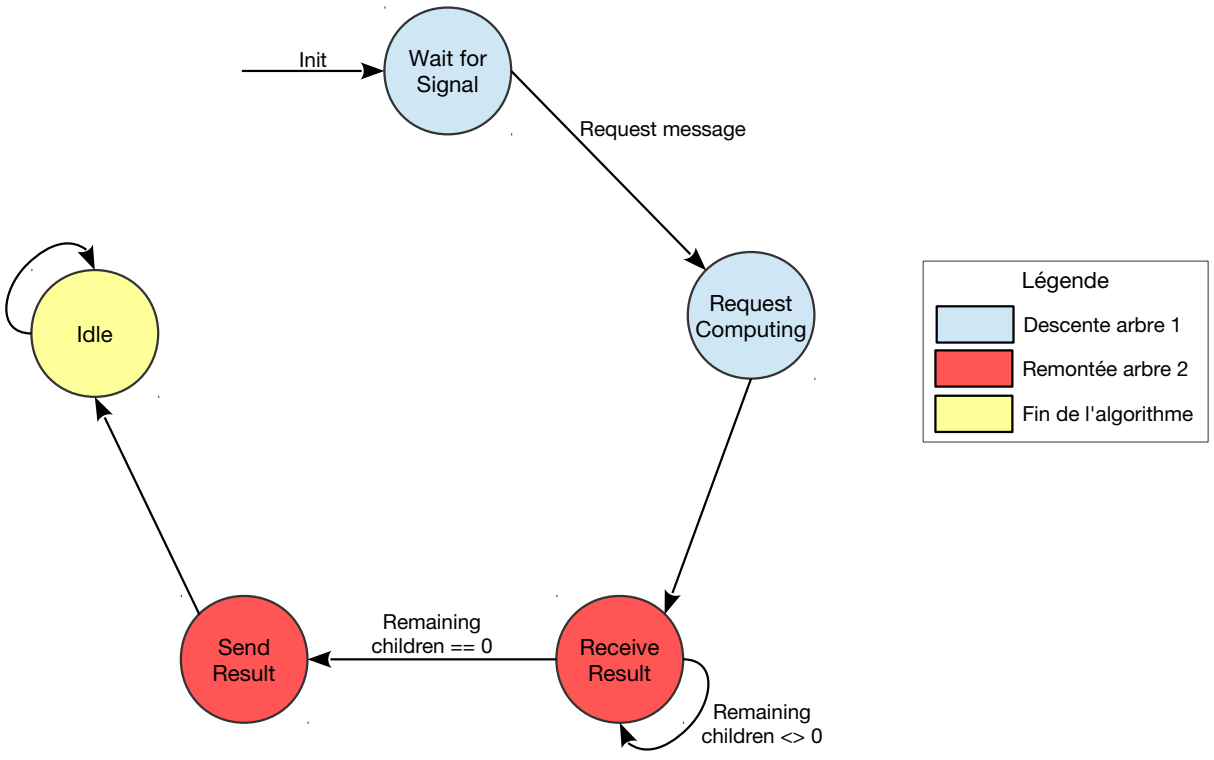


FIGURE 5.8 – Représentation de l'automate de propagation montante

1. **WaitForSignal**. Le nœud est en attente de la requête. Il peut recevoir les réponses des autres nœuds (les fils du 2nd arbre). Lorsqu'il reçoit une requête, il passe à l'état suivant.
2. **StateRequestComputing**. Le nœud propage la requête à ses fils (1er arbre). Il peut toujours recevoir des réponses des autres nœuds. Il passe ensuite à l'état suivant.

3. **StateReceiveResults**. Le nœud attend la réponse de chacun des fils du 2nd arbre. Une fois que tous les fils ont répondu, il passe à l'état suivant.
4. **StateSendResult**. Le nœud transmet sa réponse au parent du 2nd arbre. Ensuite, il passe à l'état suivant.
5. **StateIdle**. Le nœud a terminé l'algorithme, il reste dans cet état sans réaliser d'action.

Les variables utilisées sont similaires à l'algorithme précédent : une deuxième collection sert à construire le deuxième arbre.

```
DATA TYPE OrderedCollection
  RECORD
    INT NbChildren:                    -- reflect number of children, initially 0
    [MaxFanOut] INT tabOfChildren:     -- index of channels toward children
:
OrderedCollection TabOfChildrenTree1: -- initially empty
OrderedCollection TabOfChildrenTree2: -- initially empty
INT indexParentTree2:                  -- index toward parent of BFS2
INT remainingChildrenTree2:            -- number of children we are waiting for
BoundingBoxArea currentBoundingBox:    -- bounding box to be updated
```

Détail du code Occam

À chaque tour, le nœud communique avec ses voisins en émettant et recevant des messages. Ensuite, le traitement des messages reçus dépend de l'état dans lequel se trouve le nœud.

L'état `StateWaitForSignal` accepte les messages `BBoxRequest` et `BBoxResult`. Si une requête est reçue, elle est propagée à tous les fils. Si le nœud n'a pas de fils, il émet le résultat à son parent.

```
StateWaitForSignal :
  IF Request.received.somewhere
    IF (rootFirst[Children][NbFils] = 0) AND (rootSecond[Children][NbFils] = 0)
      -- a leaf node for both tree
      nextState := SendBBResult -- send back the result next round
      BufOut[rootSecond[indexParent]] := SendResult
    ELSE
      -- propagation de la requete
      Set (BuffOut[allChildrenOfTree1], Request)
      nextState := StateRequestComputing
  ELSE
    Set(BufOut[all], NULL)
    nextState := StateWaitForSignal
```

Le nœud reste dans cet état pendant un seul tour.

```
StateRequestComputing :
IF Result.received.somewhere THEN
        MergeResult (CurrentResult, BufIn[indexReceived])
        remainingChildrenTree2 := remainingChildrenTree2 - 1
IF remainingChildren == 0 THEN
        Set (BufOut[indexParentOfTree2], SendResult)
        nextState := StateSendResult
        Set(BufOut[all], NULL)
        nextState := StateReceiveResult
```

Le nœud accepte les résultats de ses fils : tant qu'il reste des résultats à recevoir, le nœud reste dans cet état. Ensuite, il propage son résultat à son parent.

```
StateReceiveResult :
IF Result.received.somewhere THEN
  MergeResult (CurrentResult, BufIn[indexReceived])
```

```
4     remainingChildrenTree2 := remainingChildrenTree2 - 1
5   IF remainingChildren == 0 THEN
6     Set (BufOut[indexParentOfTree2], SendResult)
7     nextState := StateSendResult
8   ELSE
9     Set(BufOut[all],NULL)
```

Pendant un tour, le nœud propage les résultats à son parent.

```
1   StateSendResult :
2     Set(BufOut[all],NULL)
3     nextState := StateIdle
```

L'algorithme est terminé, le nœud reste inactif.

```
1   StateIdle :
2     Set(BufOut[all],NULL)
3     nextState := StateIdle
```

Un réseau de 50 nœuds placés aléatoirement sur une carte montre la circulation des données entre le premier point de contact du satellite et le dernier, lors de l'exécution de l'algorithme. Le graphique produit (figure 5.9) indique le numéro du tour où a lieu la communication.

5.7 Évaluation des performances des algorithmes, conclusion

Notre objectif a été de faire fonctionner trois propositions d'algorithmes dans une simulation de déploiement. Ce chapitre a décrit une méthode de travail permettant de mettre au point plusieurs algorithmes à même de planifier des visites par mobile afin d'effectuer des collectes ou des opérations de contrôle sur des champs de capteurs. Cette étude a été effectuée principalement en Occam, mais le travail de traduction pour passer du code Occam au code CUDA a partiellement été effectué.

Un premier résultat est la mise en évidence de métriques de performances, en particulier le nombre d'étapes pour réaliser un calcul, et le nombre de messages échangés lors d'une visite.

Un autre résultat est la possibilité d'évaluer les délais de réaction, la vitesse de fonctionnement du réseau, la consommation d'énergie, et cela en regard des algorithmes proposés. Il est donc possible de choisir l'algorithme adapté à la configuration du réseau, en terme d'algorithme le plus rapide tout en consommant le moins d'énergie.

Ce travail a été publié à la conférence Wisats en 2015 [7].

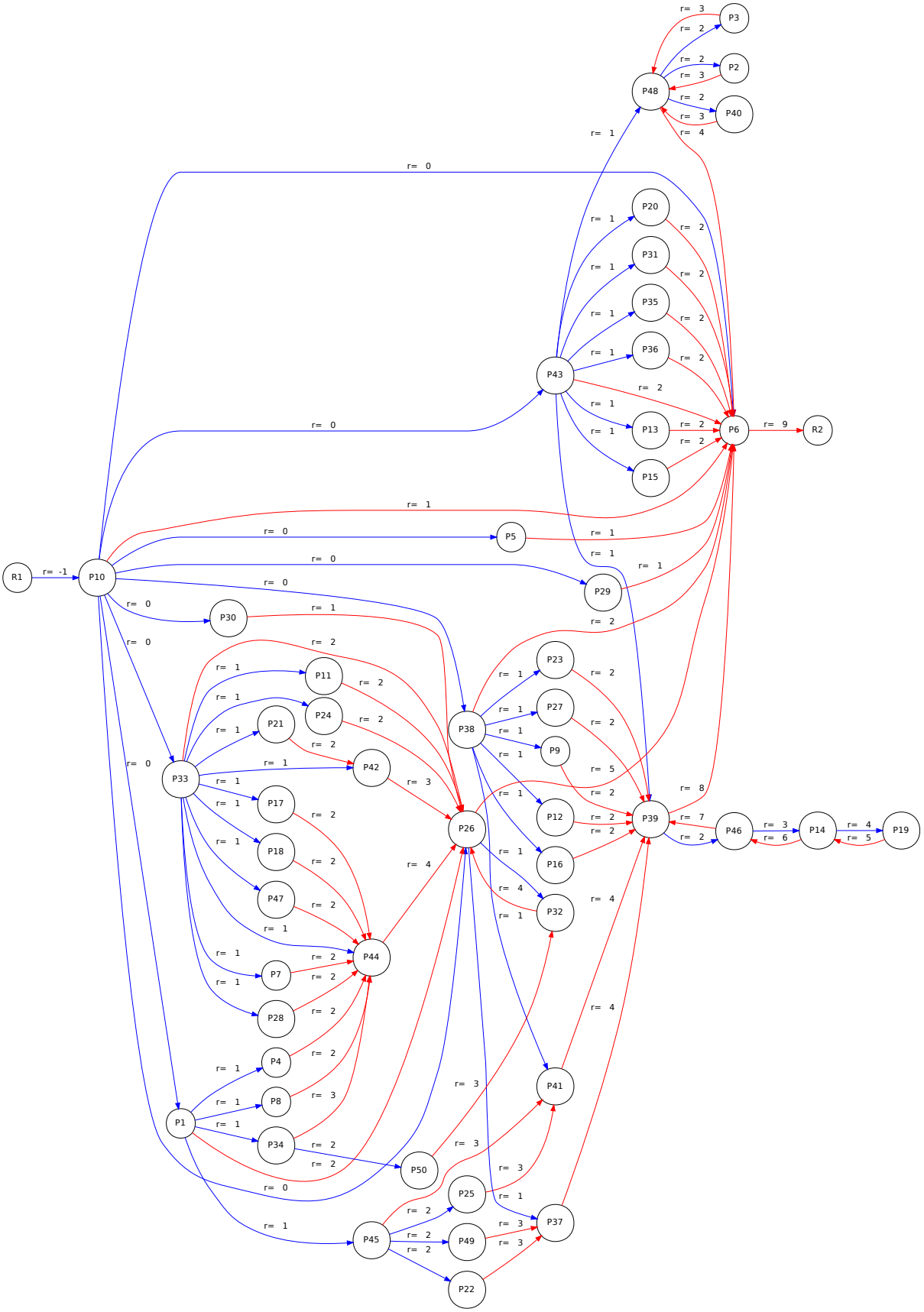


FIGURE 5.9 – Algorithme flot montrant la circulation des commandes (bleu) et résultats (rouge) avec une circulation de la gauche (R1, contact du satellite) vers la droite (R2, décrochage du satellite).

6
Virtualisation et réseaux de capteurs

6.1 Motivations

Nous avons présenté et justifié l'expression de traitements distribués en terme de processus communicants reproduisant le modèle synchrone (chapitre 5). Une illustration en est l'usage du langage Occam, pour représenter spatialement des systèmes de capteurs opérant en parallèle et coopérant par radio.

L'approche NetGen de la conception des réseaux de capteurs, et celle de PickCell pour son adaptation aux contextes physiques, constituent des abstractions pour deux réalités. La réalisation d'outils informatiques procède en général à partir de modèles simplifiés observés représentant des organisations de données et des opérations d'analyse ou de transformation qu'on leur applique. Le modèle interne de NetGen est ainsi un simple graphe de processus menant des activités disjointes, mais synchronisés lors des échanges. Nous avons vu que CSP recouvre ce modèle en étant plus général, et qu'il était possible de forcer des générations de code SIMD à partir des organisations du modèle, et des barrières induites par les communications radio.

En ce sens, NetGen, QuickMap, et les autres outils interfaçant le modèle central, réalisent une virtualisation du réseau de capteurs qu'ils sont capables de représenter de manière générique et générale, l'organisation du réseau, le calcul des portées et des puissances dépensées facilitant la conception des applications, éliminant beaucoup de spéculations.

Dans ce chapitre, nous allons revenir sur Occam, utilisé cette fois dans son rôle local, mais avec la nécessité d'intégrer la totalité des activités locales, y compris les échanges radio avec les partenaires.

En avant-propos à cette visite, deux constats peuvent être effectués :

- l'exécution d'Occam a été virtualisée de manière précoce. Ce langage a en effet été développé sur des simulateurs de niveau architecture avant les implémentations matérielles. Il s'agit ici d'une méthode de travail connue, le simulateur au niveau jeu d'instructions « *Instruction Set* » (IS) précédant toujours la réalisation d'une vraie machine.
  On trouve par exemple des simulateurs IS simples, tels que P4 [44], qui ont évolué et se sont enrichis. Transputer Virtual Machine (TVM) est ainsi une machine virtuelle portable compatible avec la chaîne de développement KRoC et plusieurs micro-contrôleurs.

Des implémentations matérielles ont pris le relais, le Transputer[45] de Inmos et David May en étant la plus connue. Des variantes dans la famille d'Occam sont des transformations syntaxiques pour l'exécution sur FPGA (Handel-C) et pour les multi-cœurs (XMOS et XC).
- la portabilité sur les micro-contrôleurs (MCU) est un réel problème pour les applications. Ces circuits sont en évolution permanente, avec des améliorations fonctionnelles portant sur les interfaces, l'architecture, l'état de sommeil, la mémoire. Les jeux d'instructions sont différents et évolutifs, les niveaux des systèmes d'exploitation ne sont pas standardisés. Dans le fond, ces MCUs sont cependant très similaires et reproduisent le même modèle matériel.
  Il y a deux réponses connues à cette situation dont l'une est de virtualiser la spécification et l'exécution, en prévoyant des travaux de portages et d'adaptations qui peuvent être menés très efficacement en environnement industriel. Les utilitaires utilisés pour exécuter MacOSX, Linux, ou Windows sont une illustration de cette approche. La seconde alternative est d'observer que les concentrations industrielles font le ménage, et là encore, on peut se référer aux trois systèmes de PC, ou aux deux systèmes de mobiles.

Les difficultés potentielles de machines virtuelles pour les systèmes embarqués sont les pertes possibles en performance et en consommation d'énergie, à comparer avec des solutions standard ou évoluées en terme de programmation.

Dans ce chapitre nous présentons une adaptation de TVM pour les réseaux de capteurs sans fil. Cette adaptation est basée sur la réalisation de primitives de communication radio supportées par des circuits adaptés aux protocoles de réseaux maillés, tels que Digimesh ou 802.15.4, ou pour des réseaux cellulaires tels que LoRa abordés dans [17].

Les travaux menés sur le flot de spécification (chapitre 2) présument que l'architecture des nœuds permet de faire supporter la phase de communication par des processus spécifiques implantés sur les modules radio. Nous décrivons l'organisation d'exécution abstraite de TVM, le principe d'extension de cette machine virtuelle avec une primitive radio et l'exécution concurrente du modèle synchrone dans ce contexte.

6.2 Exécution d'Occam

Le chapitre va en premier lieu (section 6.2.1) présenter un état de l'art sur les machines virtuelles pour l'embarqué en particulier *Transputer Virtual Machine* (TVM)[29] qui, agrégée à des programmes Occam compilés en code intermédiaire, permet d'exécuter le cycle synchrone du réseau de capteurs. TVM est ici complétée par des primitives de communication sans fil assurant l'exécution distribuée (section 6.5). Des exemples de programmes concurrents pour une plate-forme Arduino sont proposés pour illustrer la cohésion de l'acquisition locale et des contributions distribuées. Les questions qui émanent de l'usage des machines virtuelles sont l'efficacité de l'exécution interprétée, la proportion de code natif et de code interprété et la consommation électrique, pour les réseaux de capteurs dont la survie dépend de cette caractéristique.

6.2.1 Flot : des programmes à l'interprétation

On a vu que deux possibilités existent pour exécuter les programmes Occam :
- le traducteur tranx86, qui produit après compilation, du code exécutable sur processeur x86 [46],
- l'usage d'une machine virtuelle telle que TVM qui va exécuter le programme en interprétant ses instructions. L'usage de cette machine pour les systèmes embarqués a été présentée par

C.L. Jacobsen et M.C. Jadud dans [39].

Le flot de compilation et de chargement pour l'interprétation est présenté en figure 6.1. On reconnaît, à droite, la préparation d'un fichier hexadécimal intégrant les instructions, et destiné à l'exécution sur la machine virtuelle. La partie gauche du flot est la préparation de cette machine virtuelle, qui en association avec le membre droit, représente un programme Occam complet exécutable sur la machine cible (ici Arduino).

L'approche TVM permet la portabilité du code sur plusieurs matériels supports de capteurs courants, nous avons utilisé la branche pour le processeur AtMega que l'on trouve sur les cartes Arduino.

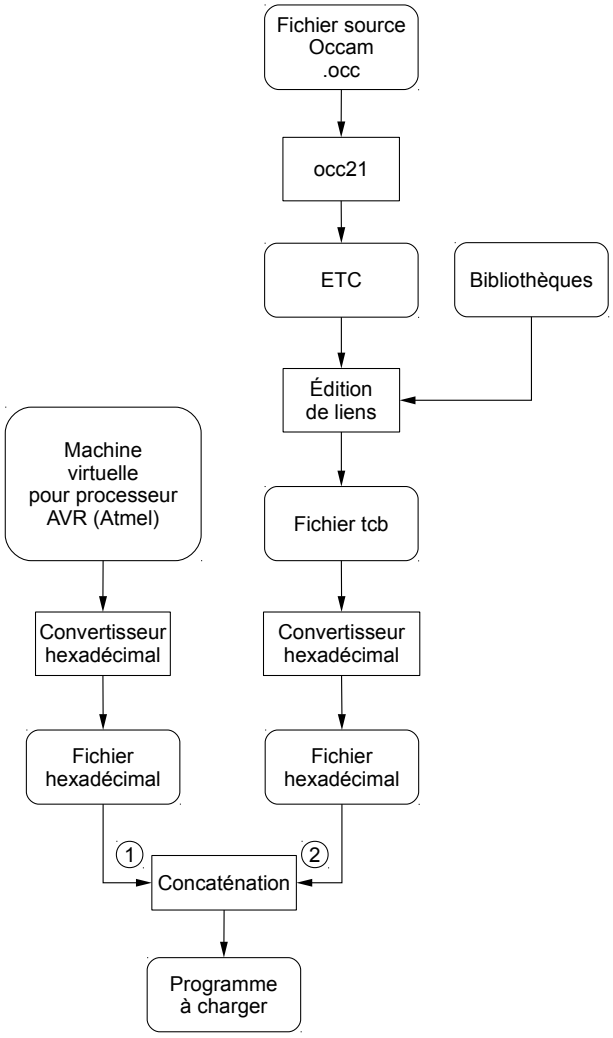


FIGURE 6.1 – Flot de compilation d'un programme Occam en vue de son exécution sur TVM. À gauche la machine virtuelle, à droite l'exécutable.

6.2.2 Structure de la machine virtuelle

La machine virtuelle est présentée figure 6.2. On reconnaît à gauche les services fixes et notamment l'interpréteur présenté sous la forme d'une librairie stable et portable. La partie centrale est constituée des composants logiciels assurant l'adaptation à une cible particulière. Elle inclut

par exemple l'implantation mémoire du code et des données, les périphériques d'acquisition et de contrôle, et les timers.

La partie droite représente les extensions fonctionnelles apportées à la machine virtuelle. Dans notre cas, l'accès au canal radio sera implanté par des routines écrites en C, implantées dans une table standard de primitives et rendues visibles au programme Occam en tant que procédures normalisées en termes de noms et de paramètres.

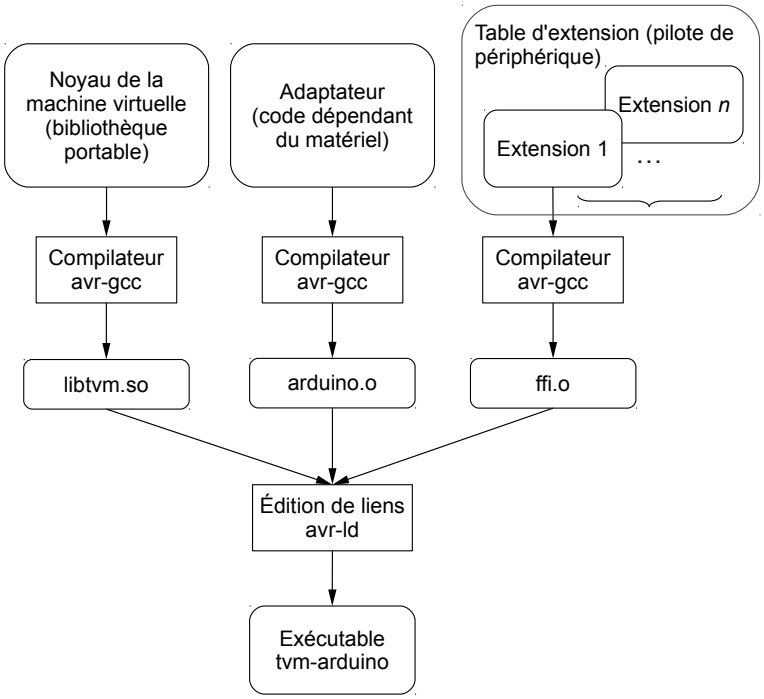


FIGURE 6.2 – Flot de compilation de la machine virtuelle Occam, avec les primitives d'extension

6.3 Aperçu du jeu d'instruction et de son exécution

La concurrence est effective ou fictive. Elle va être décrite dans le but d'éclairer les propriétés de l'exécution concurrente sur une cible telle qu'un capteur communiquant par radio, en entrelacement, avec des acquisitions, et de possibles traitements intensifs locaux.

6.3.1 Format des instructions et Bytecodes

Un des problèmes posés par les MCU est celui de la taille des mots standards. Cette question est aussi apparue avec les jeux d'instruction des microprocesseurs qui ont évolué de 8 bits sur les premiers circuits intégrés, vers actuellement 32 bits ou 64 bits.

Les MCUs ont besoin de garder toutes les possibilités pour préserver des fonctions de coûts énergétiques et économiques, et de compacité. La taille des mots est un problème à la fois pour les applications, et les outils de programmation.

Le code intermédiaire de TVM se nomme ETC (extended Transputer code) et convient pour pour une famille de processeurs de chemins de données de 8, 16, ou 32 bits. Les instructions sont présentées sous forme de bytecodes (8 bits), permettant le chargement séquentiel d'opérandes de

tailles variables [47]. Chaque bytecode se divise en 4 bits spécifiant un code opératoire, et 4 bits fournissant un opérande. Les opérandes sont donc enfilées dans un registre spécifique *operande register* (OReg), par tranches de 4 bits. L'adaptation de l'IS à l'architecture cible est donc dynamique et assurée par la machine. Le même programme binaire peut s'exécuter sur tous les chemins de données, grâce aux décodeurs d'instructions.

Le synoptique général de la machine est présenté en figure 6.4, et sera commenté ultérieurement, section 6.4.

L'accès aux constantes est donc géré via ce registre Oreg. L'instruction Pfix se divise en un code opération et en une tranche d'opérande. Pour charger la constante 6, on a besoin des deux bytecodes listés dans le Code 6.1. Ceci étant fait, la constante peut être transférée vers la pile matérielle d'évaluation à trois étages A, B, C.

Code 6.1 – Chargement constante 8bits

```
1       pfix 1
2       pfix 2
3       ; OReg vaut (1 <<4) + 2
```

Les opérations utilisent implicitement les données de la pile d'évaluation. Les opérations portent en général sur les deux premiers étages A et B. Par exemple, pour réaliser une multiplication de petite constante, on génère et on empile ces constantes puis on demande l'opération, voir le Code 6.2.

Code 6.2 – Chargement de deux petites constantes et multiplication

```
1       pfix 2
2       ldc     ; empile 2
3       pfix 3
4       ldc     ; empile 3
5       mul     ; A := A * B
```

Les instructions directement accessibles sont au nombre de 16, ce qui pourrait suffire pour des automates simples. La démultiplication du code opératoire s'effectue en dépassant la capacité des 4 bits initiaux. En effet, les instructions peuvent être encodées progressivement par la même technique que les opérandes. Le nombre de bytecodes dans les instructions peut donc être arbitrairement long ou au contraire, très court. Ceci rend le processeur particulièrement adapté à l'embarqué, avec un code très compact, mais développé dans le micro-code de processeurs taillés pour des applications spécifiques.

L'architecture intègre une mémoire embarquée, des liens de communication série couplés à des mécanismes DMA, un noyau commandé dans le micro-code des instructions. Ce noyau est complet et recoupe les fonctionnalités du langage :

- commutation de processus en temps partagé ;
- mécanisme de synchronisation ;
- horloges temps réel ;
- entrées-sorties actionnées par DMA (*Direct Memory Access*).

6.3.2 Structure de données pour la gestion des processus

La gestion des processus est réalisée par le micro-noyau, lors de l'exécution des instructions système : démarrage et terminaison des processus, synchronisation, entrées-sorties, opération sur le timer. Cette section va expliciter le fonctionnement de ces opérations eu égard au parallélisme de la spécification. Il est en premier lieu nécessaire de préciser qu'en dehors des contraintes de

synchronisation de cette spécification, les processus activables avancent en temps partagé. Les commutations sont opérées à l'occasion d'instructions laissant la pile d'évaluation vide. Il y a donc très peu de sauvegardes et de restaurations de contexte réalisées à ces occasions, dont le coût peut se résumer à des opérations sur listes chaînées menées par le micro-noyau. Les commutations de contexte sont des opérations légères menées en mémoire embarquée rapide.

Les processus disposent chacun d'un espace de travail qui sert à stocker les variables locales et les valeurs temporaires qui sont manipulées. Il correspond à un vecteur de mots consécutifs en mémoire, organisé à la manière d'une pile descendante. Ce contexte est repéré par son adresse mémoire que l'on désigne par Wptr (Workspace pointer). Les variables locales qu'il contient sont accédées de manière relative à cette référence.

Avant de créer de nouveaux processus concurrents, le Transputer mémorise dans l'espace de travail courant :

- leur priorité, stockée à l'adresse Wptr + 2 ;
- leur nombre, stocké à l'adresse Wptr + 1 ;
- l'adresse de la suite du programme (Wptr + 0), une fois que tous ont terminé.

Un espace de travail est donc alloué à chaque nouveau processus : il est dimensionné de manière à contenir les variables locales, ainsi qu'un espace supplémentaire de gestion, qui dépend du type de processus. Dans le cas d'un processus sans entrée ni sortie, elle est de 2 mots. L'instruction STARTP ajoute ce nouveau processus dans l'ordonnanceur.

L'ordonnancement des processus actifs est réalisé sur des queues de priorités par le micro-noyau du Transputer (figure 6.4). La commutation de processus est provoquée par :

- l'interruption de temps partagé déclenchée par l'horloge matérielle, toutes les microsecondes ;
- une suspension sur un délai, avec un passage à l'état inactif ;
- une communication, où l'on attend le partenaire dans un canal. Le processus est alors prêt à communiquer ;
- une alternative, construction similaire à une attente sur plusieurs canaux.

6.3.3 Barrières de synchronisation PAR

Il s'agit ici de la construction parallèle PAR, ou PAR répliquée. Plusieurs processus s'exécutent concurremment fictivement ou réellement. La construction se termine avec le dernier processus (figure 6.3).

Une adresse de la pile est réservé pour le compteur de processus. Un compteur indique le nombre de processus restants. Lorsqu'un processus termine, il décrémente ce compteur. Quand le dernier processus décrémente le compteur, un branchement est fait vers la suite du programme.

6.3.4 Communication entre processus

La synchronisation entre les processus se fait à l'occasion des communications, celles-ci étant bloquantes. L'échange de données se fait par un lien série entre Transputer, dans le cas de processus exécutés réellement simultanément, ou bien par écriture de la donnée dans l'espace de travail, puis sa lecture.

Le code 6.3 présente deux processus concurrents définis dans deux procédures P et Q, et qui peuvent communiquer sur un canal partagé n.

Code 6.3 – Construction parallèle avec communication bloquante

```
1 CHAN OF INT n:
```

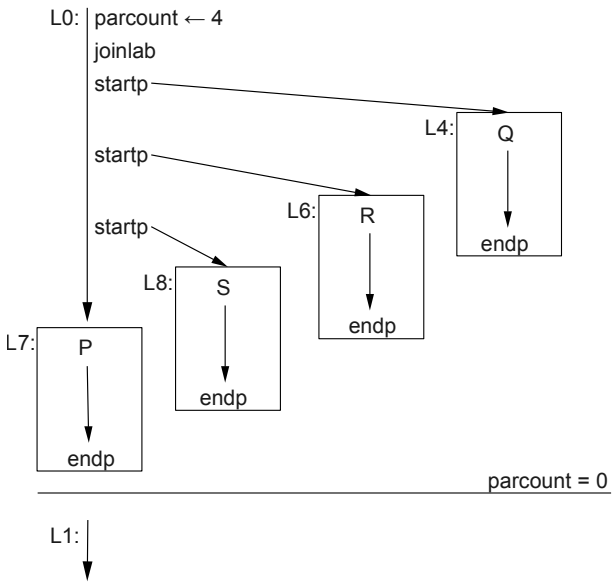


FIGURE 6.3 – Lancement de quatre processus P, Q, R, S et barrière de synchronisation. L'instruction *endp* décrémente le compteur *parcount*, initialement à 4 dans cet exemple. Lorsque le compteur devient nul, le processus sait qu'il est le dernier à terminer. Il réalise un branchement vers la suite du programme, à l'adresse stockée au préalable par l'instruction *joinlab*.

```
2  PROC P
3    SEQ
4      n ! 1
5  :
6
7  PROC Q
8    INT num:
9    SEQ
10     n ? char
11 :
12
13 PAR
14   P
15   Q
```

Le code généré pour cette opération inter-processus se résume à quelques instructions qui s'appuient sur le micro-code du processeur virtualisé (code 6.4). Ces opérations prennent en moyenne une vingtaine de cycles, engendrant très peu d'accès en lecture d'instructions.

Code 6.4 – Flot d'instructions pour la communication bloquante

```
1  L0:
2          AJW -6 ; réservation de 6 mots
3          .NOTPROCESS
4          STL 3 ; canal <- nil
5          LDLP -5
6          STARTP
7
8  L3:             ; code du processus P
9          LDC 1 ; valeur 1
10         STL 5
11         LDLP 5
12         LDLP 3 ; adresse du canal
13         LDC  4 ; un mot de 4 octets à transmettre
```

```
14          OUT     ;
15          LDLP 0
16          ENDP
17
18  L4:
19          LDLP 9
20          LDLP 8 ; adresse du canal
21          LDC 4
22          IN
23          LDLP 5
24          ENDP
25
26  L1:
27          AJW 6
28          RET
```

La figure 6.4 présente le noyau en exécution face à un processus, la pile d'évaluation et les queues de processus.

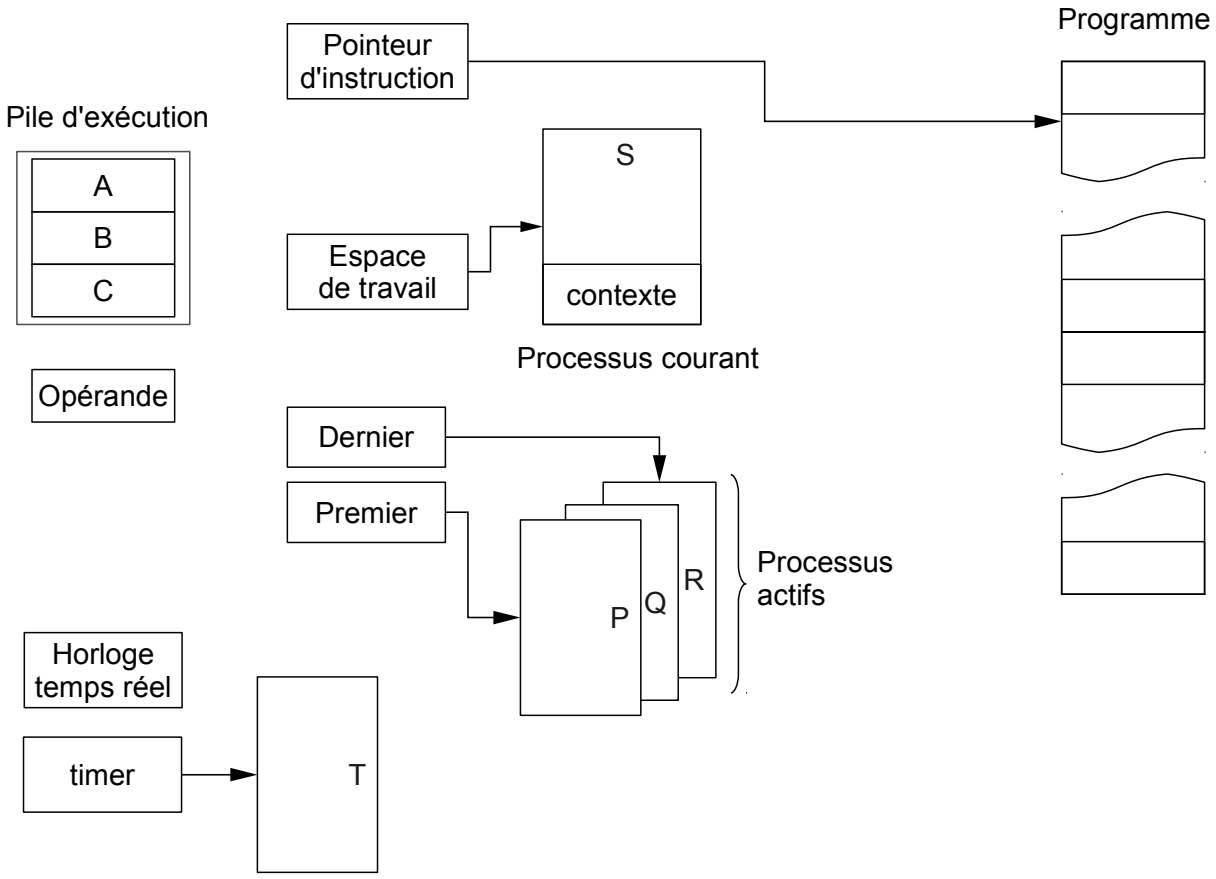


FIGURE 6.4 – Architecture interne simplifiée d'un Transputer : les processus actifs P, Q, R, S, sont ordonnancés par le noyau micro-codé, le processus T est inactif, en attente du délai indiqué par le registre *timer*.

6.4 Architecture de la machine virtuelle

La réalisation de TVM présente la particularité d'être portable, avec des déclinaisons ciblant en particulier les matériels suivants :

- les familles de processeurs compatibles x386 ;
- la famille des circuits Atmel, comportant les processeurs AVR sur les plates-formes Arduino [48] ;
- le Blackfin de Analog devices, utilisé dans le robot SVR1.
- le processeur ARM7 LPC de NXP.

Nous allons voir l'organisation de cette machine virtuelle, en utilisant comme illustration le cas de fonctionnement sur les circuits Atmel équipant la famille de cartes Arduino.

6.4.1 Modèle d'exécution

Les éléments de la structure à l'exécution sont présentés figure 6.4. Ce sont le programme, en général implanté en mémoire statique interne, ou en flash, la pile d'évaluation des processus gérée sous forme de contextes, les registres du noyau pour les files des processus rangés par priorité, les files des timers et les files des processus en entrées-sorties.

6.4.2 La structure logicielle de TVM

Le logiciel est conçu en deux composants principaux : d'une part, l'interpréteur, sous la forme d'une bibliothèque compilée pour l'architecture sous-jacente. L'interpréteur est modulaire et indépendant des architectures cibles. Il contient le noyau d'exécution, pour charger et exécuter les bytecodes, gérer les files des processus et l'ordonnancement de ces derniers, ainsi que la mémoire relative aux contextes d'exécution.

D'autre part, le *wrapper* fait la liaison entre les composants logiciels et les ressources matérielles de la machine cible, notamment l'architecture du processeur (Harvard ou Von Neumann), les ressources supportant le timer ou le gestionnaire d'interruptions. C'est ce *wrapper* qui doit être modifié lors d'un portage vers un nouveau matériel.

Une autre caractéristique intéressante de la machine virtuelle est sa capacité d'extension par l'ajout de primitives. Les primitives sont des procédures appelées à partir du programme Occam et exécutées directement sur le processeur, sans utiliser l'interpréteur. Ces procédures présentent deux intérêts :

- accélérer l'exécution du code : la surcharge due à l'interprétation des bytecodes disparaît, si bien que la vitesse d'exécution est similaire à un programme compilé nativement.
- accéder aux ressources matérielles sur la cible : le pilotage de composants périphériques, comme un émetteur-récepteur radio, se programme ainsi aisément.

6.5 Intégration d'un émetteur radio

Dans le cas de la concurrence effective et distribuée d'une exécution sur un réseau de Transputer, ceux-ci sont reliés entre eux par des liens série asynchrones contrôlé par des mécanismes DMA agissant pour le compte du noyau.

Le protocole utilisé est un protocole acquitté. Le processus émetteur transmet ses données octet par octet, et chaque octet est l'objet d'un paquet d'acquittement ACK du récepteur. Les liens sont

bidirectionnels et les émissions et réceptions peuvent être opérées en recouvrement, permettant d'atteindre des débits proches du débit nominal du lien.

On peut retrouver cette concurrence dans un réseau de capteurs, où les nœuds communiquent entre eux grâce à des émetteur-récepteurs radio, en utilisant des protocoles 802.15.4 (CSMA ou TDMA). Ici encore les phases de communication du modèle synchrone sont utiles, car elles permettent d'échanger les constructions de simulation (code 4.3), pour des rendez-vous de communications radio de proximité.

Intégrer la gestion du lien radio dans la machine virtuelle Occam est donc naturel pour le fonctionnement du système distribué sans fil.

6.5.1 Aspects techniques de réalisation

Une première réalisation a été réalisée sur plate-forme Arduino avec un module XBee, ce qui permet d'utiliser les fonctionnalités de gestion automatique du canal radio. La gestion des retransmissions automatiques lors de collisions, ou celle des super-trames TDMA est ainsi masquée et exécutée dans le module radio.

Il reste à commander les transferts de données de, et vers, ce module. Celui-ci est relié à la carte principale par un port série bidirectionnel. Il s'agit de deux primitives de communication radio écrites en C et qui fournissent un service de haut niveau. La première, `xBeeSend`, transmet les données ; la seconde, `xBeeReceive`, se place à l'écoute du port série et récupère les données dans une mémoire. Le code Occam de la phase de communication synchrone peut donc être schématisé comme en 6.5 :

Code 6.5 – Chargement constante 8bits

```
1 PROTOCOL xBeeData xBeeDataIn, xBeeDataOut:
2 PAR
3   xBeeSend    (xBeeDataOut)
4   xBeeReceive (xBeeDataIn)
```

6.5.2 Sémantique synchrone

Le code simulé contient une phase de communications synchrones où tous les processus appartenant au même réseau se donnent rendez-vous au même moment pour échanger des informations. Chaque processus recopie l'information à transmettre sur ses canaux sortants, qui le lient à ses voisins. Simultanément, il est prêt à recevoir des données sur ses canaux entrants. Le code Occam de cette phase synchrone se représente de la manière suivante :

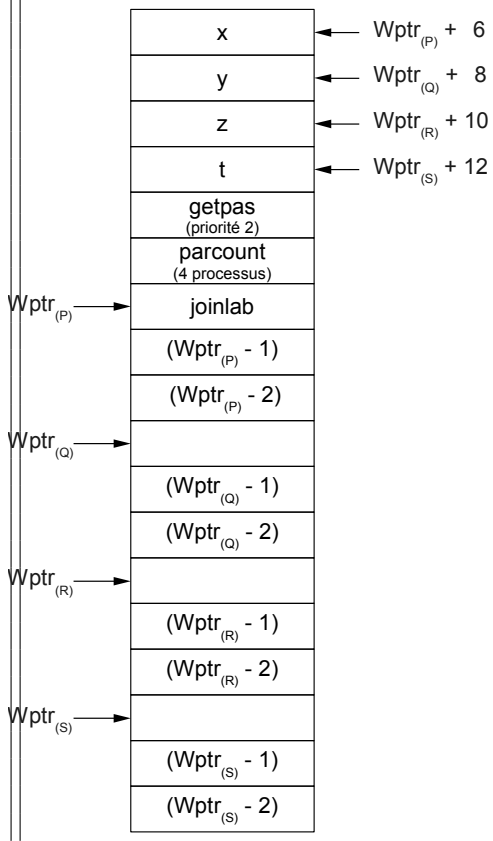
```
PAR
  PAR i FOR SIZE output_channel
    output_channel[i] ! output[i]
  PAR j FOR SIZE input_channel
    input_channel[j]  ? input[j]
```

Dans le monde physique, les communications radio atteignent simultanément tous les récepteurs à portée d'un émetteur, la simulation des communications sortantes est donc vérifiée ici. De plus, pour qu'un message soit transmis, il faut que le destinataire soit à l'écoute : émetteur et récepteur doivent donc se donner rendez-vous.

Toutefois, la réception des données est différente : si un nœud reçoit simultanément les messages de plusieurs voisins, il se produit une collision : les signaux ne peuvent être récupérés correctement.

Il est donc nécessaire d'ordonnancer les communications radio de façon à ce que deux voisins d'un même nœud n'émettent pas simultanément, en utilisant par exemple un protocole de transmission basé sur TDMA (time division multiple access).

```
1  .GLOBAL Sys
2  .SETWS 15
3  .SETVS 0
4  .SETMS 0
5  .FILENAME test4.occ
6  .LINE 1
7  .PROC Sys
8          AJW -6
9          .LINE 3
10         GETPAS
11         STL 2
12 .OCCCOMMENT INT priority affinity state
13         LDC 3    -- #00000003
14         STL 1
15 .OCCCOMMENT INT parcount
16         .LOADLABADDR L1
17 .OCCCOMMENT PTR joinlab
18         STL 0
19 .OCCCOMMENT PTR joinlab
20         .LOADLABDIFF L4 L3
21         LDLP -3
22         STARTP
23 L3:
24         .LOADLABDIFF L6 L5
25         LDLP -6
26         STARTP
27 L5:
28         .LINE 4
29         LDL 5
30         LDL 5
31         ADD
32         STL 5
33 .OCCCOMMENT tstack empty (stackdepth = 0)
34         LDLP 0
35 .OCCCOMMENT joinlab
36         ENDP
37 L4:
38         .LINE 5
39         LDL 7
40         LDL 7
41         ADD
42         STL 7
43 .OCCCOMMENT tstack empty (stackdepth = 0)
44         LDLP 3
45 .OCCCOMMENT joinlab
46         ENDP
47 L6:
48         .LINE 6
49         LDL 9
50         LDL 9
51         ADD
52         STL 9
53 .OCCCOMMENT tstack empty (stackdepth = 0)
54         LDLP 6
55 .OCCCOMMENT joinlab
56         ENDP
57 .OCCCOMMENT .MAGIC JOINLAB
58 L1:
59         .LINE 2
60         AJW 6
61         RET
62 .GLOBALEND Sys
63         .ALIGN 2
64         .END
```



6.6 Un bilan de l'intégration Occam des communications

C'est à notre connaissance la première expérimentation portant sur l'intégration des communications radio dans un environnement Occam. L'usage d'une machine virtuelle pour supporter la programmation dans un réseau de capteurs est également une nouveauté.

Les propriétés qui découlent de cette initiative sont prometteuses, avec d'abord la portabilité des applications sur des plates-formes MCU multiples, avec des interfaces entre les composants logiciels bien compris. Le travail que nous avons présenté a utilisé les services d'un port série basse vitesse pour adresser le module radio, mais des travaux plus récents ont utilisé un port SPI bien plus performant pour commander un module radio LoRa (travaux de Tuyen Truong).

L'usage des machines virtuelles est très répandu, avec des avantages de portabilité. La machine la plus courante est probablement celle qui supporte le langage Java, mais on peut aussi noter que Smalltalk bénéficie d'avantages similaires avec des plates-formes comme Visualworks ou Pharo. La contrepartie la plus souvent citée est la baisse de performance liée à l'interprétation. Cette lacune est partiellement compensée par des techniques de compilation *just in time* que l'on doit à Smalltalk.

Dans le cas des capteurs, on peut faire deux remarques :

- le poids des performances est moindre dans la mesure où beaucoup de fonctions sont supportées dans le matériel,
- la concurrence des activités réduit encore le handicap de l'interprétation. Les opérations radio sont typiquement déportées sur des modules périphériques qui peuvent travailler en recouvrement avec le MCU du capteur.
- beaucoup de fonctions sont implémentées en C, sous forme de primitives.

La voie *machine virtuelle* apparaît donc comme une voie intéressante, avec des perspectives à explorer : investigations de plusieurs MCU, et surtout construction d'un flot de synthèse facilitant la gestion des activités locales dans l'ensemble distribué (chapitre 4).

7
Conclusion et perspectives

Nous pensons avoir accompli des progrès personnels considérables en terme de couverture du sujet *Réseaux de capteurs sans fil*, un sujet qui est à l'ordre du jour en regard du développement durable et des bouleversements climatiques.

Les apports scientifiques de cette thèse se trouvent dans les interactions avec les systèmes d'information géographiques, les coordinations systèmes, et un flot général de conception incluant les aspects distribués et locaux.

Parmi les problèmes ouverts, on peut retenir :

- l'intégration de techniques radio de nouvelle génération, longue portée (LoRa), et la prise en charge des communications publiques proposées par les systèmes cellulaires.
- l'approfondissement des travaux sur des champs de capteurs inaccessibles (remote), et l'usage de mobiles, en particulier de satellites
- la structuration de la production de code autour de modèles unifiés, locaux et distribués, où Occam est un support possible.
- la validation à grande échelle des mécanismes de machines virtuelles permettant de se débarrasser des détails des outils de développement de bas niveau.

Le parcours de cette thèse a été (trop) long, la couverture posée d'emblée étant un problème qui s'est compliqué graduellement avec l'expansion énorme du domaine.

Bibliographie

[1] M. Weiser, « The computer for the 21st century », *Scientific american*, vol. 265, no. 3, p. 94–104, 1991. 11

[2] E. A. Lee, « Cyber physical systems : Design challenges », *in Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, p. 363–369, IEEE, 2008. 12

[3] « Berkeley sensor and actuator center ». `http://www-bsac.eecs.berkeley.edu/`. 12

[4] « Streetline – connecting the real world ». `http://www.streetline.com`, 2010. 12

[5] J. Markoff, « Can't find a parking spot ? Check smartphone. », *The New-York Times*, 12 juillet 2008. `http://www.nytimes.com/2008/07/12/business/12newpark.html`. 13

[6] « Stats – OpenStreetMap wiki ». `http://wiki.openstreetmap.org/wiki/Stats`. 19

[7] P.-Y. Lucas, V. L. H. Nguyen, T. P. Truong et B. Pottier, « Wireless Sensor Networks and Satellite Simulation », *in WiSATS 2015* (P. P. et al., éd.), vol. 154*in LNICST*, Springer, nov. 2015. 20, 47, 71, 85, 128

[8] Libelium, *Waspmote Technical Guide*, 5 juillet 2016. `http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf`. 27

[9] K. Ammouche, « Interpolation des mesures du signal radio reçu », rapport de master, Université de Bretagne occidentale, juin 2011. 28

[10] B. Pottier et P.-Y. Lucas, « Capteurs et programmation à bas niveau : une approche par machine virtuelle. Présentation de la plateforme arduino. », *in RESSACS'12, UPPA, `http://cpham.perso.univ-pau.fr/iWEB/RESSACS2012`* (C. Pham, éd.), 2012. 34

[11] C. Teodorov, « Modelling sensor networks as concurrent systems », rapport Master UBO, `http://wsn.univ-brest.fr/TER-rapports/TER-ModelingSensors.pdf`, Université de Bretagne occidentale, juin 2007. 39

[12] L. P. Pridal, « Projection de mercator, classe smalltalk ». `klokan@klokan.cz`, GDAL2Tiles Google summer of code 2007 et 2008, 2008. 43

[13] « Projection de mercator, wikipedia ». `https://fr.wikipedia.org/wiki/Projection_de_Mercator`. 44

[14] E. A. Lee, « Cyber-physical systems – Are computing foundations adequate ? », *in NSF Workshop on Cyber-Physical Systems*, 2006. Position paper. 44, 67

[15] « BMO OpenStreetMap wiki ». `http://wiki.openstreetmap.org/wiki/BMO`, Sept 2013. 46

[16] T. P. Truong et B. Pottier, « Monitoring of environment : A high performance method for radio coverage exploration », *in IEEE Radio, Indian Ocean*, IEEE, La Reunion, oct. 2016. 46, 52, 55, 60

[17] T. P. Truong et B. Pottier, « Wireless network on coastal topologies : Parallel simulation of radio coverage on cellular systems », *in A Connected Ocean*, SeaTech week, Brest, oct. 2016. 46, 52, 55, 88

[18] Y. Le Gall et R. Herry, « Calcul de couverture radio par lancer de rayon », rapport master ubo, `http://wsn.univ-brest.fr/TER-rapports/TERLeGallHerry.pdf`, Université de Bretagne occidentale, juin 2011. 47, 74, 127

[19] T. V. Hoang, « Cyber physical systems and mixed simulations, master thesis », rapport Master UBO, `http://wsn.univ-brest.fr/pottier/hoang.pdf`, Université de Bretagne occidentale, juin 2015. 52, 59

[20] D. Gajski, N. Dutt, A. Wu et S. Lin, *High Level Synnthesis, an introduction to chip and system design*. Springer Science+Business Media New York, 1992. 52

[21] G. Berry, R. Chandra, D. Gajski, K. Konigsfeld, P. Schaumont et I. Verbauhede, « The next HDL : If C++ is the answer, what was the question ? », *in Proceedings of 38th conference on Design automation*, p. 7172, ACM Press, `https://securewww.esat.kuleuven.be/cosic/publications/article-727.pdf`, 2001. 52

[22] Jet Propulsion Laboratory (CalTech), « U.s. releases enhanced shuttle land elevation data », sept. 2014. `http://www2.jpl.nasa.gov/srtm/`. 55

[23] « Graphviz - graph visualization software : Envisioning connections ». `http://graphviz.org`, 2016. 56

[24] IEEE, « 1516-2010 - ieee standard for modeling and simulation (m&s) high level architecture (hla)– framework and rules ». `https://standards.ieee.org/findstds/standard/1516-2010.html`. 59

[25] C. A. R. Hoare, « Communicating sequential processes », *Communications of the ACM*, vol. 21, p. 666–677, 1978. 59

[26] C. A. R. Hoare, *Communicating sequential processes*. Prentice Hall International, 1985. 59, 123

[27] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic et P. J. Hazewindus, « The design of an asynchronous microprocessor », *SIGARCH Comput. Archit. News*, vol. 17, p. 99–110, juin 1989. 59

[28] Mentor Graphics, « Handel-c language reference manual ». http ://www.mentor.com/products/fpga/handel-c/upload/handelc-reference.pdf. 59, 66

[29] University of Kent, « Welcome to kroc ». `http://projects.cs.kent.ac.uk/projects/kroc/trac`. 59, 68, 88, 121

[30] P. Welch, F. Barnes et S. O. Computing, « Position paper : "concurrency first", workshop : "curricula for concurrency" (oopsla 2009) ». 59

[31] G. Kremer et N. Melot, « Mobility and synchronizations in wireless sensors networks », rapport Master UBO, `http://wsn.univ-brest.fr/TER-rapports/KremerMelot.pdf`, Université de Bretagne occidentale, février 2010. 60, 63, 72, 127

[32] H. Dutta, T. Failler, N. Melot, B. Pottier et S. Stinckwich, « An execution flow for dynamic concurrent systems : simulation of WSN on a Smalltalk/Cuda environment », *in Proceedings of SIMPAR 2010 Workshops Intl. Conf. on simulation, modeling and programming for autonomous robots*, Dynamic languages for robotic and sensors systems (DYROS), (Darmstadt, Germany), p. 290 – 295, 15-16 novembre 2010. 60, 61

[33] N. Lynch, *Distributed algorithms*. Morgan Kaufman, 1996. 60

[34] D. Pountain et D. May, *A tutorial introduction to Occam programming*. New York, NY, USA : McGraw-Hill, Inc., 1987. 61

[35] A. Iqbal et B. Pottier, « Meta-Simulation of Large WSN on Multi-core Computers », *in Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim SCS Conference, (Orlando, USA), p. 133, Society for Computer Simulation International, April 2010. 61

[36] Nvidia, Nvidia Cuda *Programming Guide 2.0*, 2008. 61

[37] T. Failler, « Netgen : un générateur de code pour cuda, principes, implémentation et performances. », rapport Master UBO, `http://wsn.univ-brest.fr/TER-rapports/TERThibaultFailler.pdf`, Université de Bretagne occidentale, juin 2010. 61

[38] « TinyOS home page ». `http://www.tinyos.net/`. 66, 122

[39] C. L. Jacobsen et M. C. Jadud, « Towards concrete concurrency : occam-pi on the lego mindstorms », 2005. 68, 89

[40] SGS-Thomson Microelectronics Limited, *Occam 2.1 Reference Manual*, 1995. 69

[41] H. Nguyen Van Long, « Wireless sensor network to satellite interface », rapport Master USTH et UBO, `http://wsn.univ-brest.fr/pottier/long.pdf`, USTH et UBO, novembre 2014. 71, 128

[42] AMSAT, « Keplerian elements ». `http://www.amsat.org/amsat-new/tools/keps.php`, 2016. 72

[43] David Vallado, *Fundamentals of Astrodynamics and Applications*. Microcosm Press Year, mars 2013. 72

[44] S. Turner et A. Back, « A t9000 implementation of the p4 parallel programming model », *in Transputer research and applications 7* (I. Press, éd.), 1995. 87

[45] IEEE, « Transputer architecture », 2013. 88

[46] A. Chalmers, M. Mirmehdi, H. M. (eds et F. R. M. Barnes), « tranx86 - an optimising etc to ia32 translator », 2001. 88

[47] Inmos Limited, *Transputer Compiler Writer Guide*, 1988. 91

[48] « Site public pour tvm ». `http://concurrency.cc`. 95

[49] T. H. Kim, « Implementation of a lattice gas simulation », rap. tech., M.I.T., printemps 2007.

[50] T. J. Lightfoot et G. J. Milne, « Modelling emergent crowd behaviour », *in Proceedings of the Australian Conference on Artificial Life*, (Canberra), p. 159 – 169, The University of New South Wales, 6-7 décembre 2003.

[51] G. Russel, A. Cowie, J. McInnes, M. Bate et G. Milne, « Simulating vehicular traffic flows using the circal system », rap. tech., University Of Strathclyde, mai 1994.

[52] « Arduino home page ». `http://www.arduino.cc`. 121

[53] « Libelium, wireless distributed communications ». `http://www.libelium.com`. 121

[54] B. Pottier, P.-Y. Lucas, E. Keita, R. Herry, Y. LaGall et K. Ammouche, « Les capteurs dans la ville. (i) du maquettage à la génération de code, (ii) calcul de couverture radio, (iii) netgen : support pour l'implémentation », *in RESSACS'11, UPPA, `http://cpham.perso.univ-pau.fr/iWEB/RESSACS2011`*, 2011. 127

[55] B. POTTIER, P.-Y. LUCAS et A. PLANTEC, « Rdc : des simulateurs aux capteurs, transformations de sources occam », *in RESSACS'13, UPPA,* *http://cpham.perso.univ-pau.fr/iWEB/RESSACS2013* (C. PHAM, éd.), 2013. 128

A

Les systèmes matériels

L'objectif de ce chapitre est de présenter des principes et des fonctions que l'on trouve dans les plates-formes les plus courantes pour les capteurs sans fil. L'originalité de ces plates-formes est d'intégrer un processeur compact, basse consommation, doté de multiples périphériques. Les marchés de ces composants étant importants, beaucoup de constructeurs se sont intéressés à décliner des produits de fonctionnalité et puissance variables.

On peut citer :

- Atmel, avec la famille de microcontrôleurs ATmega, dont certains intègrent maintenant des émetteurs-récepteurs radio, ATmegaRF. Arduino, Libelium sont des exemples de cartes de prototypage utilisant cette famille de microcontrôleurs.
- Texas Instruments, avec les composants MSP430. La carte SoftBaugh comporte un microcontrôleur MSP430.
- Cypress, avec des circuits reconfigurables à gros grain. Ces circuits contiennent des modules génériques qui sont configurés en périphériques spécialisés. Cela rend le circuit modulaire et adaptable à une plus grande diversité de réalisations.
- ARM,
- Jennic, avec leur dernière génération de micro-contrôleurs sans fil JN5148, à ultra basse consommation d'énergie. Ils sont compatibles avec les protocoles de transmission sans fil utilisés dans les réseaux de capteurs.

Ces composants se retrouvent dans des produits finalisés tels que Arduino, Libelium, Micaz, ... Dans ce chapitre, nous détaillons l'architecture typique de ces systèmes, les composants typiques qui les forment en permettant l'assemblage des capteurs périphériques, leurs caractéristiques de fonctionnement par rapport aux usages sans fil : mémoires internes, modes d'économie d'énergie, statuts.

Bien évidemment, la plupart de ces systèmes-sur-puces (ou *system on a chip*) se destinent à des usages très compacts, ouverts et économes en énergie.

Le chapitre va d'abord présenter les architectures, et nous nous appuierons principalement sur les circuits de Texas et Atmel. Ensuite, nous décrirons les interfaces périphériques en insistant particulièrement sur celles qui procurent des capacités d'intégration : les unités asynchrones (USART),

les bus de communication tels que I2C et SPI. Parce que la conception d'un système nécessite l'usage de protocoles logiciels, nous donnerons deux exemples d'usage des interconnexions par port série : intégration des radios telle qu'elle est effectuée pour les modules XBee de la société Digi, et l'intégration des GPS et des instruments de marine, avec la norme NMEA.

Les explications sont données par composants selon un plan standardisé qui comporte des informations d'usage et des informations techniques, qui peuvent être lues optionnellement.

A.1 Architecture d'un système et programmation

Les microcontrôleurs sont des systèmes sur puce : sur un seul circuit intégré (une même puce de silicium), sont regroupés plusieurs composants élémentaires. On y trouve le processeur, la mémoire vive, la mémoire flash et les périphériques. Ces périphériques servent à l'acquisition des données provenant des capteurs physiques (comme le convertisseur analogique vers numérique) et aux communications avec les systèmes extérieurs. Lorsqu'ils ne sont pas utilisés, ces périphériques sont désactivés afin de réduire la consommation électrique. L'ensemble du circuit est très compact, avec une taille inférieure au centimètre carré. Nous allons décrire ces éléments, en prenant comme exemple les circuits MSP430 de Texas Instruments et Atmega d'Atmel.

A.1.1 Le microprocesseur, cœur du microcontrôleur

Le processeur est la partie centrale du microcontrôleur. Son rôle est de lire les instructions du programme enregistré en mémoire, de les décoder et de les exécuter. Le processeur comporte une unité de calcul, appelée unité arithmétique et logique (UAL), et des mémoires pour les données et les programmes. Il est relié à des périphériques par des bus de communication. Ces périphériques ont des fonctions variées : ils peuvent permettre l'accélération de l'exécution des programmes, ou permettre au microprocesseur de communiquer avec d'autres composants.

On distingue deux types d'architecture parmi les processeurs :

- l'architecture Von Neumann : les programmes et les données sont stockés dans la même zone mémoire. C'est le cas du microprocesseur MSP430 de Texas Instruments. Les données peuvent être traitées comme un programme, ce qui permet de modifier à la volée les instructions, comme la modification du champ d'adresse, pour faire des boucles par exemple.

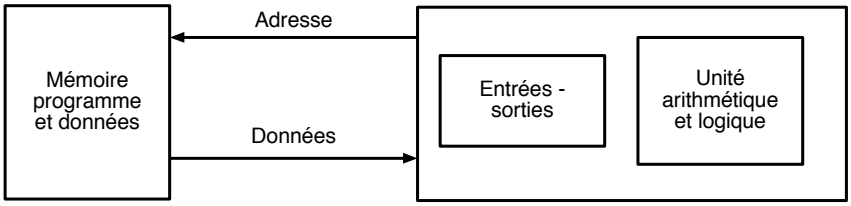


FIGURE A.1 – Architecture von Neumann

- l'architecture Harvard : les mémoires des programmes et des données sont séparées, avec un accès distinct pour chaque mémoire. C'est le cas du microprocesseur ATmega, d'Atmel. Cette structure permet de transférer des instructions et des données simultanément, ce qui améliore les performances. Par ailleurs, les instructions étant séparées des données, le programme est protégé des modifications accidentelles.

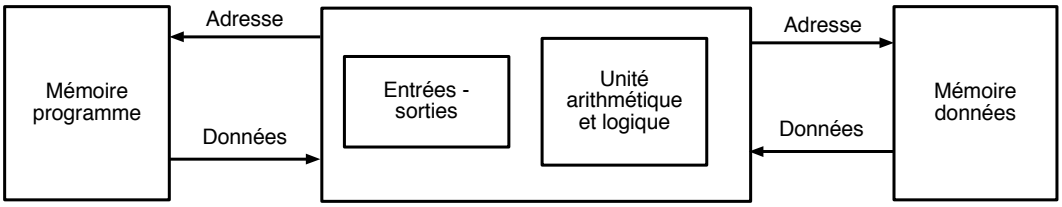


FIGURE A.2 – Architecture Harvard

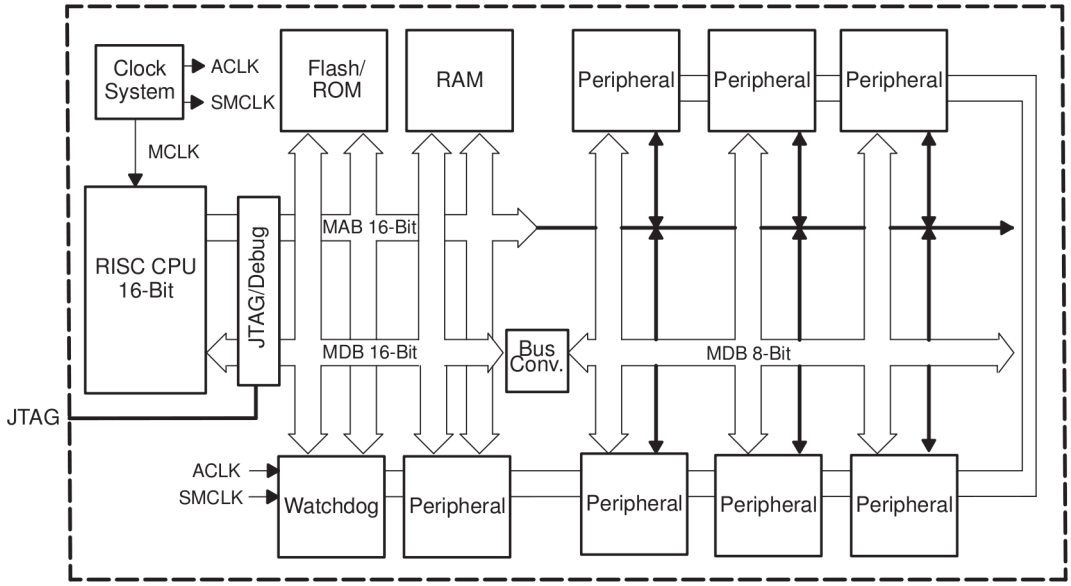


FIGURE A.3 – Architecture d'un microcontrôleur de type MSP430. (source Texas Instruments)

A.1.2 Organisation de la mémoire

L'espace mémoire du microcontrôleur est découpé en plusieurs zones, correspondant à des fonctions physiques différentes : la mémoire vive qui reçoit des variables, la mémoire flash utilisée pour implanter le programme et des données de l'application, les registres permettant de configurer, lire et écrire des données pour chaque périphérique intégré, et la pile d'exécution.

Suivant l'architecture du microprocesseur, l'organisation de la mémoire sera différente. Sur le circuit MSP430F1612, qui possède une architecture « Von Neumann », les adresses de 16 bits des programmes voient par exemple :

- 5 Ko de mémoire vive (RAM, random access memory). Elle reçoit les données temporaires nécessaires aux calculs. Cette mémoire est volatile, et perd son contenu quand l'alimentation électrique est coupée. Elle est accessible directement en lecture et écriture.
- 55 Ko de mémoire programmable (ROM, read-only memory), dont le premier rôle est de permettre le chargement initial du micrologiciel à exécuter. Elle est accessible en lecture et, sous certaines contraintes, en écriture, par exemple pour stocker des données capturées. Cette mémoire est de type « flash », c'est-à-dire qu'elle peut être effacée puis réécrite plusieurs fois, de manière électrique.
- les registres associés aux modules périphériques permettent de configurer ces composants en sélectionnant des modes de fonctionnement tels que par exemple la vitesse d'un port série, ou le nombre de bits dans une conversion numérique. Ils servent aussi de ports pour les échanges de données et permettent d'associer un vecteur d'interruption si ce mode de fonctionnement

est choisi.
- les fonctions spéciales sont des registres qui définissent le comportement du système : activation des interruptions, sélection des périphériques, modes de veille du système.
- la table des vecteurs d'interruption : ces registres contiennent en mémoire l'adresse du programme qui est exécuté lorsqu'un événement survient.

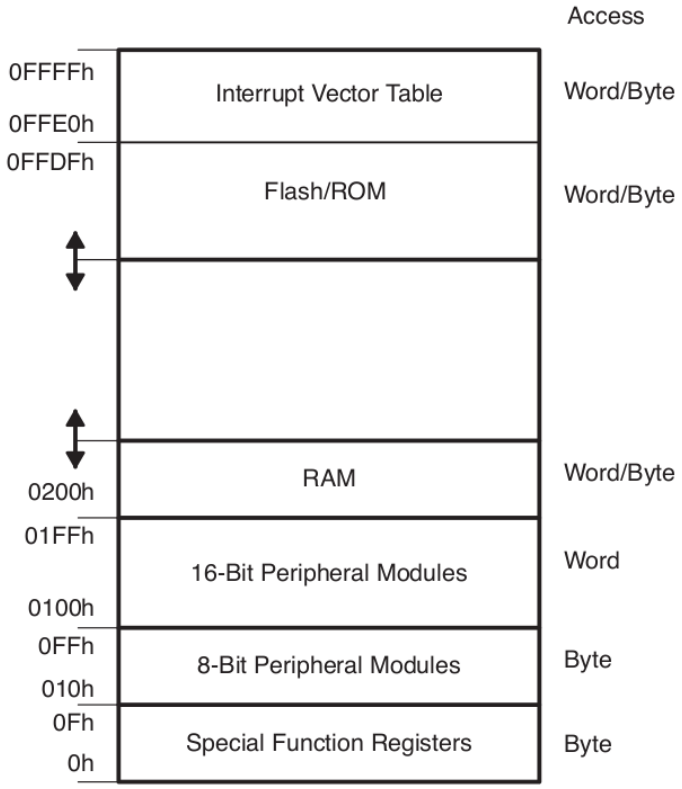


FIGURE A.4 – Espace d'adressage d'un microcontrôleur de type MSP430. (source Texas Instruments)

A.1.3 Les périphériques

Les périphériques sont capables d'effectuer des tâches spécifiques. Les plus courants sont :
- le timer, pour la gestion du temps. Il permet de mesurer des intervalles de temps, de compter les impulsions, de générer des impulsions, isolées ou périodiques, et de mesurer des fréquences.
- le convertisseur de signal analogique vers numérique (ou CAN). Ce composant produit périodiquement une valeur numérique à partir de la tension électrique à ses bornes. Il sert à l'acquisition des mesures venant des périphériques analogiques externes.
- le convertisseur de signal numérique vers analogique (ou CNA). Il produit une tension analogique à partir de valeurs numériques venant du microcontrôleur.
- les ports d'entrées-sorties numériques. Ils servent à l'acquisition de signaux logiques externes au microcontrôleur, et produisent également des sorties logiques pour commander des ressources externes.
- les périphériques de communication. Ce sont par exemple le port série, les bus *Serial peripheral interface* (SPI) et *Inter integrated chip* (I2C).

Il est possible d'ajouter des périphériques supplémentaires si les périphériques internes ne sont pas suffisants, grâce à des modules additionnels.

Le MSP430 contient plusieurs modules configurables. La configuration est faite en écrivant dans des registres de 8 bits ou 16 bits.

A.1.4 Mémorisation, chargement et pérennité des données

Dans le cas du MSP430, une option des outils de développement est constituée par la chaîne mspgcc, de souche GCC, adaptée au microcontrôleur de Texas Instruments. Le code source est en général écrit en langage C, avec des extensions permettant d'adresser les fonctions particulières du matériel cible.

Le compilateur et le chargeur (msplinker ?) produisent un fichier qui se divise en segments correspondants au code, aux données constantes, aux données variables, aux interruptions. Lors de l'installation d'un programme, ce fichier est passé par l'interface Jtag (.) qui l'interprète et l'installe aux localisations physiques adéquates.

Pour le développeur, Mspgcc permet de localiser les segments à l'aide de directives Pragma. Dans le cas d'une interruption, on écrirait par exemple :

```
1 #define PORT1_VECTOR (à documenter)
2 interrupt (PORT1_VECTOR) MyInterruptServiceRoutine (void)
3 {
4
5 }
```

Les pragmas sur les segments de données, ou les déclarations de registres se décrivent selon l'exemple qui suit :

```
1  #define PORT1_VECTOR (à documenter)
2  #include <msp430.h>
3
4  #pragma location=0x1800
5  const unsigned char port_bit = BIT0;
6
7  void main(void)
8  {
9    WDTCTL = WDTPW + WDTHOLD;  // disable watchdog
10   P1DIR = port_bit;          // set P1.x as output
11
12   while(1)
13   {
14     P1OUT ^= port_bit;         // toggle P1.0
15           __delay_cycles(100000);
16   }
17 }
```

L'usage du plan mémoire est donc complètement à disposition du programmeur. Concernant la mémoire flash, des précautions doivent toutefois être prises par le programme. Le Msp divise sa mémoire flash en pages que l'on peut effacer unitairement. L'usage dynamique de cette mémoire présume que l'on sache quelles pages correspondent à quelles données. Sachant de plus que l'état de cette mémoire dépend des exécutions antérieures, il convient de s'assurer que les pages sont effectivement vides avant d'y écrire. L'exemple de code qui suit est proposé par Texas Instruments pour indiquer la procédure d'effacement.

```
1 void erase_segment (uint16_t * start_addr)
2 {
3         FCTL1 = FWKEY + ERASE;                     // Set Erase bit
```

```
4   FCTL3 = FWKEY;                          // Clear Lock bit
5   *start_addr = 0 ;                       // Dummy write to erase Flash segment
6   FCTL3 = FWKEY + LOCK;                   // Set LOCK bit
7 }
```

La procédure d'écriture implique elle aussi une opération de contrôle, la lecture étant immédiate. De plus, cette procédure implique l'installation d'un mode de fonctionnement particulier pour que l'écriture se fasse.

```
1 void flash_write_char (uint8_t * dest, uint8_t value)
2 {
3   FCTL3 = FWKEY;                          // Clear Lock bit
4         FCTL1 = FWKEY + WRT;                    // Set WRT bit for write operation
5         *dest = value ;
6         FCTL1 = FWKEY;                          // Clear WRT bit
7         FCTL3 = FWKEY + LOCK;                   // Set LOCK bit
8 }
```

En conclusion, on voit que la construction d'un programme capable de tirer parti complètement des ressources d'un tel microcontrôleur est une tâche qui reste complexe dans l'état actuel des choses, la diversité des périphériques, leur activité concurrente, l'observation de temps de réponse adéquats, et même la gestion des bancs mémoires résilients opposant de sérieuses contraintes.

Concernant les performances d'accès aux données, la mémoire vive est accédée en *nv* cycles, alors que la mémoire flash implique *nf* cycles en lectures, l'effacement ou l'écriture d'une page de *n* octets prenant *ne* cycles typiquement.

A.1.5 Exemple d'usage : enregistreur de chemin GPS en mémoire flash

Il est intéressant d'illustrer le fonctionnement d'un programme en regard des ressources de mémorisation, et pour cela, nous proposons d'expliquer comment un enregistreur de parcours *(data logger)* peut tirer parti des ressources de mémorisation stables, et les contraintes que cette mémorisation amène. Plusieurs versions de l'enregistreur ont été développées pour tirer parti de la mémoire locale flash, ou pour alimenter une machine extérieure en données via un lien série. Ici nous expliquons l'usage embarqué enregistrant en mémoire flash.

L'enregistreur est relié au récepteur GPS par un lien série (UART). Le GPS donne sa position régulièrement (une fois par seconde, généralement) en envoyant une trame au format NMEA à l'enregistreur. Ce dernier décode les informations, extrait la position ainsi que des métadonnées comme la date et l'heure, le nombre de satellites utilisés pour le calcul, la précision du calcul. Un bouton sur la carte permet de mémoriser certaines positions, ce sont des points d'intérêts.

- les registres de configuration et de lecture de l'UART
- un vecteur d'interruption utilisé pour signaler l'arrivée d'un caractère,
- un tableau en mémoire vive, utilisé pour la lecture d'une trame et son analyse
- deux tableaux en mémoire flash, l'un contenant les trames transférées et l'autre stockant les points d'intérêts.

Le déchargement de la mémoire flash est réalisé une fois le système arrêté. On utilise un boîtier de contrôle qui se branche sur le port Jtag du microcontrôleur d'une part, et sur un ordinateur d'autre part. Il est capable de lire la mémoire et la copier sur l'ordinateur. On récupère un fichier de vidage de la mémoire (dump), qui est traité par un logiciel pour reconstruire les trames GPS, ainsi que les points d'intérêt.

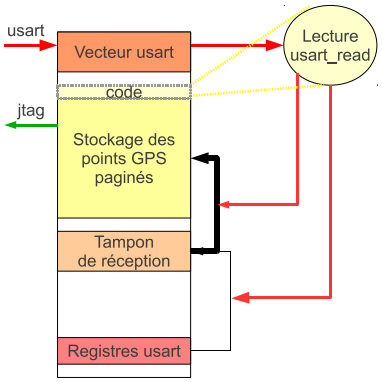


FIGURE A.5 – Implantation mémoire et flot de contrôle pour l'enregistreur GPS. Activation d'une routine d'interruption à réception du premier caractère, lecture en RAM des caractères reçus, dans une page de 512 octets, sauvegarde en mémoire flash de cette page à échéance de sa capacité.

A.1.6 Exemple d'usage : un analyseur de paquets

Un deuxième exemple de programme qui utilise plusieurs ressources du microcontrôleur, est un analyseur de paquets radio. Il nécessite le transceiveur radio de Texas Instruments, le CC2420, utilisé comme récepteur. Cet équipement est présent sur la carte de Softbaugh, DZ1612. Le programme transmet tous les paquets interceptés à un ordinateur pour les visualiser.

Cet analyseur comporte une partie de réception radio dont la fonction est la capture des paquets perçus par le transceiveur CC2420. Celui-ci est accédé par un bus SPI, décrit en section A.4. L'intérêt de ce logiciel est majeur, puisqu'il permet de présenter l'activité radio autour du système de réception. On utilise plusieurs sous-ensembles de l'espace mémoire pour son implantation :

- les registres de configuration et de lecture du bus SPI,
- un vecteur d'interruption utilisé pour signaler l'arrivée d'un paquet, la lecture de ce paquet s'effectuant par scrutation, le paquet ayant été initialement stocké dans un tampon du transceiver,
- un tableau en mémoire vive, utilisé pour la lecture du paquet,

Ensuite, le paquet est transmis sur le port série (par l'UART, voir section suivante), vers un ordinateur. Le paquet est formaté pour être lu par le logiciel Wireshark, un programme d'analyse de trafic de réseau informatique.

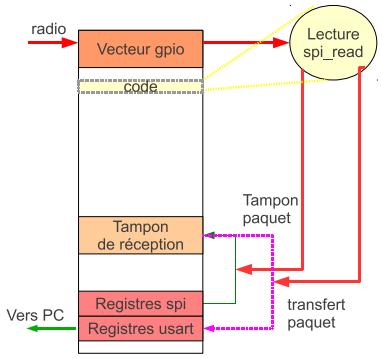


FIGURE A.6 – Implantation mémoire et flot de contrôle pour l'analyseur de paquets. Éveil sur interruption du circuit radio, une fois le paquet reçu, transfert de ce paquet en RAM, traitement et écriture sur le lien série via l'usart.

Schéma du plan mémoire. Explication sur le téléchargement/boot, les amorçages, les vecteurs d'interruption, les limites possibles de la flash, *ses procédures*, les vitesses d'accès, statuts par rapport au processeur. À relayer vers les procédures d'amorçage et la structure du logiciel préparées par le compilateur.

		MSP430F1610	MSP430F1611	MSP430F1612
Memory Main: interrupt vector Main: code memory	Size Flash Flash	32KB 0FFFFh – 0FFE0h 0FFFFh – 08000h	48KB 0FFFFh – 0FFE0h 0FFFFh – 04000h	55KB 0FFFFh – 0FFE0h 0FFFFh – 02500h
RAM (Total) Extended Mirrored	Size Size Size	5KB 024FFh – 01100h 3KB 024FFh – 01900h 2KB 018FFh – 01100h	10KB 038FFh – 01100h 8KB 038FFh – 01900h 2KB 018FFh – 01100h	5KB 024FFh – 01100h 3KB 024FFh – 01900h 2KB 018FFh – 01100h
Information memory	Size Flash	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h
Boot memory	Size ROM	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h
RAM (mirrored at 018FFh - 01100h)	Size	2KB 09FFh – 0200h	2KB 09FFh – 0200h	2KB 09FFh – 0200h
Peripherals	16-bit 8-bit 8-bit SFR	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h

FIGURE A.7 – On explique l'organisation mémoire d'un MSP430, espace d'adressage, mémoire vive, mémoire flash, programme, data

A.2 UART : aspects matériels et usages

Dans la section A.3, nous présenterons les procédures structurant les flots série, de manière à faciliter l'exploitation de périphériques évolués, tels que les GPS, et les boîtiers de communication radio, par exemple.

Présentation du périphérique

L'UART (Universal Asynchronous Receiver-Transmitter) est un périphérique de communication qui permet de transmettre et recevoir des données en série sur quelques fils.

Historique

L'uart est le composant utilisé dans la communication. Le standard correspondant se nomme RS-232, aussi appelé port série.

Apparition de l'uart : périphériques - imprimantes - modem - terminaux

Vitesses : 9600, 192000

toujours utilisé : GPS, XBee, AX25/ Cubesat, adaptateur USB/série.

Niveaux électriques 3.3 V, 12 V, etc.

Une communication asynchrone en série des données

Le composant UART est à l'interface entre le bus de communication interne du microcontrôleur et deux fils de données externes (figure A.8). Il permet d'émettre, ou de recevoir des caractères d'une

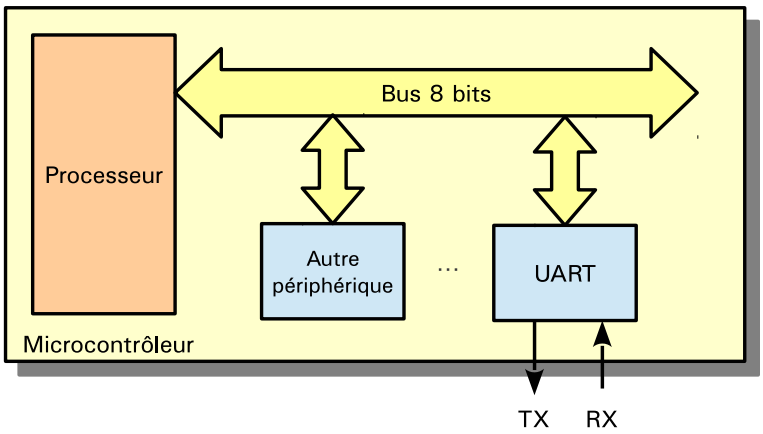


FIGURE A.8 – Intégration système de l'UART. Le bus de communication relie le processeur (CPU) aux périphériques. Les deux signaux externes de l'UART, *RX* pour la réception et *TX* pour l'émission, sont connectés aux broches du microcontrôleur.

taille de 5 à 8 bits sur un fil de communication dédié.

Il permet donc de transformer une liaison parallèle, où 8 bits sont transmis simultanément sur le bus de données du microcontrôleur, en liaison série : les 8 bits sont transmis successivement sur un fil de liaison.

Le périphérique est dit asynchrone, car il ne transmet pas de signal d'horloge. Ainsi, les deux entités qui communiquent doivent partager les mêmes paramètres de transmission. Il existe ainsi plusieurs débits de transmission, exprimés en bauds.

La liaison est dite duplex, car les données peuvent être transportées simultanément dans les deux sens.

Au repos, le signal est au niveau logique haut (1 logique). La trame UART est constituée des informations suivantes, dans l'ordre (figure A.9) :

- un bit de start à 0 (niveau logique bas), il sert à synchroniser le récepteur,
- les données du caractère à transmettre,
- un bit de parité, facultatif (paire ou impaire),
- un bit de stop, niveau logique haut.

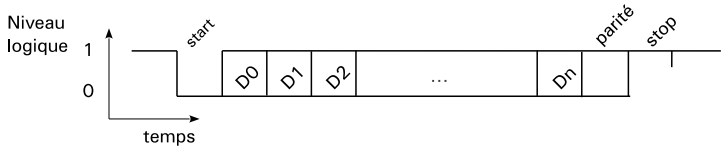


FIGURE A.9 – Une trame UART, avec un bit de start, n bits transmis, un bit de parité et un bit de stop.

Les paramètres de transmission

Le paramétrage est le suivant :

- le débit de la transmission : il correspond à la fréquence à laquelle les données sont transmises,
- la taille des données à transmettre : de 5 bits à 8 bits,

- la présence d'un bit de parité,
- le nombre de bits de stop (1 ou 2 bits).

Quelques exemples de débit rencontrés sont 4800, 9600 ou 115200 bauds.

Fonctionnement architectural du périphérique

Le schéma A.10 présente le diagramme simplifié de l'UART. Les composants suivants sont présentés :

- un tampon d'émission qui stocke le caractère à émettre,
- un registre à décalage d'émission qui forme la trame UART,
- un registre à décalage pour recevoir une trame,
- un tampon de réception qui stocke le caractère reçu,
- un registre indiquant le statut du caractère reçu,
- un générateur de débit qui contrôle les deux registres à décalage.

L'émission et la réception des données sont donc distinctes, car elles sont assurées par des registres indépendants.

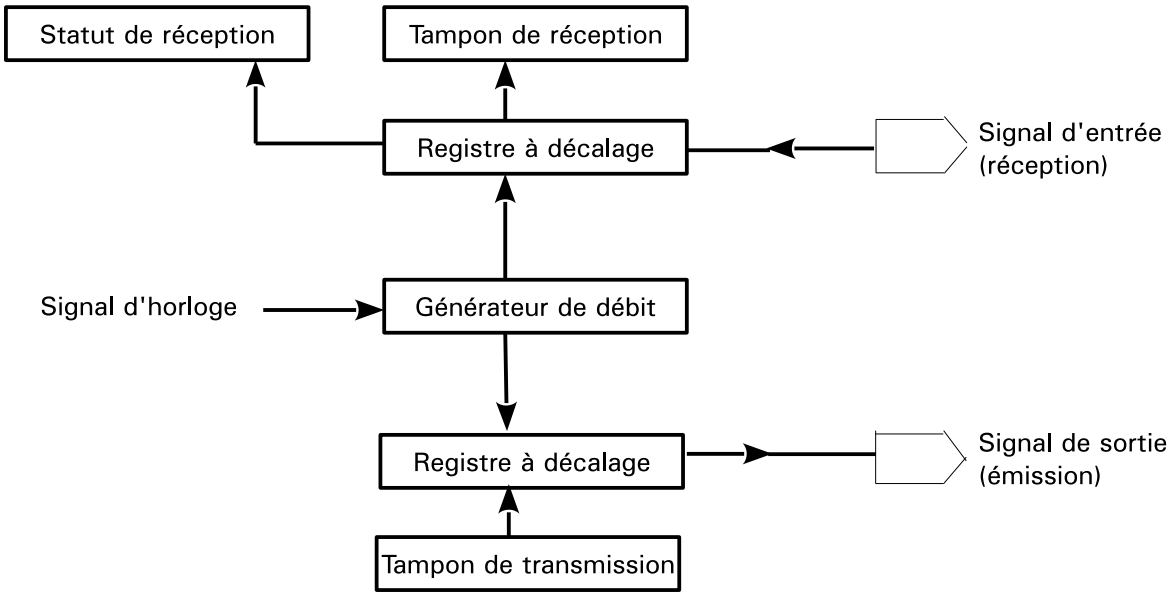


FIGURE A.10 – Fonctionnement simplifié de l'UART. Le bus de communication n'est pas représenté, il est relié aux tampons d'émission et de réception.

Flot de fonctionnement

L'émission d'un octet. On écrit un octet dans le registre du tampon de transmission. Cet octet est transféré dans un registre à décalage, qui est commandé par le générateur de débit. La trame UART constituée est émise bit par bit sur le signal de sortie.

La réception d'un octet Les valeurs du signal entrant sont stockées dans un registre à décalage, commandé par le générateur de débit. Lorsqu'une trame complète est reçue, le caractère est copié dans le tampon de réception. Un registre de contrôle du caractère reçu est mis à jour avec le statut de

la dernière réception. Il indique notamment si une erreur de réception est survenue, ou si un nouveau caractère est arrivé sans que le précédent ne soit lu (*overrun*).

On peut configurer l'UART de manière à ce qu'une interruption soit déclenchée à chaque nouveau caractère reçu. Cela évite au processeur de scruter en continu l'état du registre de réception.

On a vu en section A.1.6 que la table des vecteurs d'interruption est utilisée pour implanter les adresses de fonctions associées aux exceptions. De cette manière, il suffit d'implanter un pointeur sur notre fonction serialASync() pour qu'elle soit automatiquement appelée lorsque le matériel détecte des circonstances telles que caractère reçu, ou tampon vide.

Exemple de configuration matérielle du périphérique

Nous allons présenter la méthode de configuration d'un périphérique UART à travers l'exemple du microcontrôleur MSP430.

Le MSP430 dispose de deux périphériques USART. Ce sont des périphériques génériques qui peuvent servir aussi bien pour des transmissions asynchrones (UART), que synchrones (comme SPI ou I2C).

La configuration se fait par des registres accessibles depuis l'espace d'adressage du microcontrôleur. Les registres utilisés dans la configuration sont listés dans le tableau A.11.

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h with PUC

† Does not apply to '12xx devices. Refer to the register definitions for registers and bit positions for these devices.

FIGURE A.11 – Registres utilisés pour configurer l'USART 0. (source Texas Instruments)

Le rôle des principaux registres

USART control register Ce registre contrôle la taille du caractère, la parité, le nombre de bits de stop.

Transmit control register Il sélectionne la source d'horloge utilisée.

Receive control register Il indique les erreurs de réception.

Baud rate control register 0 et 1 Il configure le débit de données de l'UART.

Modulation control register C'est la modulation utilisée à partir du signal d'horloge choisi.

Receive buffer register Ce registre accessible seulement en lecture contient le dernier caractère reçu.

Transmit buffer register Ce registre retient le caractère en cours de transmission. L'écriture dans ce registre déclenche la transmission.

SFR module enable register Ce registre active l'UART.

SFR interrupt enable register 1 et 2 Ces registres configurent si une interruption est levée lorsqu'un caractère est reçu, ou bien qu'une transmission est terminée.

Un exemple de configuration Le protocole à suivre pour configurer l'UART est défini dans la documentation :

```
1  U0CTL   = SWRST           // Mode configuration
2  ME1    |= UTXE0 | URXE0; // TX, TX modules actifs
3  U0CTL  |= CHAR ;          // 8-bit, sans parite, 1 bit stop
4  U0TCTL |= SSEL1 ;
5  U0BR1   = 0x3 ;           // modulation 4800 bauds
6  U0BR0   = 0xF0 ;
7  U0MCTL  = 0x08 ;
8  IFG1   &= ~(BIT4 + BIT5);
9  U0IE   |= URXIE0 ;         // interruption activee en reception
10 U0CTL  &= ~SWRST ;         // Fin du mode configuration
```

Flot de programmation : scrutation

Deux méthodes sont possibles pour programmer l'écriture et la lecture sur le microcontrôleur : la scrutation par attente active et l'utilisation d'interruptions.

L'attente active utilise le processeur jusqu'à ce qu'un caractère arrive dans le tampon de réception.

La fonction qui transmet un caractère se présente sous la forme suivante :

```
1 void uart_tx (char data)
2 {
3   // attente si la transmission precedente est en cours
4   while (!(U0TCTL & TXEPT)) ;
5   U0TXBUF = data ;
6 }
```

De même, un exemple de fonction pour lire un caractère :

```
1 char uart_rx ()
2 {
3   // attente active avant de lire un caractere
4   while (! ( IFG1 & URXIFG0 );
5   return U0RXBUF;
6 }
```

Flot de programmation : interruption

La deuxième méthode permet de placer le processeur en veille, ou de réaliser une tâche de fond.

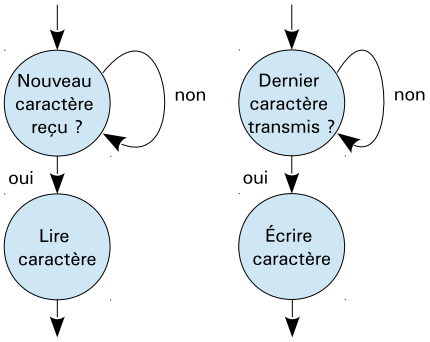


FIGURE A.12 – Diagramme pour la scrutation

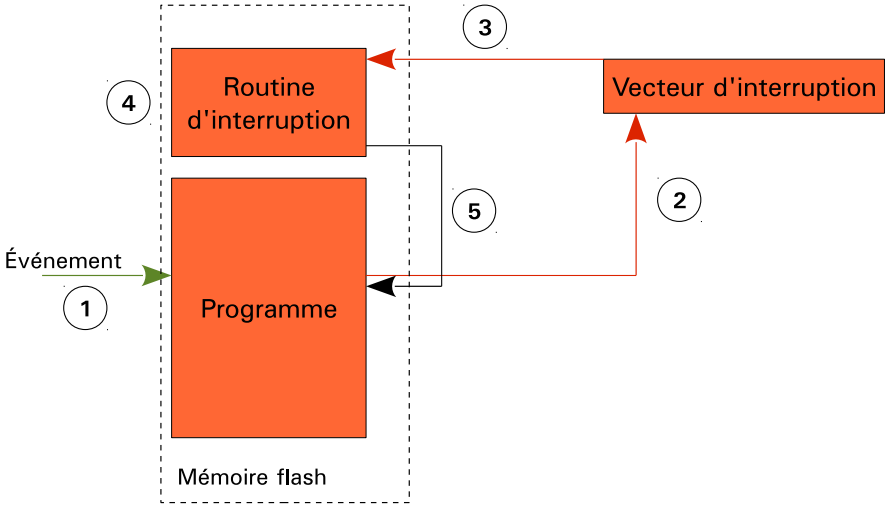


FIGURE A.13 – Diagramme pour l'interruption

Le câblage du circuit

L'UART utilise deux fils :

- un pour la transmission,
- l'autre pour la réception.

Pour que la liaison électrique soit correcte, la masse des deux systèmes doit être reliée par un troisième fil.

MSP430 Deux UART sont disponibles sur le MSP430. Les signaux sont reliés aux broches suivantes :

- P3.4 : UART0, transmission,
- P3.5 : UART0, réception,
- P3.6 : UART1, transmission,
- P3.7 : UART1, réception.

Arduino Quatre UART sont disponibles sur l'Atmel Mega 2560.

Exemple d'application

Ce périphérique a été utilisé dans deux montages : une liaison avec un GPS sur MSP430 et une communication avec un module radio XBee sur carte Arduino.

Liaison avec un GPS

Liaison avec un module XBee

Photos des montages

A.3 UART : protocoles de haut niveau

A.3.1 NMEA

Le protocole NMEA permet la communication entre les équipements d'un bateau, comme le récepteur GPS, l'anémomètre, le sonar, le gyrocompas, le pilote automatique. Il s'agit de transmettre une phrase de communication par un émetteur à un ou plusieurs périphériques à l'écoute, en utilisant des caractères ASCII.

La décomposition d'un message NMEA est la suivante :

- le message NMEA débute par un signe dollar « $ »,
- les cinq caractères qui suivent identifient l'émetteur (2 caractères) et le type de message (3 caractères),
- les champs qui suivent sont délimités par une virgule,
- si une donnée n'est pas disponible, le champ est vide : ainsi se trouvent deux délimiteurs consécutifs,
- une astérisque suit le dernier champ,
- une somme de contrôle de deux chiffres,
- la phrase est terminée par les caractères retour chariot et nouvelle ligne (<CR> et <LF>).

Exemple de trame $GPGGA …

A.3.2 Interface modem

A.3.3 Autres interfaces : SiRF

A.4 SPI, I2C, GPIOs, CAN, CNA

A.5 Le convertisseur analogique-numérique

- pinout (schéma),
- registres de configuration,
- déclenchement d'une conversion,
- lecture du résultat

A.6 Les Unités asynchrones et leurs protocoles

Il existe deux blocs USART (universal asynchronous or synchronous receiver-transmitter) sur le MSP430. Ces modules peuvent être configurés dans l'un des trois modes suivants :

- I2C
- UART
- SPI.

A.7 Vers une approche de haut niveau du système d'exploitation

Le laboratoire a repris une des idées initiales du projet NetGen qui est de programmer les aspects locaux des capteurs en s'appuyant sur un langage concurrent intégrant la communication et le temps. Nous sommes parvenus à exécuter des programmes Occam sur deux familles de capteurs qui sont Arduino [52] et Libelium [53], cela en utilisant le logiciel public TVM de l'arbre du compilateur Occam [29]. Cette approche est très intéressante sur plusieurs plans : support architectural pour l'acquisition et le système, possibilité d'intégrer les communications radio, et extension du cadre de la synthèse de réseau (par exemple, ordonnancement des acquisitions de capteurs fonctionnant simultanément).

A.8 (NetGen : simulation et déploiement des réseaux de capteurs)

Partie basse : la programmation des capteurs

Il s'agit de produire un code exécutable pour chaque capteur. Ce code intègre la contribution distribuée, le contrôle et l'acquisition sur les capteurs locaux. En conformité avec les simulations côté réseau, partie haute, l'expression locale la plus naturelle serait de procéder en Occam ou C, syntaxe proche de Cuda.

Ces capteurs sont souvent réalisés à base de micro-contrôleur typiquement basse consommation (par exemple MSP430, AtMEGA 2560, Jennic, ARM, etc.). Ces architectures comportent une variété de mémoires internes, des interfaces d'acquisition et de contrôle, des bus périphériques. La figure A.14 montre l'architecture d'un MSP430 avec ses mémoires et ses unités de périphériques. La réduction de la consommation est obtenue par gradation des activités en éteignant les ressources logiques inutiles.

TABLE A.1 – Les périphériques du MSP430 et de l'Atmel, utilisés au laboratoire

Périphériques de communication	MSP430 F1612	ATMEGA 2560
Périphériques génériques d'entrée/sortie	48	54
Convertisseurs Numérique/Analogique (CNA)	2	0
Convertisseurs Analogique/Numérique (CAN)	8 (12 bits)	16 (10 bits)
Canal *Pulse Width Modulation* (PWM)	16 (16 bits)	16 (16 bits)
Inter Integrated Chip (I2C)	2	1
Serial Peripheral Interface (SPI)	1	1
Universal Asynchronous Receiver Transmiter (UART)	2	4

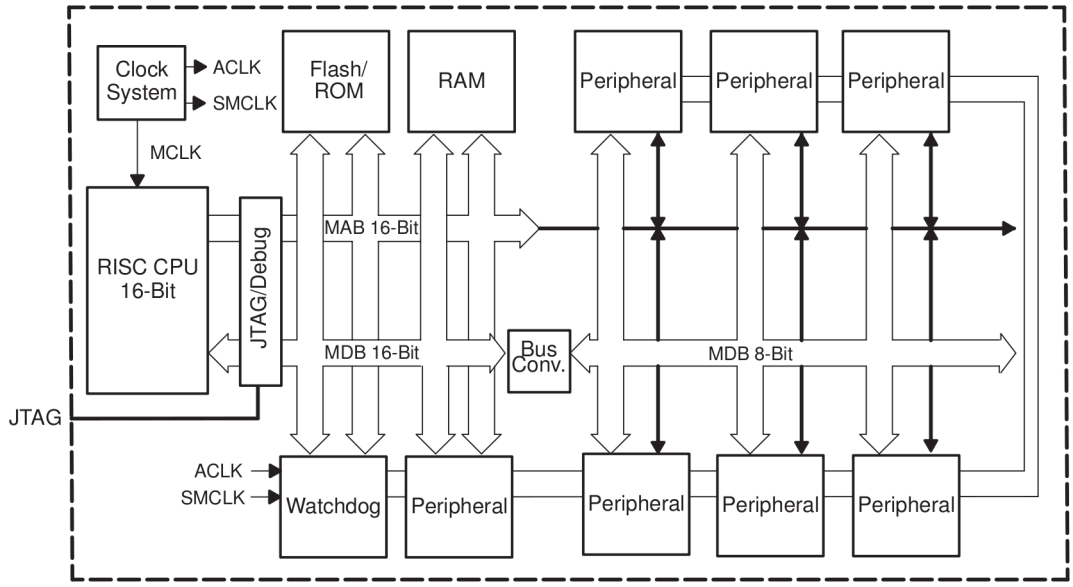


FIGURE A.14 – Architecture d'un microcontrôleur de type MSP430. (source Texas Instruments)

Programmer signifie produire des logiciels embarqués pour des environnements contraignants et qui requièrent un apprentissage qui peut être long et complexe. La figure A.15 montre des modules assemblés dans un tel programme : contribution distribuée conforme à la simulation, exécutée sous forme de communication radio (en orange), tâches concurrentes locales qui produisent et traitent les données acquises des capteurs physiques (en vert). Un système environnemental interrogerait, par exemple, des capteurs physiques pour la température, l'hygrométrie, la concentration de gaz, ou la détection de présence par infrarouge.

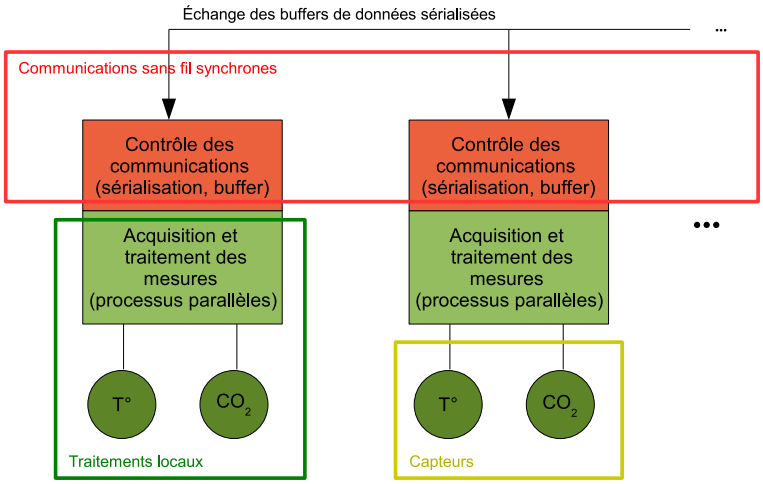


FIGURE A.15 – La figure montre l'assemblage de deux nœuds via un système radio.

Plusieurs modèles de programmation sont connus tel que la programmation à bas niveau en C, l'usage d'un microsystème d'exploitation (TinyOS [38]), ou des Environnements de Développement Intégrés (IDE), tels qu'Arduino ou sa version commerciale Libelium. Ces IDE sont présentés en figure A.16, ils s'appuient sur une syntaxe proche du langage C++, et une organisation de programme représentant un automate qui, après une initialisation, exécute un cycle d'actions de communication et de capture.

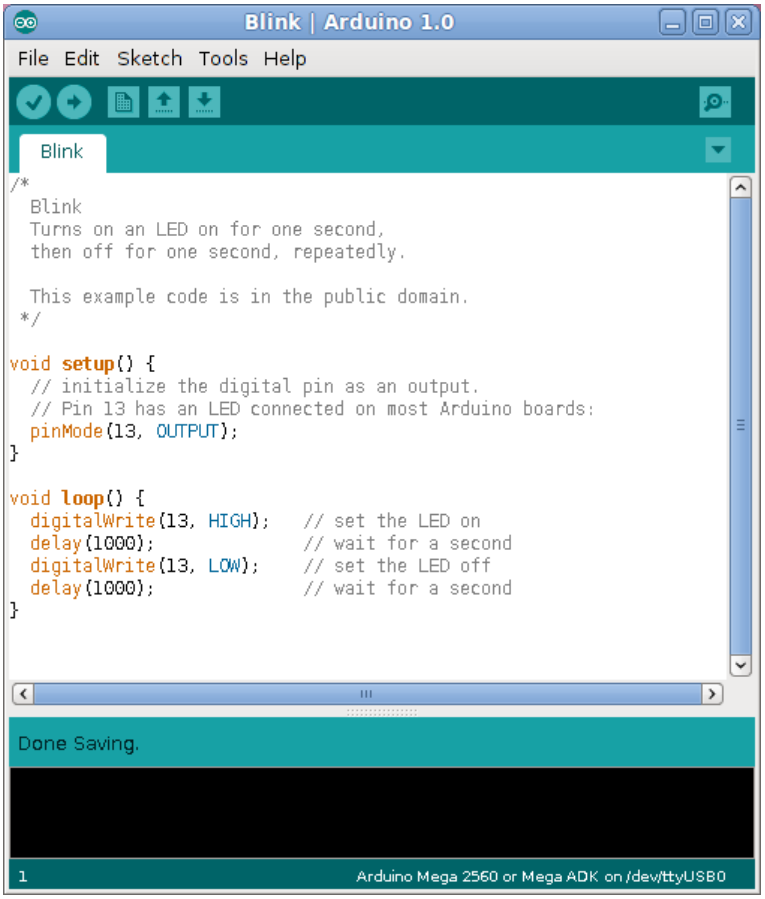


FIGURE A.16 – L'environnement de développement intégré Arduino permet de compiler le code et programmer un système embarqué. Deux fonctions structurent le code : la fonction `setup()` pour l'initialisation, et la fonction `loop()` pour l'exécution.

Nous avons étudié et pratiqué plusieurs de ces plates-formes, en concluant qu'elles sont perfectibles en terme de *sécurité* et de *fiabilité* de la programmation. L'idée est venue d'investiguer Occam pour ses capacités à modéliser la concurrence locale, inhérente aux capteurs, la vérification possible des programmes et sa capacité à la spécification des algorithmes distribués. Occam s'appuie sur le concept de *Communicating Sequential Processes* (CSP) permettant de représenter l'exécution sous forme de construction parallèle ou séquentielle en réaction à l'environnement des processus (C.A.R. Hoare [26]).

De plus, les aspects de programmation à haut niveau d'Occam facilitent le développement des algorithmes : la concision de l'écriture et la vérification lors de la compilation rendent leur exécution plus robuste.

A.9 Contexte technique actuel

Nous avons choisi deux systèmes pour accueillir la machine Occam : Libelium et Arduino. Le prototypage des communications radio utilise des modem XBee, qui sont adaptables sur les deux familles de cartes.

Le développement de protocoles de communication radio nécessite des outils de mise au point adéquats. Pour cela, nous avons développé un analyseur de protocole radio qui capture les données

émises par les émetteurs pour une analyse en temps réel ou hors ligne. L'analyse des communications radio échangées entre les entités permet d'étudier le fonctionnement du réseau en conditions réelles et ainsi ajuster les algorithmes et la topologie.

L'autonomie du réseau est une considération importante à prendre en compte lors de la conception : elle conditionne la durée de vie d'une entité et du réseau entier. La consommation d'électricité des nœuds est mesurée pour les cartes que nous avons utilisées. La modélisation de ces informations sert à élaborer des algorithmes efficaces pour augmenter l'autonomie du réseau.

A.9.1 Deux cibles pour le test et le développement des applications

Le simulateur comme la production de code des capteurs ne peuvent être considérés sérieusement qu'à partir de mesures soigneuses et d'études des phénomènes physiques ou humains distribués [14].

Dans le cadre d'une thèse, les mises en œuvre pratiques sont limitées par des facteurs de coûts et de temps. Pour cette raison, le contexte de développement vers les applications vont cibler les deux familles de cartes embarquées : Libelium et Arduino. Celles-ci comportent le même micro-contrôleur Atmel qui supporte TVM, grâce au portage de l'Université de Kent (Simpson, Jacobsen, Jadud [3]). TVM est disponible sous licence libre.

Les sections **??** et **??** donnent quelques détails sur les intérêts respectifs de ces plates-formes. La section 4 détaille l'organisation et les contenus que nous proposons pour la dernière année de thèse.

A.10 Une plate-forme « capteur sans fil » : la carte DZ1612 de Soft-Baugh

La carte DZ1612 est une plate-forme de démonstration ZigBee réalisée à partir de la note d'application de Texas Instruments.

Cette carte comporte deux principaux circuits :

- un microcontrôleur MSP430 de Texas Instruments
- un transmetteur radio CC2420 (ChipCon), bande de fréquence 2.4 GHz.

Le microcontrôleur réunit dans un même circuit les principaux composants d'un ordinateur :

- le microprocesseur,
- la mémoire vive (RAM),
- la mémoire permanente de stockage (mémoire de type flash). Elle contient le programme exécuté par le processeur.
- des ports d'entrée et de sortie, comme des convertisseurs analogiques-numériques, ou des bus de communication.
- des composants spécialisés pour accélérer les traitements.

Le microcontrôleur MSP430 peut traiter des signaux analogiques et numériques. Les interfaces analogiques :

- 6 convertisseurs analogique vers numérique (CAN, ou ADC pour analog to digital converter)
- 2 convertisseurs numérique vers analogique (CNA, ou DAC pour digital to analog converter)

Les interfaces numériques :

- 48 ports d'entrée ou sortie génériques
- 16 ports PWM (pulse width modulation, modulation codée par largeur d'impulsion)
- 2 interfaces de communication synchrone ou asynchrone (USART), configurables

Les interfaces de communication permettent d'utiliser les périphériques suivants :

- 2 ports série asynchrones (UART, universal asynchronous receiver-transmitter)
- 2 ports I2C (inter-integrated chip)
- 1 port SPI (serial peripheral interface)

Le microcontrôleur est basse consommation : les périphériques non utilisés sont désactivés pour réduire la consommation électrique. Le fabricant indique une consommation de x mA en activité et de x μA en veille. (faire un tableau avec le détail des consommations)

A.11 Utilisation des interfaces matérielles

A.11.1 L'accéléromètre

A.11.2 Le compas

A.11.3 Enregistreur de parcours

Le GPS produit des informations de localisation sous la forme de trames, au format NMEA. Chaque trame est une ligne que l'on peut afficher sur l'écran de l'ordinateur. Les trames sont produites en continu, à la fréquence de 1 Hz environ.

On utilise la carte SoftBaugh pour lire les données du GPS et stocker en mémoire le trajet parcouru.

B

Historiques des frontaux de NetGen, et contributions

On peut résumer cet historique comme suit [1] :

(A) Février 2010 : Interface de saisie sur fond de carte : UIPick, PickView PickModel et NetworkGeometry. Application à la gestion de mobiles dans les réseaux de capteurs (niveau géométrique exploité dans Nicolas Melot, [31], et figure 3.1).

Le pseudo-code qui suit montre comment le contrôleur du système MVC interagit avec le gestionnaire de graphe implanté dans NetworkGeometry. Des paramètres tels que la portée ont été configurés dans le modèle PickModel. La saisie d'un point provoque le calcul du modèle et son affichage (figure B.1).

```
1 redButtonPressed (PickController)
2     insertion du clic dans le modele (PickModel)
3      appel de buildGraph: gc (contexte afficheur)
4           buildNetAndDisplayOn: gc (PickModel)
5                knownNetwork: self  drawingOnlyOn: gc( NetworkGeometry )
6                 // construction modele abstrait, et  calcul d' un objet
                      geometrique )
7           // modification du modele qui force l'affichage
```

(B) Février 2011 : Intégration de chemins de saisie *capturés (MSP430+GPS)* sur le système de la figure 2.4. Affichage interactif. Production de modèles NetGen pour les capteurs localisés.

(C) Mai et Juin 2011 : Intégration des *Shapefiles* présentant les bâtiments de Brest sur l'afficheur initial. Calculs de portée radio par lancer de rayon (TER M1 [18, 54], voir figure 3.8 puis les figures C.7, C.8, C.9, C.10 dans les premières évolutions de GMap),

1. L'historique des développements est déduit des estampilles des archivés Store, une base de données qui permet de retrouver versions et évolutions. Le premier développement (A) vient de B.Pottier et existait au démarrage de la thèse, GMap (D) et Quickmap (F) ont été créés dans le cadre de la thèse et à l'initiative de Pierre-Yves Lucas, ainsi que (E). (G) a été développé par B.Pottier, en partie en vue de la thèse de M.Traoré et celle de E.Keita. (H) est une coopération entre Pierre-Yves Lucas, Van Long Huu et B.Pottier.

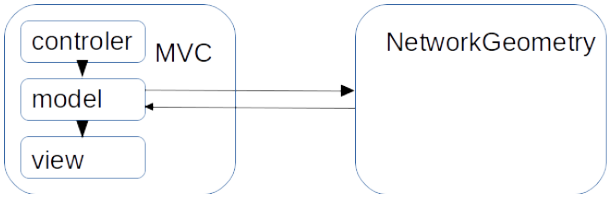


FIGURE B.1 – Relation entre un client MVC (interface PickCell, par exemple) et le noyau de synthèse de graphes. Un contrôleur PickController envoie son modèle géométrique PickModel au noyau qui le complète en construisant le graphe associé et sa représentation graphique. La vue PickView affichera immédiatement cette représentation.

(D) Février 2013 : GMap (voir annexe C) logiciel intégrant le géo-positionnement des capteurs, et l'accès aux systèmes de cartes par tuilages (OpenStreetMap, Google map, …). Les points de saisie permettent de synthétiser des réseaux NetGen, à portée constante.

(E) Mars 2013 : Intégration des données temps réel pour le serveur *smart city Santander* (Ressacs 2013 [55] et figure 3.2).

(F) Avril 2014 : Première version de QuickMap, brosseur universel géo-positionné avec un défilement interactif.

(G) Mai 2014 : Transfert des informations de QuickMap vers l'outil de gestion cellulaire PickCell qui a été refait à partir de A.

(H) Juin 2014 : Intégration avec GPredict, présentation de défilement d'un satellite CubeSat. Travaux avec Van Long Huu sur les synchronisations et communications entre capteurs et mobiles [7, 41]. Voir le chapitre 5.

C

GMap : a first NetGen-compatible map browser

This chapter will present two evolutions of the initial Netgen program for support of precise geographic positions and map browsing, then for displaying buildings or obstacles representations extracted from OpenStreetMap databases. The present tools allow to interact with the more common public information systems such as Google map and OpenStreetMap.

The map browser is a tool allowing to display various kind of maps and to represent locations of interest such as sensors set in the country. As this tool is developed on the same platform as NetGen, the procedures described in chapter **??** will apply to access the software :

- start a fresh image and ensure that the NetGen package is loaded with one of the last version (1.28.1.2.5 should work)
- open the store dialog from VisualWorks main window
- select GoogleMap package (we need to change this name)
- select version 1.15.5.2 or later, and type load from the pop-up menu

The initial window displays as show figure C.1.

C.1 Moving on the map

A predefined position is visible inside the xtile and ytile fields in the left column of the browser. Just below, a zoom factor is also provided. Whatever are these values, by pushing the refresh map button, the browser will download geographic information to be presented in the graphic pane.

All around this pane, four sidebars allow to move the graphics top, down, left, and right. The four rectangles at the corner of this view control moves on diagonal directions.

By changing the zoom factor, the absolute view size will decrease or increase. As an example going from 15 to 16 increase the level of details.

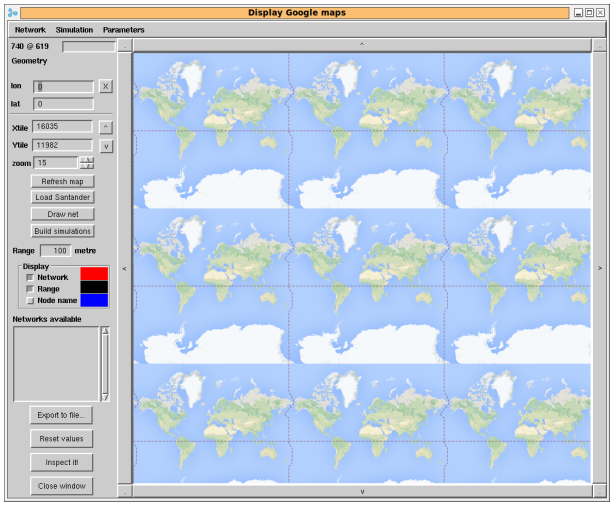


FIGURE C.1 – Initial view on the map browser : the right part displays tiles from the map from public servers. The left column displays geographic information and allows to control network presentation.

C.2 Loading networks

Networks are loaded from external files (further versions will allow direct selection from the interface).

Currently, GPX file format is used, as it is a very common way to describe set of points featured with attributes.

C.2.1 Scenario for loading informations

Suppose that by moving on the map, we have reached a particular region where a sensor network is setup or planned.

First, let us comment what a GPX file is. In this format, we find a header that keep information about the initial source of contents. As an example, a header from the Santander website could contain :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <gpx version="1.0" creator="NetGen for Santander">
3   <metadata>
4   <name>SmartSantander's sensors</name>
5       <desc>Sensors in city: Santander, Spain</desc>
6       <link>http://smartsantander.eu/</link>
7       <time>2013-06-13T17:45:10</time>
8 </metadata>
```

After this there is a list of entries for each of the location documented by the file. In the case of our river, we will find tens of similar entries such as :

```
1 <wpt lat="36.679926936710501" lon="4.911090436802451">
2   <name>H1-2</name>
3   <sym>El Kseur</sym>
4 </wpt>
5 <wpt lat="36.682937769536323" lon="4.919011088180600">
6   <name>I2</name>
7   <sym>El Kseur</sym>
8 </wpt>
```

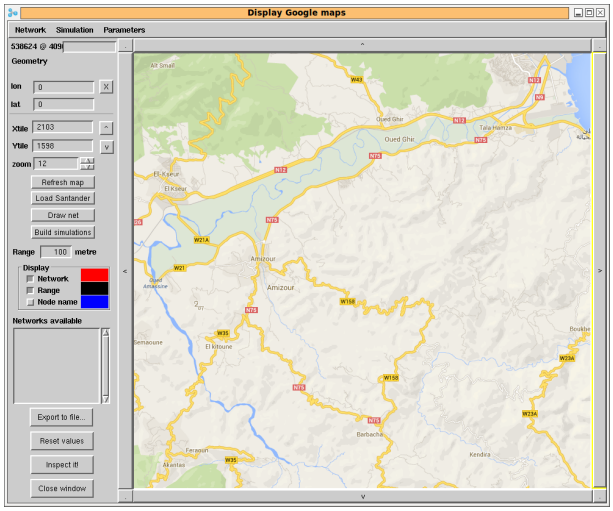


FIGURE C.2 – The browser is pointing to a North Africa river where sensors are to be installed for water observations.

```
<wpt lat="36.686345105073165" lon="4.929854203839318">
  <name>J3</name>
  <sym>El Kseur</sym>
</wpt>
...
```

This is a very short information since no practical values appear from sensors. The file extraction presents three waypoints (from the initial purpose of the NMEA standard for GPS), with geographic coordinates as decimal expression of degrees. Following we find a name for this particular point, and a symbol to display.

C.2.2 Loading a network

By using the *Network* menu at top-left we can select the *Load Gpx* function that brings a dialog to select a GPX file. In our case, it is *soummam.gpx* file to remember the name of the river and the file format. The file is parsed and its contents appears as points on the graphic part.

The symbols appearing in the entries of the file are used to group waypoints together inside networks. This networks are shown in a list in the left column. They are selectable, and as an example, the network *El Kseur* is validated for display figure C.3.

C.2.3 Network configuration

Several of the functions of NetGen as described in the previous chapter are available from this front-end window which capabilities exceed the picking tool shown chapter **??**. As an example, the range used to decide whether a sensor is connected to another can be defined using a dedicated numeric field. Furthermore, the present tool has precise knowledge about geographical points and related distances including display distances. Thus, the distance can be defined as meters conforming to radio capability specification.

On figure C.3, the range has been tuned to the point where each sensor in the El Kseur network was reachable, giving a necessary range of 3 000 meters to include all the sensors. The window does

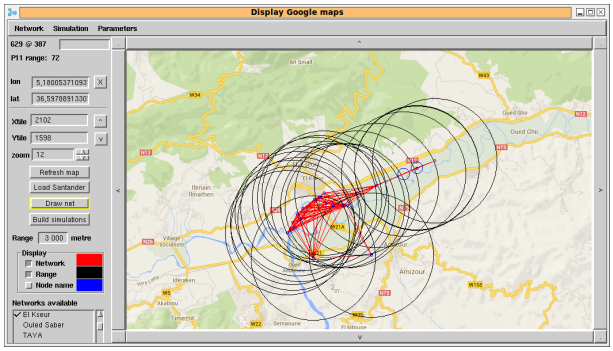


FIGURE C.3 – The browser is pointing to a North Africa river where sensors are to be installed for water observations.

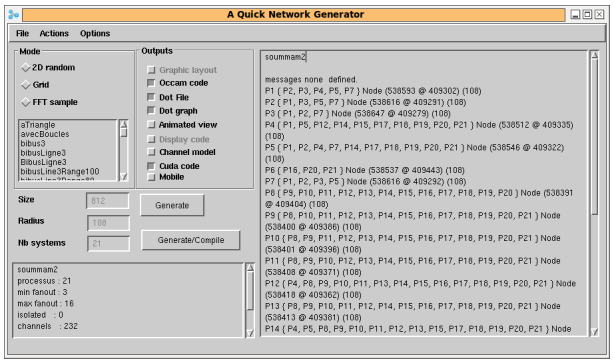


FIGURE C.4 – Netgen window with El Kseur specification and statistic.

not react directly to range modification, it is necessary to call *Draw net* button.

Some colouring functions ease the display of sensor names, range circles, and connectivities.

The mouse location over a graphic presentation is tracked on the top-left of the window :

- point coordinates inside the window, changing to red when the mouse is precisely over a sensor.
- this case the logical name of the node is shown in the edit field.
- a second line presents the range and position of the closest sensor, or a communication channel in the case where the mouse is over such a channel.

C.2.4 Network generation

To reach Netgen functionalities related to the process graph specification, it is sufficient to depress the build simulations button : the specification is loaded in every Netgen window (chapter **??** and figure C.4) for further use : building simulation, graph drawing (see figure C.5), etc.

After this exploration we have learned that this network has 21 nodes with maximum connectivity of 16, with 232 communications channels.

Executing the simulation brings a value of 4 hops for the network diameter. We probably have interest to reduce the general range keeping the further nodes to a 3 000 m range. An important aspect behind sensor nodes and map is that the simulator generated in Occam or Cuda preserve the logic link between simulation processes and graphical representation. Thus a simulator can animate the graphical view from outside concurrent execution.

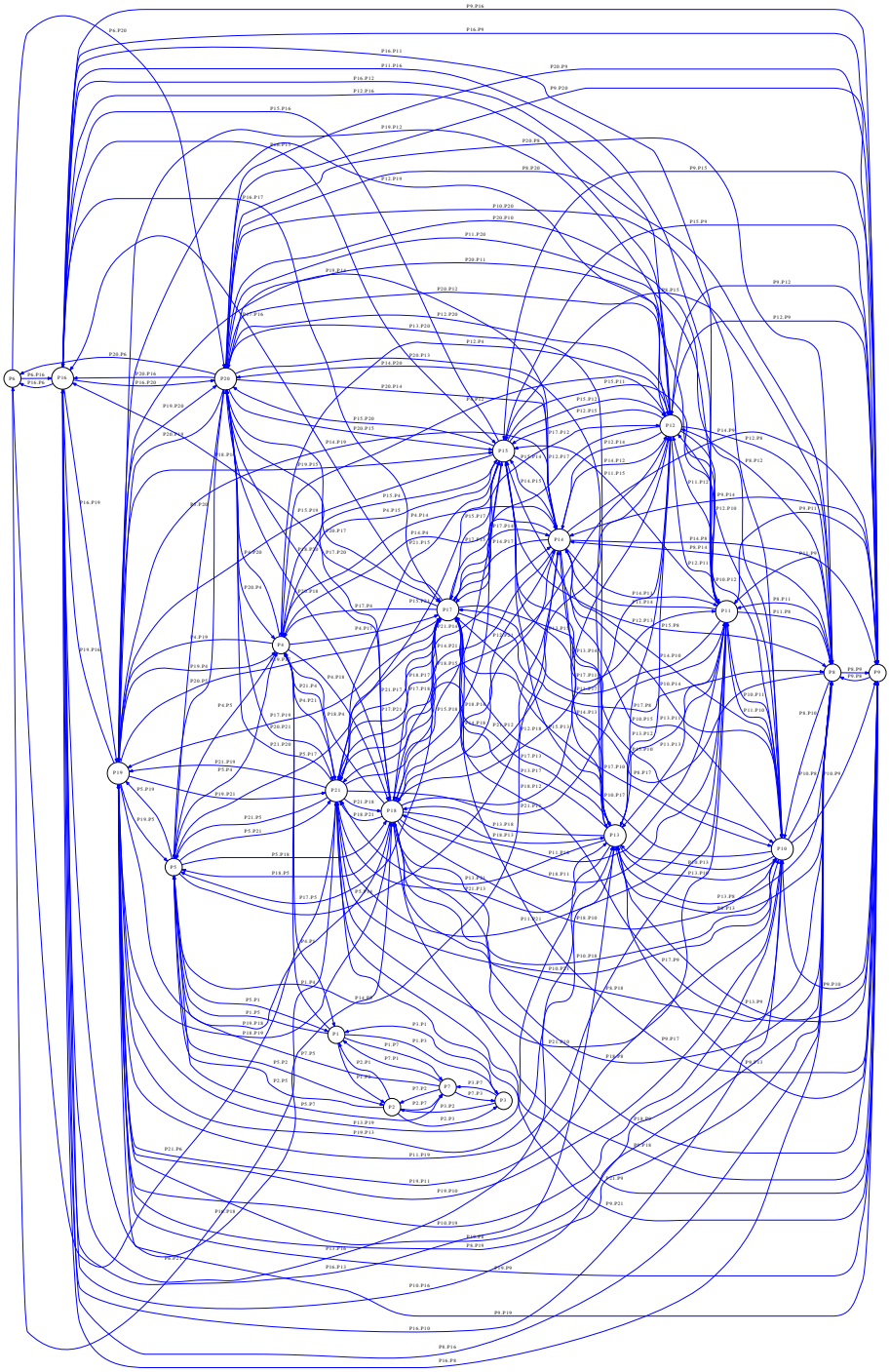


FIGURE C.5 – Logic graph for El Kseur location network.

C.3 Loading building architectures

The package concerned with building representations is MapAccess one, also available from the same store server as for chapter **??**.

In addition to this package, we need some files in the format of "shapefile". This time, we propose to do our scenario on the case of Brest city which administration decided to produce large description as 80 000 buildings set.

- start a fresh image and ensure that the NetGen package is loaded with one of the last version (1.28.1.2.5 should work)
- open the store dialog from VisualWorks main window
- select MapAccess package (in the future, likely to be merged with the map browser)
- select version 1.6 or later, and type load from the pop-up menu
- open a system browser, look for the MapAccess package, and the comments for this package. It is a button in the middle of the browser window. This comment looks like the following sentences :
  Display maps from tile servers, like Google Map or OpenStreetMap. Georeference points on the map.
  Display objects from shapefiles. Library is located here : http ://wsn.univ-brest.fr/MapAccess/library/libShapeFile.tar.
  Run 'make' to compile it. BMO shapefile is located here : http ://wsn.univ-brest.fr/MapAccess/bmo/.
  Copy the two files shp and shx in the same directory.
  This comment is likely to change, but we will keep location allowing to download software for the package : dynamic libraries compiled for Linux and MacOSX, and shapefiles for Brest. We do not support Window currently.
- in your working directory, download the libraries :
  `wget http://wsn.univ-brest.fr/MapAccess/library/libShapeFile.tar.`
  Desarchive the tar file by doing tar xf libShapeFile.tar.
- and download the two shapefiles :
  `wget http://wsn.univ-brest.fr/MapAccess/bmo/bati-WGS84.shp`
  and
  `wget http://wsn.univ-brest.fr/MapAccess/bmo/bati-WGS84.shx.`
  (ftp, curl, or any web browser can do similar work)

C.3.1 Checking the configuration

To check the installation, it is best to open a system browser on class ShapefileReader (figure C.6). This class uses directly the public domain library for assessing file conforming to the shapefile format. It translates external definitions into objects to be displayed on a derivative of the map browser interface.

It can be necessary to adapt the two directories for includeDirectories, for libraryDirectories to the platform, observing that a regeneration can be possible based on this public software.

C.3.2 Interface opening

By selecting the Tools menu from the main window, then choosing "Universite... MapAccess" the new interface opens. This interface is quite similar to the map navigation interface, at the exception at the left column. Having the interface open, by clicking Reset values, then Refresh map, we obtain a default view on Brest city (figure C.7).

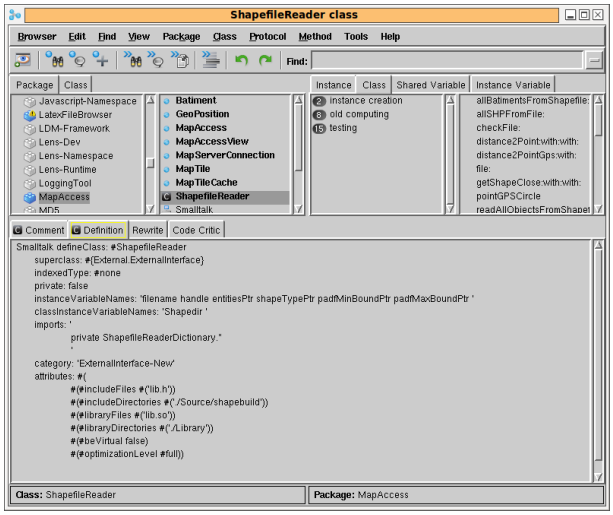


FIGURE C.6 – Path configuration for the dynamic libraries

FIGURE C.7 – Brest presentation before loading a shapefile

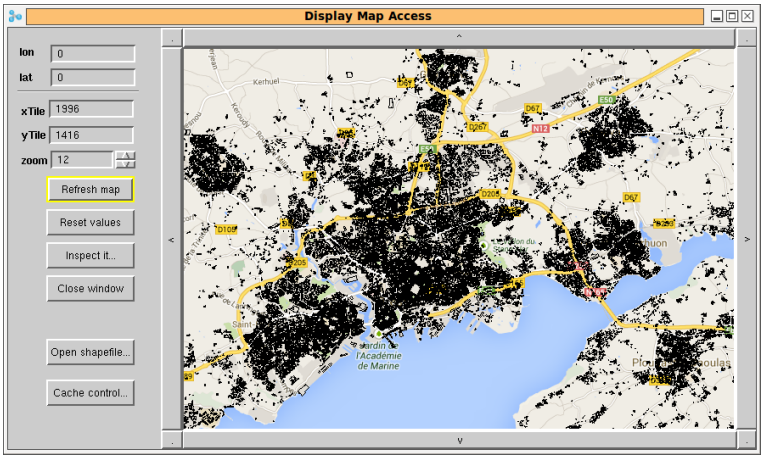


FIGURE C.8 – Brest presentation once the shapefile is loaded

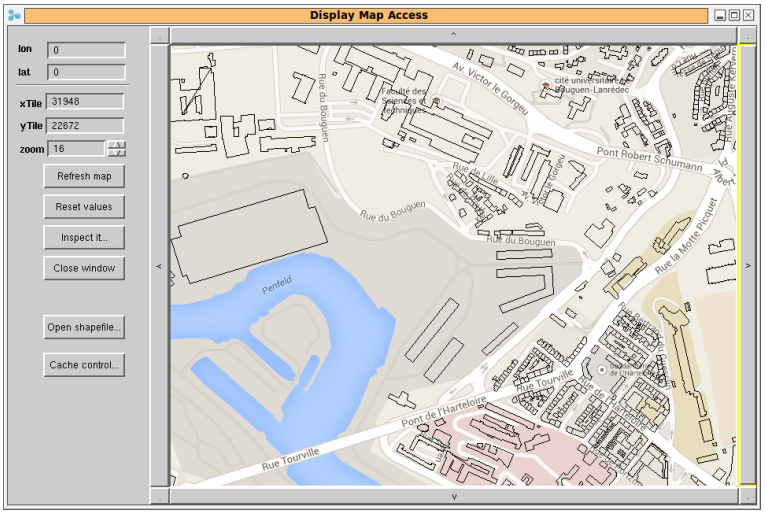


FIGURE C.9 – Brest university and river Penfeld after loading BMO shapefile, Google Maps

C.3.3 Loading shapefile

The button Open shapefile brings a dialog where a .shx file must be selected The figure C.8 shows Brest city map with more than 80 000 buildings represented as polyline 2D objects.

C.3.4 Browsing the city

The capabilities of the map browser exists in this preliminary tool. As example, one can zoom and move the display view changing the level of details. The geographic coordinates being preserved, it is possible to focus very precise situations to enable simulations and computations. See figures C.9 and C.10.

Switching between map layout is possible using a dialog installed under the cache control button. This dialog also allows to empty a cache used in the present software to speed up access to distant map tiles.

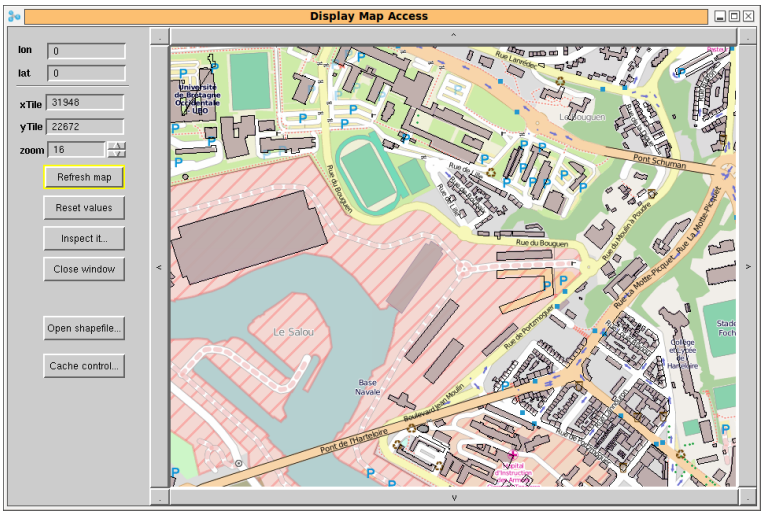


FIGURE C.10 – The same view as for figure C.9, using OpenStreetMap.

Modélisations, simulations et synthèse pour des réseaux dynamiques de capteurs sans fil

—Résumé

L'intégration de l'environnement et des systèmes d'information progresse très vite depuis 10 ans. Cette intégration permet de suivre des évolutions naturelles, physiques, sociétales ; de les mesurer, de les comprendre ; quelquefois de les contrôler. On peut assimiler cette intégration à des besoins, tels que les changements climatiques ou les économies de ressources ; mais aussi à des progrès technologiques dans les domaines des systèmes miniatures, des communications sans fil et des capteurs.

Dans ce contexte, nous avons d'abord effectué un apprentissage technologique, en réalisant plusieurs petits systèmes et des logiciels applicatifs de bas et de haut niveau. Nous nous sommes attaqués à la partie frontale des chaînes logicielles, celle qui permet de passer des bases de données pour Systèmes d'Information Géographique à l'implantation et l'exploitation de systèmes distribués de capteurs. QuickMap est ainsi un logiciel de navigation multi-services, incluant OpenStreetMap, construit pour le placement des capteurs et la spécification de systèmes de cellules physiques.

Sur la plateforme NetGen, nous avons réalisé un simulateur concurrent associant un ou plusieurs mobiles à un ou plusieurs champs de capteurs. Une application est l'étude des interactions entre satellite en orbites basses, réseaux de capteurs lointains et stations de contrôle. Le séquencement par le temps permet d'associer plusieurs activités aériennes et au sol, en bénéficiant de hautes performances.

Enfin, les questionnements méthodologiques ont amené à considérer la possibilité de virtualisation, à la fois du capteur, en le recouvrant d'une machine virtuelle, et à la fois du système d'observation distribué, en utilisant la plateforme NetGen. Nous avons maquetté un capteur et son interface radio en termes de processus communicants, en cherchant à réduire la complexité et la diversité de la programmation des petits systèmes matériels.

Plusieurs de nos réalisations sont effectives et servent à des projets de recherche actifs.

Cette thèse a été réalisée à l'Université de Brest, laboratoire Lab-STICC, grâce à une allocation de la communauté brestoise BMO.

Mots-clés : Réseau de capteurs sans fil, satellite, systèmes d'information géographique, systèmes distribués, processus concurrents, simulation haute-performance, NetGen, Quickmap, Occam, Cuda, TVM, Gpredict, micro-contrôleur, IEEE 802.15.4, ...

Modeling, simulations and synthesis for dynamic wireless sensor networks

—Abstract

The integration of environment and information systems is progressing quickly since 10 years. This allows to monitor natural, physical or societal evolutions; to capture their logic and sometimes to control their effects. This integration is feasible thanks to many technical and scientific progresses: sensors, wireless communications, system on chips, batteries, distributed systems and geo-localization. The benefits are important for climate change monitoring and resource savings.

In this context, we have firstly achieved a learning of technologies and several practical system realizations. We have produced a navigation software (QuickMap) allowing to interface GIS databases and tile servers similar to OpenStreetMap, taking care of sensor locations and outputs. QuickMap is also a nice frontend to produce cellular systems oriented to physical simulations.

Using the NetGen platform, we have produced a simulation framework allowing to schedule mobile moves with sensor field activities. A case study was LEO satellites visiting remote sensor systems, with investigations on two algorithms suitable for data collection and control.

Finally we have addressed the question of observation system virtualization by using an high level, process oriented virtual machine (TVM) to control the wireless link, a step forward to make the distributed and local behaviours homogeneous in terms of programming and simulation.

Many of our developments are currently involved in active projects.

This thesis was funded by a grant of Brest Metropole Oceane (BMO) and was achieved in a wireless research group at University of Brest, Lab-STICC laboratory.

Keywords: wireless sensor network, distributed systems, concurrent sequential processes, high-performance simulation, NetGen, Occam, Cuda, TVM ...

Lab-STICC

Brest Métropole

FACULTÉ DES SCIENCES & TECHNIQUES

UBO Université de Bretagne Occidentale

n° d'ordre : 2016UboXYZ

Université de Bretagne Occidentale
20 avenue Victor Le Gorgeu – C.S. 93837 – 29238 Brest cedex 3
Tél : + 33(0)2 98 01 60 00 — www.univ-brest.fr

UNIVERSITE BRETAGNE LOIRE