

INTRODUCTION

Dans une période de fortes transformations des modes de vie où les préoccupations environnementales ont une place à part entière, l'aménagement des territoires et plus particulièrement de l'habitat sont en pleine révolution. La problématique qui se pose est la suivante : comment utiliser les nouvelles technologies pour accompagner au mieux les mutations actuelles ? Les **Nouvelles Technologies de l'Information et de la Communication (NTIC)** ont connu un essor important ces dernières années, apportant une connaissance plus accrue du monde mais aussi de l'impact de l'Homme sur son environnement. A la vue des capacités présentées par ces nouvelles technologies, leur faculté de répondre aux nouvelles attentes aussi bien écologiques, économiques, de transport ou d'amélioration du cadre de vie ne sont plus à prouver. Cependant, la mise en place et l'utilisation de telles solutions, notamment au niveau de l'habitat, en sont encore à leurs débuts. Ce rapport présente une solution, de la théorie aux tests réels, permettant de connecter des capteurs hétérogènes au sein d'un réseau pour qu'ils exécutent des applications fonctionnelles. Ces applications, de natures variées, peuvent répondre à des besoins de tous les jours comme le contrôle des appareils électriques de la maison, la gestion de l'énergie ou encore la sécurité. Ce rapport est découpé en trois parties, d'abord une présentation des usages, des normes et du domaine de recherche concernés par la solution proposée, ainsi que son Cahier des Charges. Ensuite, la seconde partie présente les clés de cette solution, nommée **Environment Monitoring and Management Agents (E.M.M.A.)**. Enfin, la troisième partie apportera les résultats obtenus suite aux premiers tests effectués. La section suivante concerne donc le contexte du projet, à commencer par la notion de bâtiment intelligent.

TABLE DES MATIÈRES

I	Présentation	2
I-A	L'internet des Objets	2
I-A1	Bâtiments Intelligents	2
I-A2	Les standards IEEE et IETF	2
I-A3	Contiki OS	3
I-B	Le LACSC	3
I-B1	Le laboratoire	3
I-B2	Les réseaux de capteurs	4
I-B3	Le projet EMMA	4
I-C	Cahier Des Charges	4
I-C1	Objectifs	4
I-C2	Démarche	4
I-C3	Planning	5
II	Le Framework EMMA	6
II-A	Fonctionnement théorique	6
II-A1	Problématique	6
II-A2	Application distribuée	6
II-A3	Déploiement	8
II-B	Implémentation	8
II-B1	Hypothèses	8
II-B2	Architecture	9
II-B3	Limites et Améliorations	15
II-C	Livrables	15
III	Expérimentation	17
III-A	Protocole	17
III-A1	Objectifs	17
III-A2	Hypothèses	17
III-A3	Matériel et méthode	17
III-B	Résultats	18
III-C	Interprétation	20
III-C1	Validations	20
III-C2	Limitations	20
III-C3	Améliorations	21
Références		21

I. PRÉSENTATION

A. L'internet des Objets

1) *Bâtiments Intelligents*: Le terme de "bâtiment intelligent" est de plus en plus employé aussi bien dans le domaine du bâtiment, de l'énergie ou des nouvelles technologies. C'est en quelque sorte la réponse apportée aux préoccupations environnementales et au changement de mode de vie des pays développés en termes de consommation d'énergie et de NTIC. Prenant appui sur les nouvelles réglementations mises en place par les pouvoirs politiques, des solutions de gestion de l'énergie apparaissent. Elles ciblent avant tout la connaissance de la consommation électrique des bâtiments en aval, afin de faciliter la régulation en amont. Deux notions se dégagent de l'expression "bâtiment intelligent"¹ :

- la maison communicante individuelle, c'est-à-dire l'utilisation d'objets connectés pour communiquer sur l'état de la maison.
- le bâtiment à énergie positive, c'est-à-dire que ce dernier n'est plus consommateur mais producteur d'énergie.

Different corps de métier se regroupent donc derrière cette dénomination, expliquant dans le même temps la difficulté à apporter une solution globale et cohérente. D'autre part, le faible renouvellement du patrimoine, du fait de la longévité d'un bâtiment, pose des contraintes supplémentaires en termes de déploiement des solutions proposées. L'utilisation de nouveaux matériaux, la pose de matériel spécifique est certes efficace mais se heurte à des contraintes de coûts importantes. Une solution à moindre coût implique la ré-utilisation de ce qui existe déjà. Le nombre croissant d'appareils électroniques, ayant une durée de vie plus limitée qu'une construction, dans les foyers doit donc être vue comme une aubaine. C'est en effet la solution actuelle la plus rentable : utiliser l'électronique et l'informatique dernière génération au service du bâtiment intelligent, au travers notamment du concept de maison communicante. Dans ce domaine, des normes et standards apparaissent dans le but d'assurer une cohérence et une interopérabilité parfaite, c'est le sujet de la section suivante.

2) *Les standards IEEE et IETF*: L'Internet Engineering Task Force (IETF) est un groupe de travail ayant pour but d'améliorer le fonctionnement d'Internet, au travers de documents qui influencent les développeurs, les administrateurs ou encore les utilisateurs sur les meilleures façon d'utiliser cet outil. Ce groupe qui semble à première vue relativement éloigné de la problématique des bâtiments intelligents pose les bases de fonctionnement des réseaux de capteurs. Au travers de standards tels que (**6LowPAN**) et **Constraint Application Protocol (CoAP)**, l'IETF assure l'interopérabilité des micro-contrôleurs à tous les niveaux du modèle **Open Systems Interconnection (OSI)**. Cette interopérabilité est primordiale pour construire des applications entre produits issus de différents constructeurs. Elle rend possible la création de réseaux de capteurs et donc à terme la création de la maison communicante (voire intelligente) individuelle.

Pour bien comprendre les tenants et aboutissants de ces standards, il faut en revenir aux sources. Le développement croissant d'Internet et les avancées technologiques faites dans le domaine des micro-contrôleurs ces dernières années sont à la source du concept d'**Internet Of Things (IoT)**. C'est-à-dire la faculté de connecter au travers du réseau **Internet Protocol (IP)** de plus en plus d'objets qu'ils soient d'utilité courante ou simplement pour les loisirs. Cette perspective ouvre le champ de nombreux autres domaines beaucoup plus vastes tels que la santé, le militaire ou encore l'énergie comme il en a été question dans la section précédente. Toutes ces applications nécessitent l'existence d'objets sans fils, basse consommation, connectés à un réseau local ou à internet. Les technologies actuelles permettent déjà de telles applications. En effet, les micro-contrôleurs sont petits, consomment peu et sont suffisamment puissants pour être dotés d'une liaison radio. L'avènement de l'**(IPv6)** a ouvert un très large espace d'adressabilité². En somme, il est donc possible aujourd'hui de connecter des objets à Internet.

Cependant, l'un des principaux défis après les contraintes physiques liées à la consommation, la mémoire, la puissance de calcul et la connectivité est celui de l'interopérabilité. C'est à ce titre qu'est apparu l'ensemble de standards rédigés par l'**IETF** sous le nom de **6LowPAN**. S'appuyant sur le standard 802.15.4³ rédigé par l'**Institute of Electrical and Electronics Engineers (IEEE)**, **6LowPAN** dresse une pile complète⁴ semblable à celle de l'**IPv6** mais respectant les contraintes propres aux micro-contrôleurs citées. Un second standard rédigé par l'**IETF**, **CoAP**, donne la réplique au très célèbre **Hypertext Transfer Protocol (HTTP)**⁵ et ouvre la voie des services WEB aux micro-contrôleurs. La figure 1 illustre la relation existante entre la pile IP et la pile **6LowPAN** en prenant le modèle **OSI** comme outil de comparaison. En résumé, doté des standards (802.15.4), **6LowPAN** et **CoAP** il est possible de connecter des objets entre eux et à Internet.

1. Source : <http://www.smartgrids-cre.fr>

2. Le nombre d'adresses uniques attribuables sur le réseau.

3. Ce standard définit un protocole de communication pour les appareils à faible puissance dans des réseaux où la portée est réduite. En termes techniques, il spécifie la couche PHY et la couche MAC pour les **Low Rate Wireless Personal Area Network (LR WPAN)**.

4. En référence au modèle **OSI** qui définit des couches fonctionnelles nécessaires à la communication entre deux machines. Chaque couche ayant à s'appuyer sur la précédente pour construire une fonctionnalité plus haut-niveau.

5. **HTTP** est le protocole utilisé pour échanger du contenu sur Internet, comme un site WEB. **CoAP** est son équivalent adapté (compressé) aux micro-contrôleurs

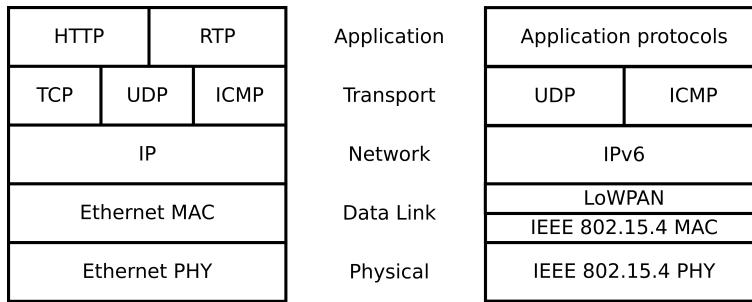


FIGURE 1. Pile IP (à gauche), Couches OSI (au centre), Pile 6LowPAN (à droite) [1]

3) *Contiki OS*: Plusieurs implémentations conformes à ces standards ont vu le jour dont une des plus connues se nomme Contiki. Contiki **Operating System (OS)**. Contiki est un système d'exploitation multi-plateformes optimisé pour micro-contrôleurs. C'est-à-dire qu'il fonctionnera aussi bien sur un micro-contrôleur 8 bits que sur un **Personnal Computer (PC)**, sans oublier l'architecture ARM⁶. Contiki propose nativement une pile **6LowPAN** complète, une gestion d'évènements et de threads. Il dispose aussi d'une console et d'un système de fichiers, voir 2. Des applications viennent se greffer au dessus du système d'exploitation, elles apportent notamment la possibilité d'utiliser certains protocoles comme **CoAP**. En définitive, Contiki **OS** est un outils adapté au développement des réseaux de capteurs, donc des objets connectés et par conséquent au développement de la maison communicante et plus généralement des bâtiments intelligents.

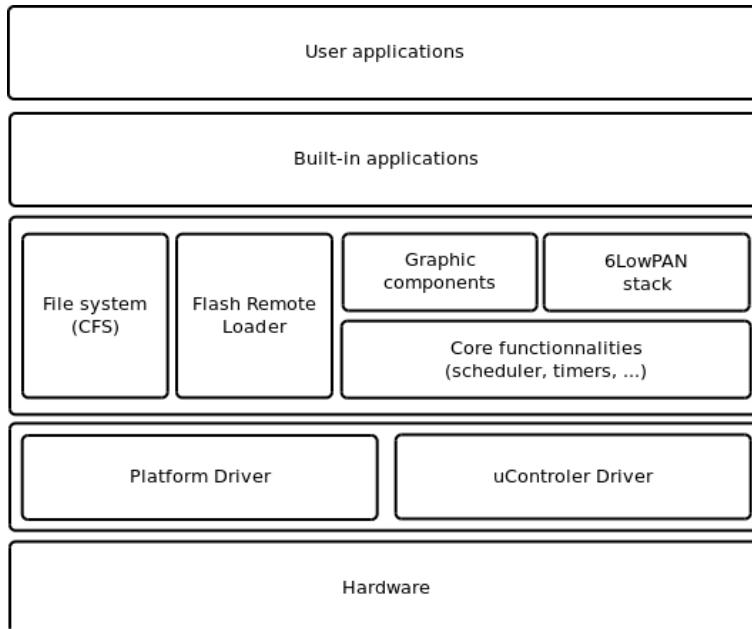


FIGURE 2. Pile Contiki.

B. Le LACSC

1) *Le laboratoire*: Le **Laboratoire d'Analyse et Contrôle des Systèmes Complexes (LACSC)** est un laboratoire de recherche affilié à l'**Ecole Centrale d'Electronique (ECE)**. Les thématiques de recherche du **LACSC** sont multiples et couvrent notamment les domaines de :

- Temps réel et systèmes embarqués
- Réseaux de capteurs
- Intelligence des systèmes complexes
- Codage et transmission de données multimédia

6. Famille de processeurs basé sur une architecture **Reduced Instruction Set Computing (RISC)**

2) Les réseaux de capteurs: Les réseaux de capteurs, partie intégrante de la recherche au **LACSC**, est en lien direct avec le sujet développé dans ce rapport. Au travers de projets menés avec les étudiants, de cours dispensés par les enseignants-rechercheurs et de publications, la connaissance des réseaux de capteurs évolue. Les problématiques inhérentes au développement d'objets connectés se précisent. Ces problématiques sont principalement de deux natures :

- Le déploiement d'applications de manière distribuée afin de limiter l'encombrement réseau.
- L'exécution d'applications, elles aussi distribuées, afin de profiter pleinement de l'architecture réseau et de contourner le problème de puissance de chaque capteur pris individuellement.

En effet, la section **I-A** présentait les outils à disposition pour construire un réseau de capteurs. Mais les problématiques de puissance de calcul, de consommation d'énergie, etc. restent entières en ce qui concerne les applications faites à partir d'un tel réseau. Dans les deux problématiques citées, un mot est récurrent : la distribution. Les réseaux de capteurs font de l'adage "diviser pour mieux régner" une réalité : l'hypothèse de départ étant d'utiliser le moins de ressources possibles, il est nécessaire de multiplier les entités de faible capacité pour construire des applications de plus haut-niveau. Partant de ce constat, il est possible de factoriser les deux problématiques que sont le déploiement et l'exécution d'applications au sein de réseaux de capteurs en une solution unique basée sur la distribution des tâches dans le réseau. C'est le but du projet **E.M.M.A.** [2] : proposer un framework de travail pour le déploiement et l'exécution d'applications distribuées sur micro-contrôleurs.

3) Le projet EMMA: Le projet **E.M.M.A.** est d'abord un logiciel, basé sur les technologies WEB, exécuté sur chaque capteur d'un réseau afin qu'ils puissent inter-agir. La théorie sous-jacente à ce logiciel est l'**Evènement-Condition-Action (ECA)** et plus précisément celle des réseaux de Pétri non limitée à des conditions binaires, voir **II-A**. Mais l'application exécutée sur chaque capteur (ou noeud) n'est pas tout, des outils sont nécessaires pour tester et valider ce fonctionnement. Il est alors possible de parler de "framework", c'est-à-dire un ensemble d'outils assurant le développement, le test, la validation et la présentation de la solution. En ce qui concerne le framework **E.M.M.A.**, les outils autres que l'application elle-même sont :

- Un émulateur de noeud pour simuler un réseau de plusieurs capteurs.
- Un navigateur WEB, doté d'un plugin gérant le protocole **CoAP**.
- Un utilitaire de chargement permettant l'envoi de paquets **CoAP** à un noeud donné du réseau simulé.

La section suivante définit avec plus de précision les fonctionnalités requises pour le framework **E.M.M.A.** ainsi que les technologies qui seront utilisées pour l'implémentation de l'application.

C. Cahier Des Charges

1) Objectifs: Le but du projet **E.M.M.A.** est à la fois de déployer et d'exécuter des applications distribuées au sein d'un réseau de capteurs. Pour valider le fonctionnement théorique, voir **II-A**, et effectuer des tests avancés sur les performances d'une telle architecture, le cahier des charges pose les objectifs suivants :

- Implémenter l'application **E.M.M.A.** sous Contiki OS pour plateforme RAVEN⁷.
- Adapter le simulateur Contiki, COOJA, pour l'émulation d'une plateforme RAVEN.
- Tester cette implémentation par le déploiement d'une application distribuée de contrôle-commande.

2) Démarche:

a) Implémenter l'application EMMA: L'implémentation d'**E.M.M.A.** est soumise à certaines règles de par le but final à atteindre. Ainsi, l'application doit utiliser le protocole **CoAP** pour faire communiquer les noeuds entre eux. L'utilisation de Contiki impose le langage de programmation (C) ainsi que le mode de communication (802.15.4, **6LowPAN**, voir **I-A**). De plus, l'application devra utiliser le format (**JSON**) pour échanger les données entre capteurs (ou noeuds). Dans tous les cas, les jalons essentiels à la réussite de cette implémentation sont :

- Etudier le fonctionnement théorique d'**E.M.M.A.**.
- Etudier le protocole **CoAP** et le format **JSON**.
- Retranscrire ces informations sous la forme d'un programme écrit en langage C et documenté.

b) Adapter le simulateur Contiki: Le micro-kernel Contiki sur lequel la solution doit être déployée dispose en effet d'outils de développement distribués avec les sources. Cependant, l'émulateur de plateforme RAVEN est en cours de développement et certaines fonctionnalités propres à cette plateforme ne sont pas disponibles à la simulation. Pour assurer un environnement de test à l'application **E.M.M.A.** développée, il est nécessaire d'ajouter ou de stabiliser les fonctionnalités manquantes. Ce travail est à effectuer en collaboration avec les développeurs de l'émulateur de la plateforme RAVEN afin de comprendre rapidement son fonctionnement. Les étapes clés du portage de l'émulateur de plateforme RAVEN pour le simulateur COOJA sont :

- Adapter le code de plateforme RAVEN pour le rendre compatible avec le protocole **Serial Line Internet Protocol (SLIP)** : cela assurera la connection entre le réseau simulé sous COOJA et le navigateur WEB de l'ordinateur.
- Faire fonctionner un exemple typique de service WEB (éprouvé sur d'autres plateformes) pour une carte RAVEN.
- Soumettre les améliorations apportées à l'émulateur de plateforme RAVEN à la communauté de développeurs Contiki.

7. Voir www.atmel.com/tools/AVRRAVEN.aspx

c) Tester l'implémentation: Ce dernier objectif permettra d'étayer les travaux de recherche effectués dans le domaine de l'**IoT** au **LACSC**. L'application **E.M.M.A.** se veut très généraliste dans son utilisation, il est donc important de cibler les éléments de test à mettre en évidence, des modifications ultérieures apparaîtront avec les usages. Les étapes clés de cette phase sont :

- Rédiger le protocole pour l'application distribuée de contrôle-commande désirée.
- Effectuer les tests.
- Analyser les résultats et modifier si possible l'implémentation, pour ensuite ré-itérer les tests dans un processus continu d'amélioration.

3) Planning: Cette section résume les dispositions prises avec le cahier des charges en un planning prévisionnel.

Objectif	Sous-objectif	Temps estimé
Prise en main et adaptation des outils de développement (1 mois)	Adapter la plateforme utilisée Exécuter un exemple e service WEB Soumettre les améliorations à la communauté Contiki	1 semaine 1 semaine 2 semaines
Implémentation d'EMMA (2 mois)	Etude théorique du programme EMMA Etude du protocole CoAP Implémentation de l'application	2 semaines 2 semaines 1 mois
Tests et déploiement (2 mois)	Rédiger le protocole de test Effectuer les tests Apporter les correctifs	2 semaines 3 semaine 3 semaines

TABLE I. PLANNING PRÉVISIONNEL

II. LE FRAMEWORK EMMA

A. Fonctionnement théorique

1) *Problématique*: Le but initial du projet est de connecter, par liaison sans fil, des capteurs hétérogènes afin de constituer une application distribuée sur le réseau ainsi formé. Le schéma ci-dessous présente un exemple d'application distribuée de type contrôle-commande.

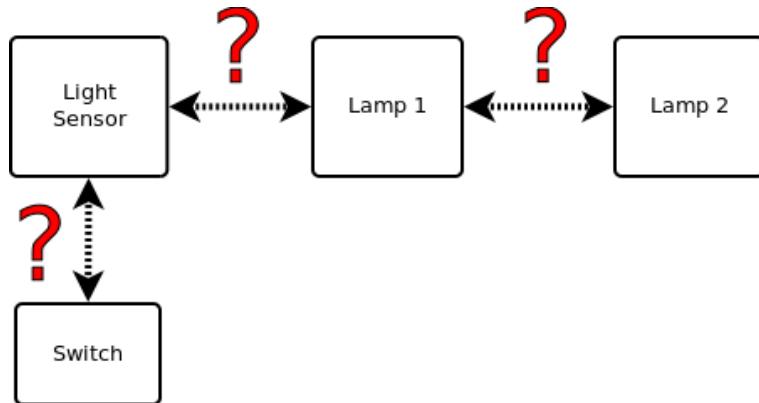


FIGURE 3. Capteurs d'une application de contrôle-commande simple

Chaque rectangle représente un noeud physique : ici, deux lampes, un capteur de luminosité ainsi qu'un élément de seuillage (type potentiomètre ou interrupteur). La problématique précédente peut être exprimée de nouveau : comment connecter, par liaison sans fil, ces noeuds hétérogènes, de manière à former une application distribuée de type contrôle-commande ?

Dans cette problématique, on retrouve des concepts présentés dans la partie précédente à savoir la notion de réseau de capteurs et celle d'interopérabilité entre noeuds hétérogènes. Il est donc essentiel de s'appuyer sur les standards et outils déjà pré-établis afin d'assurer un fonctionnement optimal et une ré-utilisabilité du travail effectué. La norme 802.15.4 pour la communication sans fils, le standard **6LowPAN** pour le réseau de capteurs formeront la base de travail pour répondre à la problématique. Cependant, il reste encore à définir comment déployer et exécuter une application distribuée sur un tel réseau. En effet, le réel défi consiste à ne pas utiliser de noeud "central" qui présente un coût supplémentaire à l'installation. Cette centralisation est aussi synonyme d'encombrements réseau dès qu'elle sera déployée à grande échelle.

Le Framework **E.M.M.A.**, proposé en [2], présente un modèle de fonctionnement d'application distribuée basé sur des **ECA**. Ce modèle permet à la fois le fonctionnement et le déploiement distribué des applications sur un réseau hétérogène. A termes, l'utilisation d'**ECA**, très proche des réseaux de Pétri, permettra de vérifier et valider la consistance du déploiement et de l'application réseau d'après la topologie réseau. Cela permettra donc de factoriser.

De la problématique initiale, nous pouvons désormais extraire trois grands axes de travail :

- Fonctionnement de l'application distribuée
- Déploiement de l'application distribuée
- Design haut-niveau de l'application distribuée

Seuls les deux premiers seront traités dans ce document, le troisième sera abordé en dernière partie afin d'assurer la transition avec les travaux futurs. Par conséquent, les deux sections suivantes traitent, de manière théorique, respectivement du fonctionnement et du déploiement d'une application distribuée selon le framework **E.M.M.A.**.

2) *Application distribuée*: Le fonctionnement d'une application distribuée telle que décrite dans [2] est celle d'un **ECA**, ou encore d'un réseau de Pétri amélioré. La démarche d'un **ECA** est simple et se définit comme suit :

- 1) Événement : Déclenche l'évaluation de la condition.
- 2) Condition : Déclenche l'action si la condition est vraie.
- 3) Action : L'action est effectuée.

Dans le cadre du framework **E.M.M.A.**, on distingue deux éléments principaux :

- Les ressources qui sont des variables (telle une quantité physique discrète comme la luminosité).
- Les agents qui définissent une Condition et l'Action qui y est associée.

Algorithm 1: Algorithme de fonctionnement d'E.M.M.A.

Données: Agents, Ressources

début

 | **si** Modification d'un agent ou d'une ressource **alors** /* Evènement */

 | **pour chaque** Agent faire

 | **si** Condition de l'agent est vraie **alors** /* Condition */

 | Action de l'agent. /* Action */

fin

L'analogie entre un ECA et le framework E.M.M.A. se présente donc ainsi :

Cet algorithme peut aussi être présenté de manière schématique, figure 4, en reprenant le même exemple que précédemment. Noter l'existence d'un agent d'environnement (AE) qui représente l'effet de la lumière sur le capteur de luminosité.

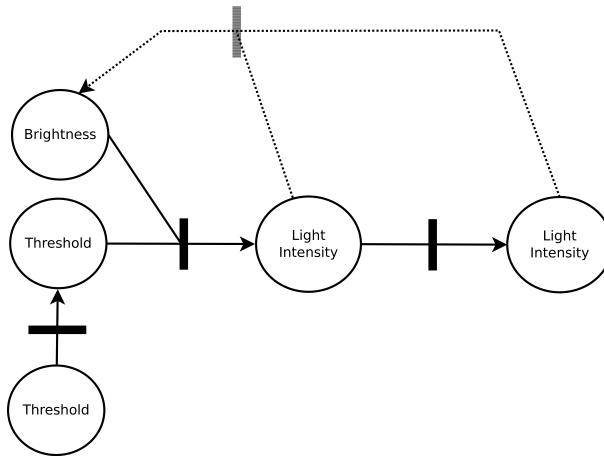


FIGURE 4. Schéma de fonctionnement d'une application de contrôle-commande simple

Cependant, l'application ainsi présentée n'est pas encore "distribuée" sur les entités que sont les deux lampes, le capteur de luminosité et l'interrupteur. Le schéma 5 présente donc la même application de contrôle-commande en explicitant la location physique de chaque ressource et de chaque agent sur un noeud donné.

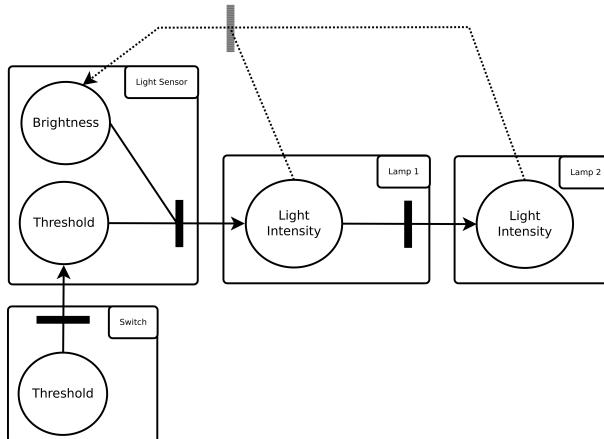


FIGURE 5. Schéma de fonctionnement d'une application de contrôle-commande simple distribuée

3) **Déploiement:** Le principe de fonctionnement d'**E.M.M.A.** peut être utilisé pour déployer une application sur un réseau donné. Cette fois, ce sont des agents (de déploiement) qui déposent d'autres agents (d'application) de proche en proche. La figure 6 présente ce mécanisme.

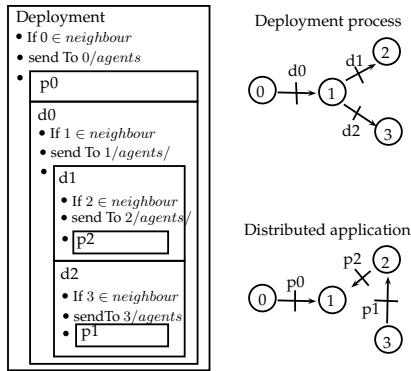


FIGURE 6. Déploiement d'une application distribuée par agents imbriqués, [2].

Les agents sont imbriqués à la manière de poupees-russes. Chaque noeud, à la réception d'un agent de déploiement, l'exécute ce qui a pour effet de déployer d'autres agents (de déploiement et d'application) sur ses voisins. Cette méthode a trois avantages principaux :

- Utiliser les mêmes méthodes que pour le fonctionnement d'une application : le déploiement est vu comme une application particulière.
- Diminuer la congestion réseau : le déploiement est lui aussi distribué, il s'effectue de proche en proche et non depuis un point central.
- Effectuer l'installation et les mises à jour facilement : celles-ci peuvent être effectuées depuis n'importe quel terminal relié au réseau, qu'il soit fixe ou mobile comme un SmartPhone.

La figure 7 reprend l'exemple 5 et présente son mode de déploiement :

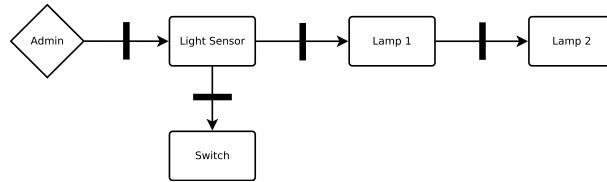


FIGURE 7. Déploiement de l'application de contrôle-commande simple.

La partie suivante présente l'implémentation réelle d'**E.M.M.A.**. Le fonctionnement théorique d'**E.M.M.A.**, présenté précédemment, s'appuie sur le document [2] qui en reste la référence absolue.

B. Implémentation

L'implémentation d'**E.M.M.A.** s'articule autour de trois parties principales :

- Les hypothèses présentent les choix d'outils effectués issus du cahier des charges et du fonctionnement théorique de l'application.
- L'architecture passe en revue l'organisation détaillée des trois éléments du framework : Noeud, Simulateur et utilitaire de déploiement.
- Les limites et améliorations dressent le bilan du projet, ce qui fonctionne et les idées d'améliorations à apporter.

1) **Hypothèses:** Reprenons le projet dans ses fondements (section II-A) : "Connecter, par liaison sans fil, des capteurs hétérogènes afin de constituer une application distribuée sur le réseau ainsi formé."

Le temps où chacun créait un protocole "maison" pour faire communiquer ses micro-contrôleurs entre eux est (presque) révolu. En effet, la section I-A rappelle les différentes normes et standards en la matière. La philosophie du projet est de proposer un environnement de travail (framework) réutilisable. Les hypothèses en termes de pile de communication sont donc :

- Norme **802.15.4** pour la communication sans fil.

– 6LowPAN en adaptation de l'**IPv6**, [1].

– CoAP version 8, le pendant du protocole **HTTP** pour les réseaux de capteurs.

Le choix du protocole **CoAP** est dû à de multiples raisons : d'une part il est dédié aux services WEB sur micro-contrôleur, d'autre part le Framework **E.M.M.A.** utilise lui-même le paradigme (**Client-Serveur**) sous la forme de services WEB car doté de ressources identifiables. Enfin, des implémentations de ce protocole et outils associés existent déjà.

Le cahier des charges impose la plateforme de développement ainsi que le micro-kernel à utiliser :

- Plateforme : Carte électronique AVR-Raven.
- **OS** : Contiki OS version 2.6.
- Langage : Langage C.

Ces choix ont plusieurs raisons d'être. Tout d'abord la plateforme AVR-Raven est constituée d'un micro-contrôleur ATMEGA128rfa1 qui intègre l'électronique nécessaire à la communication sans fil de la plateforme. D'autre part, le micro-contrôleur ATMEGA128rfa1 est compatible avec l'**OS** Contiki. Ce dernier apportera la pile réseau fonctionnelle pour les réseaux de capteurs : **6LowPAN**. Le langage utilisé est déduit de la plateforme et de l'**OS** utilisé.

Concernant le formatage des données transmises (au niveau applicatif), nous utiliserons du **JSON** car sa syntaxe propose un double avantage :

- Traitement informatique simplifié.
- Lisible par l'Homme.

Pour résumer, la figure 8 présente les hypothèses d'implémentation d'**E.M.M.A.** sous la forme d'une pile.

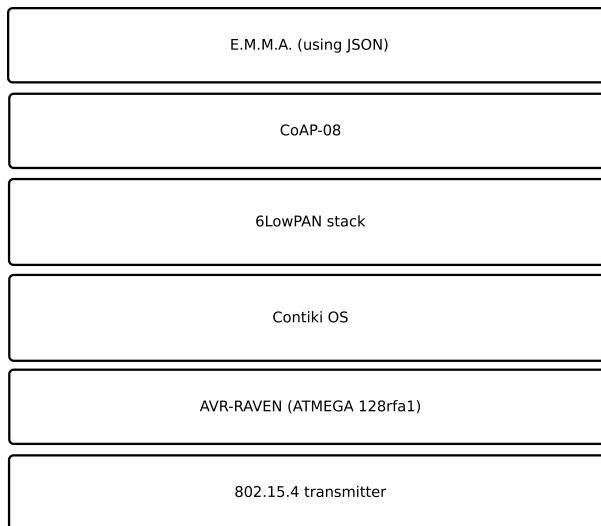


FIGURE 8. Pile architecture des contraintes du Cahier Des Charges

2) *Architecture*: Le framework **E.M.M.A.** se compose actuellement de trois éléments :

- EMMA-node : C'est le programme exécuté par chaque micro-contrôleur (ou noeud).
- EMMA-simulator : C'est le programme utilisé pour simuler le réseau de noeuds.
- EMMA-loader : C'est un logiciel permettant le déploiement initial d'un agent ou d'une ressource sur un noeud.

Chaque partie de l'architecture sera découpée en trois paragraphes reprenant respectivement le Noeud, le Simulateur et l'Utilitaire de déploiement, figure 9.

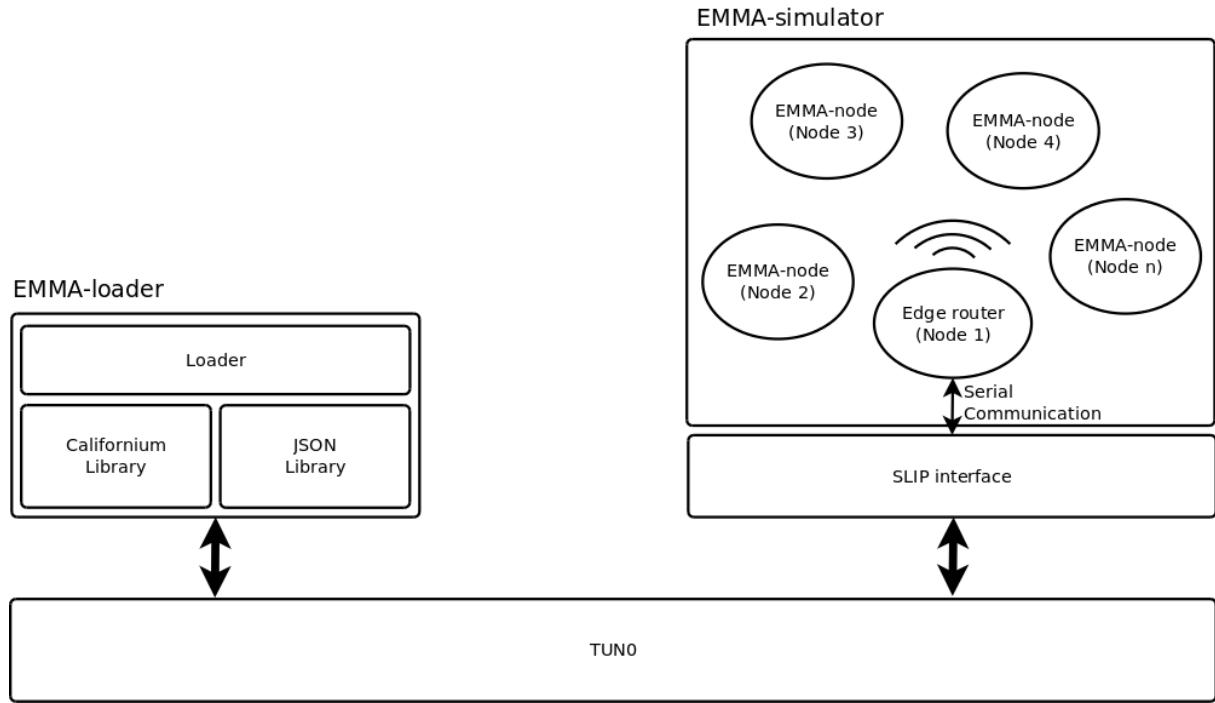


FIGURE 9. Architecture globale du Framework E.M.M.A..

Node: Le programme présent sur chaque noeud "EMMA", bien que codé en langage C, présente des caractéristiques propres aux langages orientés objets comme le C++. Ce type d'implémentation se retrouve sous le nom générique d'**Object Oriented C (OOC)**. L'architecture du programme est de ce fait présentée sous la forme d'un diagramme de classe **Unified Modeling Language (UML)**. Pour une plus grande clarté dans la relecture et la compréhension du programme, chaque classe est reliée à un fichier source et un fichier d'en-tête (extension .c et .h respectivement).

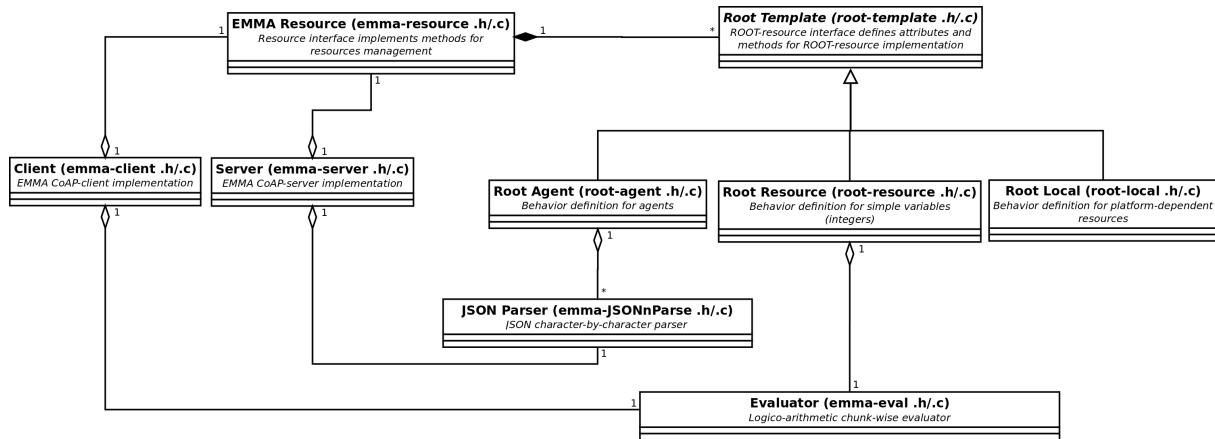


FIGURE 10. Diagramme de classe EMMA-node réduit.

Noter l'élément clé de cette architecture : l'héritage de la classe abstraite "ROOT Template". Ceci permet de généraliser le concept de ressource (au sens du protocole **CoAP**). Ainsi, un Agent, une Ressource (au sens d'**E.M.M.A.**), ou une donnée capteur peuvent être traitées de la même façon dans le reste de l'application. Pour bien comprendre la classe "ROOT Template", il faut la considérer comme une description (au travers de fonctions à implémenter) d'un type de ressources données. La racine (ROOT) d'une ressource donnée doit de fait définir le comportement :

- De gestion dynamique de la mémoire (allocation, libération, mutex).
- D'accès à la ressource (ouverture, lecture, écriture, fermeture).

- De traitement des données (analyse par expressions régulières).

La classe "Resource" se définit comme gestionnaire d'un ensemble de racines (ROOT). Plus précisément, elle met à disposition les mêmes fonctions que celles proposées par chaque racine en normalisant l'accès aux ressources. C'est-à-dire qu'elle encapsule l'appel aux fonctions spécifiques de chaque racine en identifiant chaque ressource par son **Unique Resource Identifier (URI)**. Ainsi, toutes les fonctions de cette classe prennent en paramètre un **URI** pour identifier la ressource à traiter et retrouver son fonctionnement spécifique en identifiant sa racine. Ce qu'il faut retenir de la classe "Resource" est l'organisation interne de l'ensemble des ressources : chaque racine correspond au départ d'un arbre binaire pour toutes les ressources qui y sont associées. De cette façon, il est possible de traiter avec une grande efficacité les ressources basées sur des **URI** (comme c'est le cas pour le protocole **CoAP**). En définitive, la classe "Resource" implémente un système de fichiers complet pour chaque racine donnée à la compilation du programme. La figure 11 illustre cette organisation mêlant système de fichiers, arbres binaires et polymorphisme.

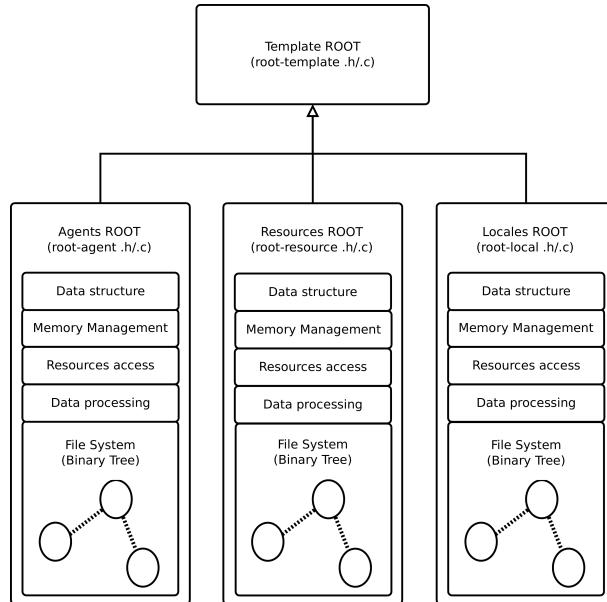


FIGURE 11. Racines, Système de fichiers et Polymorphisme.

La figure 12 présente l'organisation mémoire du système de fichiers. Les flèches symbolisent des pointeurs, chaque cercle est une structure de base de l'arbre binaire. Selon que l'on choisisse le pointeur de droite ou de gauche, on descend dans l'arborescence (à la manière de sous dossiers imbriqués) ou l'on parcours le contenu d'un dossier. Ceci est symbolisé par les flèches en pointillés. La figure 13 illustre cette organisation par un exemple concret en proposant deux autres représentations équivalentes et communément admises pour un système de fichier. Il est à noter que la troisième de ces représentations correspond à celle utilisée par CoAP.

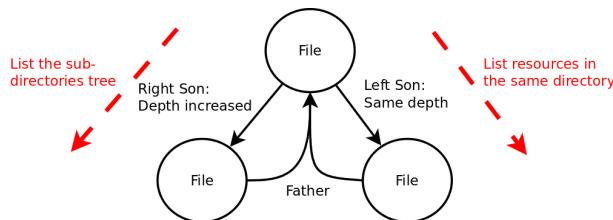


FIGURE 12. Schéma mémoire d'un arbre binaire représentant un système de fichiers.

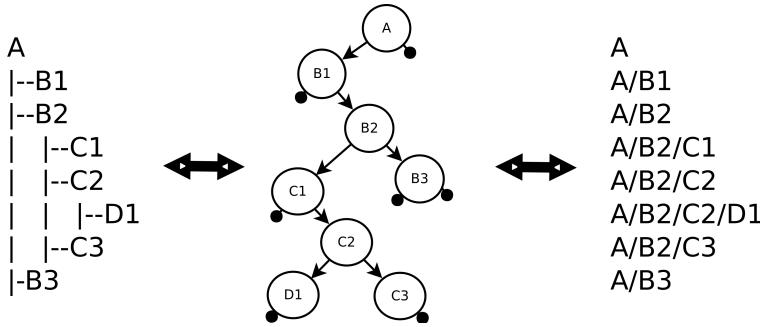


FIGURE 13. Multiples représentations d'un système de fichier.

Enfin, l'algorithme 2 explicite le parcours d'un arbre binaire pour la recherche et le traitement de données ainsi organisées. Ce parcours est non récursif du fait des contraintes mémoires dues à l'utilisation de micro-contrôleurs.

Algorithm 2: Parcours infixé, non récursif, d'un arbre binaire.

Data: previous(NULL), current(TreeRoot), next(NULL)

```

begin
    while current ≠ NULL do
        if previous = current(Father) then
            previous ← current;
            next ← current(LeftSon);
        if (previous = NULL) ∨ (previous = current(LeftSon)) then
            begin
                /* Execute in-order processing */
            end
            previous ← current;
            next ← current(RightSon);
        if (previous = NULL) ∨ (previous = current(RightSon)) then
            previous ← current;
            next ← current(Father);
    end

```

Dans cette architecture, les classes "JSON Parser" et "Evaluator" permettent d'assurer certains traitements lourds et récurrents mais ne sont pas directement reliées au fonctionnement d'E.M.M.A. tel qu'il a été décrit dans la section précédente.

La classe "JSON Parser" permet l'analyse d'une chaîne de caractère, supposée au format **JSON**, caractère par caractère. Elle est donc très intéressante pour analyser des blocs de données entrant sans jamais voir la totalité du message. Son fonctionnement est simple : pour chaque caractère entré, la classe retourne son type (au sens **JSON**, voir la liste suivante). Etant donné que le **JSON** ne limite pas l'imbrication des expressions, l'utilisateur doit définir une "profondeur" de parcours. Les types **JSON** sont définis à l'adresse www.json.org, le tableau suivant en dresse un liste non-exhaustive pour le lecteur :

- Chaîne de caractères (entre guillemets "")
- Objet (entre accolades {})
- Tableau (entre crochets [])

La figure 14 est une illustration de l'utilisation du format **JSON** utilisé pour décrire un agent E.M.M.A. La profondeur de lecture correspond au nombre d'objets, de chaînes de caractères ou de tableaux imbriqués que l'on souhaite observer. Ce qui n'est pas observé est alors vu comme du contenu brut, sans préjuger de son type. La figure 15 donne le résultat de l'analyseur **JSON** appliquée à la figure 14 pour différentes profondeurs.

```
{
    "NAME":"Agent1",
    "CONDITION":"Brightness<8",
    "TARGET":["Address1", "Address2"],
    "ACTION":["Luminosity+1", {"<Nested Agent>"}]
}
```

FIGURE 14. Agent écrit au format **JSON**.

Depth=0	Depth=1	Depth=2
-START OF OBJECT	-START OF OBJECT	-START OF OBJECT
-CONTENT (118)	-START OF STRING	-START OF STRING
-END OF OBJECT	-CONTENT (4)	-CONTENT (4)
	-END OF STRING	-END OF STRING
	-PAIR SEPARATOR	-PAIR SEPARATOR
	-START OF STRING	-START OF STRING
	-CONTENT (6)	-CONTENT (6)
	-END OF STRING	-END OF STRING
	-PAIR SEPARATOR	-PAIR SEPARATOR
	-START OF STRING	-START OF STRING
	-CONTENT (12)	-CONTENT (12)
	-END OF STRING	-END OF STRING
	-PAIR SEPARATOR	-PAIR SEPARATOR
	-SEPARATOR	-SEPARATOR
	-START OF STRING	-START OF STRING
	-CONTENT (9)	-CONTENT (9)
	-END OF STRING	-END OF STRING
	-PAIR SEPARATOR	-PAIR SEPARATOR
	-START OF STRING	-START OF STRING
	-CONTENT (6)	-CONTENT (6)
	-END OF STRING	-END OF STRING
	-PAIR SEPARATOR	-PAIR SEPARATOR
	-START OF ARRAY	-START OF ARRAY
	-CONTENT (21)	-START OF STRING
	-END OF ARRAY	-CONTENT (8)
	-SEPARATOR	-END OF STRING
	-START OF STRING	-SEPARATOR
	-CONTENT (6)	-START OF STRING
	-END OF STRING	-CONTENT (8)
	-PAIR SEPARATOR	-END OF STRING
	-START OF ARRAY	-END OF ARRAY
	-CONTENT (31)	-SEPARATOR
	-END OF ARRAY	-START OF STRING
-END OF OBJECT		-CONTENT (6)
		-END OF STRING
		-PAIR SEPARATOR
		-START OF ARRAY
		-START OF STRING
		-CONTENT (12)
		-END OF STRING
		-SEPARATOR
		-START OF OBJECT
		-CONTENT (14)
		-END OF OBJECT
		-END OF ARRAY
		-END OF OBJECT

FIGURE 15. Analyse d'un agent, figure 14, pour différentes profondeur de l'analyseur (0, 1 et 2).

De la même façon, la classe "Evaluator" propose une évaluation "par morceaux" d'expressions logico-arithmétiques. Ainsi, lorsque les expressions logiques sont découpées pour des raisons de taille de paquets sur le réseau, il n'est pas nécessaire de maintenir l'ensemble de l'expression en mémoire pour pouvoir l'évaluer. Pour un noeud **E.M.M.A.**, une expression logico-arithmétique est écrite entre guillemets. Les opérateurs supportés sont décrits ci-après :

- || : OU LOGIQUE
- && : ET LOGIQUE
- === : EGALITE
- > : STRICTEMENT SUPERIEUR
- < : STRICTEMENT INFERIEUR
- + : ADDITION
- - : SOUSTRACTION
- * : MULTIPLICATION
- / : DIVISION
- ! : NEGATION

Les priorités des opérateurs sont les mêmes que dans le langage C (priorité croissante pour la liste précédente). L'évaluateur accepte l'utilisation de parenthèses '(' et ')' si besoin est de modifier les priorités établies. Enfin, il est possible de faire référence à une ressource locale en écrivant son **URI**. La valeur de la ressource sera alors chargée pour effectuer le calcul. Noter cependant que l'**URI** devra s'écrire en utilisant le caractère '#' comme séparateur et non pas le '/'. En effet, celui-ci est réservé à la division. Exemple : "1+R#brightness<5" est vraie (vaut 1) si "un plus la valeur de la ressource locale R/brightness est inférieure strictement à cinq".

Simulator: Le simulateur utilisé pour tester un réseau de noeuds **E.M.M.A.** est le simulateur officiel de Contiki OS, nommé COOJA. Ce programme, codé en Java, est en fait bien plus qu'un simulateur. Pour être exact, c'est un logiciel permettant de :

- Emuler et/ou simuler des plateformes réelles.
- Simuler un réseau sans fils de capteurs 802.15.4.
- Accéder directement à la mémoire d'un noeud, à sa liaison série, etc.
- Se connecter, via une interface **SLIP**, au réseau local du **PC** hôte.

Ce simulateur a été écrit par les développeurs de Contiki. C'est donc avec eux qu'il peut s'avérer nécessaire d'entrer en contact. En effet, le Cahier des Charges spécifie l'utilisation de la plateforme AVR-RAVEN. Or, l'émulateur COOJA de cette plateforme est encore au stade de développement et certaines des fonctionnalités ne sont pas encore au point. Une partie du travail de ce stage a donc consisté à la stabilisation et l'ajout de fonctionnalités au simulateur COOJA pour la plateforme AVR-RAVEN. Une manière rapide pour entrer directement en contact avec la communauté de développeurs de Contiki est d'utiliser le Gmane. C'est quasiment indispensable si l'on souhaite modifier le simulateur COOJA car il n'est que très peu documenté.

L'ensemble des modifications apportées au simulateur durant le stage ont été factorisées dans des Patchs, voir section II-C. Les deux points qui suivent résument ces modifications :

- Activation des deux liaisons séries de la plateforme AVR-RAVEN, ajout de la fonctionnalité **SLIP** sur l'une d'entre-elles et sortie **DEBUG** sur l'autre.
- Adaptation de l'interface (graphique et architecturale) du simulateur pour prendre en compte les deux liaisons série (une seule reconnue par défaut).

Loader: Comme le montre le schéma d'architecture 9, il est possible d'accéder au réseau simulé dans COOJA par le biais d'un tunnel et d'une interface **SLIP**. Ainsi, toute application réseau présente sur le **PC** peut communiquer avec les noeuds émulés dans (0). Cela permet notamment de tester le fonctionnement du serveur présent sur chaque noeud **E.M.M.A.** en utilisant n'importe quel navigateur WEB pour faire des requêtes **CoAP**. Moyennant bien sûr de disposer du plugin nécessaire à l'émission et à la réception de telles requêtes. C'est le cas de Firefox et de son plugin Copper. Le document draft-ietf-coap-08 sert de référence pour la compréhension et l'utilisation de ce protocole.

Le plugin Copper offre une interface Homme-Machine simple pour appréhender graphiquement le réseau de noeuds simulé dans COOJA. Cependant, il ne permet pas la gestion de la fragmentation des paquets à l'envoi. Par conséquent, l'envoi d'un agent **E.M.M.A.** peut rapidement devenir laborieux lorsque l'utilisateur doit lui-même découper la chaîne de caractères qu'il enverra ensuite par blocs de taille fixe. C'est de ce constat qu'est né le besoin d'un utilitaire de chargement d'agents et de ressources **E.M.M.A.**, noté EMMA-loader.

Cet utilitaire, codé en Java, s'appuie sur deux bibliothèques :

- Librairie **Californium**
- Librairie **JSON**

La librairie **Californium** permet l'envoi et la réception de requêtes **CoAP** sur le réseau, elle gère la version 13 du protocole, voir draft-ietf-coap-13. La version **CoAP** la plus à jour implémentée sous Contiki est la version 8. Une adaptation a donc été nécessaire pour utiliser cette librairie avec le réseau simulé sous .

La librairie **JSON** propose des classes de traitement des chaînes de caractère au format **JSON**. Elle n'a nécessité aucune modification, sa documentation est en ligne.

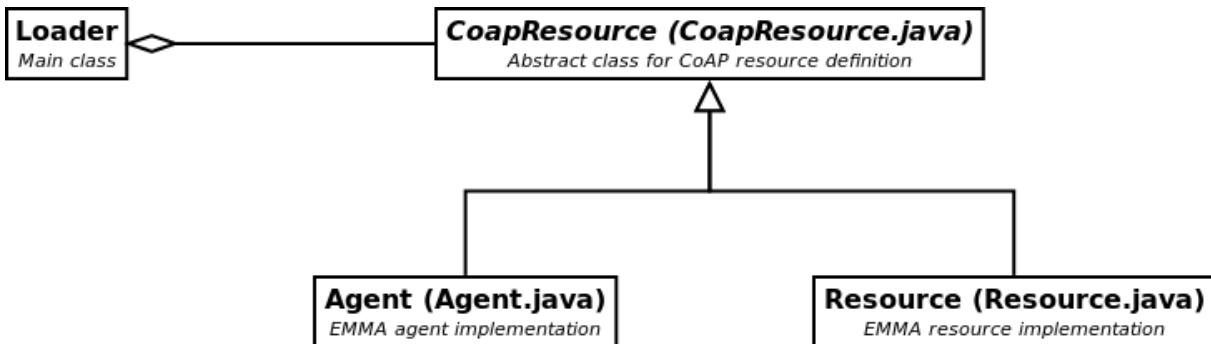


FIGURE 16. Diagramme de classe de l'utilitaire de chargement EMMA-loader.

La figure 16 est l'architecture de l'utilitaire de chargement d'agents à proprement parler. Comme pour les noeuds 10, on retrouve la généralisation du concept de ressource. La section II-C décrit l'installation de l'environnement de développement pour utiliser EMMA-loader. Quant à l'utilisation pratique, EMMA-loader est en passe d'être proposé comme un plugin au simulateur , disposant donc d'une interface graphique intuitive. Concernant l'écriture d'un agent (ou d'une ressource) au format JSON, voir II-B2 et 17, et compatible avec l'utilitaire de chargement, voici quelques règles à suivre :

- Supprimer les espaces, tabulations et sauts de lignes.
- Encapsuler l'agent (ou la ressource) dans un objet, avec comme champs obligatoires : Nom("NAME"), Condition("PRE"), Action("POST" et "TARGET")
- La clé "NAME" prend pour valeur n'importe quelle chaîne de caractères.
- La clé "PRE" prend pour valeur une expression logico-arithmétique, voir II-B2.
- La clé "POST" prend pour valeur un tableau, composé de chaînes de caractères ou d'objets séparés par un virgule.

- La clé "TARGET" prend pour valeur un tableau d'adresses **IPv6** séparées par des virgules.
- Les tableaux des clés "POST" et "TARGET" doivent contenir le même nombre d'éléments.

```
{
  "NAME":"Agent1",
  "PRE":"1",
  "POST":["R#test+1","R#test+10"],
  "TARGET":["[aaaa::200:2:2:202]:5683/R/test","[aaaa::200:3:3:303]:5683/R/test"]
}
```

FIGURE 17. Agent écrit au format **JSON** pour une utilisation avec EMMA-loader.N.B. : La première règle concernant les espaces et tabulation n'a pas été respectée par soucis de clarté.

3) Limites et Améliorations: Comme précédemment, cette section est découpée en trois paragraphes : EMMA-node, EMMA-simulator et EMMA-loader. Les limitations exposées dans cette partie peuvent être considérées comme des améliorations futures car elles ne sont pas limitées par des phénomènes physiques. Certaines autres tiennent plus de l'optimisation que d'un ajout de fonctionnalité. Enfin, cette partie permet de contourner rapidement les pièges qui ne sont pas toujours évidents à la seule lecture de l'architecture.

Node: Les noeuds **E.M.M.A.**, bien que fonctionnels, présentent des parties de code en développement. Ceci pour des raisons d'optimisation principalement. La liste suivante est complète mais ne se veut pas exhaustive :

- Implémentation d'un cache pour optimiser les accès récurrents aux ressources.
- Renforcement de la fonction de recherche de ressources par **URI** (notamment concernant les multiples '/' ou autres cas particuliers).
- Optimisation de la profondeur d'appel des fonctions dans chaque classe (en assurant un nombre maximum d'appels successifs pour toute l'application **E.M.M.A.**).
- Multiplication des codes erreur émis par le serveur pour une compréhension plus ciblée.
- Passage en notation post-fixée pour l'évaluation d'expressions logico-arithmétiques. Ceci afin de diminuer la taille maximale de la pile d'entrée et de diminuer le temps de calcul.
- Traitement des nombres négatifs et flottants par l'évaluateur logico-arithmétique.
- Généralisation du traitement des chaînes de caractères par l'utilisation d'expressions régulières : **Regular Expression (RE-GEX)**

Simulator: Le simulateur , toujours en développement par la communauté Contiki, présente certains bugs identifiés qui impactent fortement la simulation de noeuds **E.M.M.A.** :

- Mauvaise synchronisation de la liaison série qui implique des erreurs d'ordonnancement dans le simulateur.
- Mauvais comportement de la fonction STRCMP, définie dans le standard du langage C.

Le problème de synchronisation peut être réglé en modifiant l'intervalle de temps utilisé pour ordonner les paquets en provenance de la liaison série. Cependant, l'intervalle nécessaire peut varier en fonction du code compilé. il n'est donc pas possible pour l'heure de fixer un nombre dans tous les cas de figure.

Le livrable (**II-C**) EMMA-node distribue un correctif de la fonction STRCMP dans le cadre de son utilisation avec le système de fichiers Contiki : CFS. Autrement, il est conseillé d'utiliser la fonction STRNCMP en remplacement, couplée à la fonction STRLEN. Dans tous les cas, ce problème ne concerne pas le déploiement d'applications sur noeuds réels mais uniquement l'utilisation du code dans le simulateur COOJA.

- Loader:* L'utilitaire de chargement d'agents et de ressources (EMMA-loader) pourrait bénéficier des améliorations suivantes :
- Intégration de son interface dans , en tant que plugin.
 - Ajout d'un interface de validation du texte au format **JSON** inséré.
 - Prise en charge de l'envoi de multiples agents ou ressources en une seule fois.
 - Gestion des différents codes retour du protocole **CoAP** pour un affichage utilisateur.

C. Livrables

Cette section décrit la récupération, l'installation et l'utilisation des codes sources du framework **E.M.M.A.**. Les pré-requis sont :

- Machine Virtuelle (), ou **Java Runtime Machine (JVM)** version 6 ou ultérieure.
- Le logiciel ANT pour la compilation .

Les sources sont disponibles sur le dépôt GIT 'git.duhart-clement.fr :/var/cache/emma'. Une fois le fichier 'install.sh' récupéré, les actions à effectuer sont :

- 1) Exécuter 'install.sh' avec en premier paramètre le dossier d'installation choisi.
- 2) Suivre les instructions de l'installateur :
 - Saisir le nom d'utilisateur correspondant au GIT.
 - Saisir le mot de passe (potentiellement à plusieurs reprises).

La figure 18 décrit le contenu des dossiers 'emma-node' et 'emma-loader' créés après l'installation. La figure 19 décrit le contenu du dossier 'emma-cooja'. Pour démarrer le simulateur, exécuter le fichier 'start.sh'.

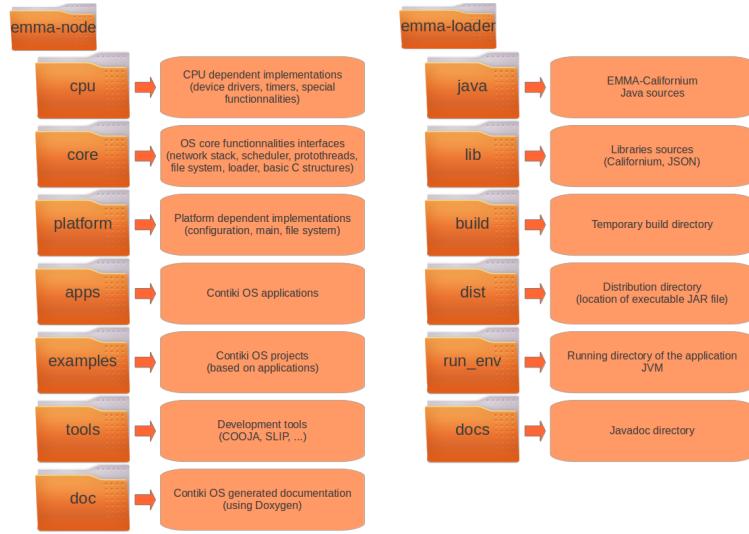


FIGURE 18. Description du contenu des dossiers 'emma-node' et 'emma-loader'.

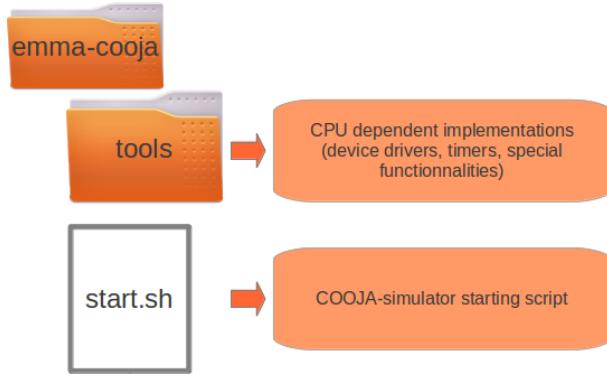


FIGURE 19. Description du contenu du dossier 'emma-cooja'.

III. EXPÉRIMENTATION

Cette section présente les tests unitaires effectués sur le framework **E.M.M.A.** afin de valider le Cahier des Charges. La première partie énonce les différents points à valider ainsi que le protocole utilisé pour effectuer les tests. La seconde partie présente les résultats sous la forme de captures d'écran ou de commandes en console. Enfin, la dernière partie analyse ces résultats pour en dégager les limitations et les améliorations possibles.

A. Protocole

1) *Objectifs:* L'objectif à atteindre avec les tests unitaires du framework **E.M.M.A.** est la validation ou invalidation des points suivants :

- 1) Connection, au travers d'une interface **SLIP**, du réseau simulé dans COOJA avec le réseau virtuel du **PC**.
- 2) Fonctionnement de requêtes **CoAP** effectuées depuis un navigateur WEB vers un noeud **E.M.M.A.** simulé.
- 3) Déploiement d'un agent sur un noeud **E.M.M.A.** depuis l'utilitaire de chargement d'agents.
- 4) Exécution d'un agent sur un noeud **E.M.M.A.**.

Ces quatre points regroupent les éléments essentiels du framework **E.M.M.A.**, ce sont les briques de base pour construire des applications complexes et propres à des besoins spécifiques.

2) *Hypothèses:* Les tests unitaires sont effectués avec le réseau le plus simple possible. Celui-ci sera donc composé d'un noeud **E.M.M.A.** et d'un autre noeud qui servira de passerelle avec le réseau virtuel du **PC**. Le schéma 20 symbolise le réseau qui va être utilisé pour les tests unitaires. La reproduction des tests nécessite l'installation du framework **E.M.M.A.**, celle-ci est décrite dans la section **II-C**.

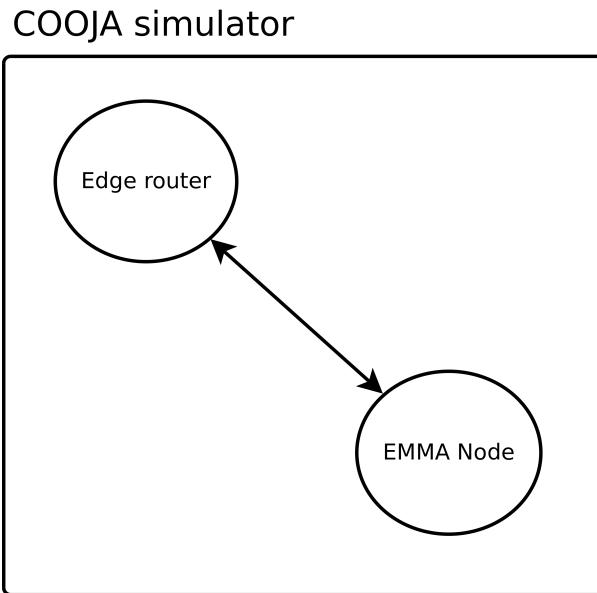


FIGURE 20. Réseau simulé utilisé pour les tests unitaires.

3) *Matériel et méthode:* Le matériel utilisé pour les tests est décrit dans le tableau II.

Matériel	Spécifications
PC hôte	OS : Ubuntu version 12.04
Java Runtime Machine	Version 7
Outil de compilation JAVA ANT	Version 1.8.2
Framework E.M.M.A. installé	Voir section II-C : Livrables
Navigateur WEB Firefox	Version 15.0
Plugin Firefox COPPER	Installé avec Firefox

TABLE II. MATÉRIEL UTILISÉ POUR LES EXPÉRIENCES.

Action	Commande associée
Démarrer le simulateur COOJA	Exécuter le fichier 'start.sh' présent dans l'installation du framework EMMA
Ouvrir la simulation	Parcourir l'installation EMMA : simulations tests.csc
Démarrer la connection SLIP	Cliquer sur le bouton START dans la fenêtre "Serial Socket (SERVER)"
Entrer le mot de passe ROOT si nécessaire	TODO
Démarrer la simulation	Bouton START dans la fenêtre "Simulation Control"

TABLE III. PROTOCOLE D'ACTIVATION DE LA CONNECTION SLIP.

Action	Commande associée
Activer la connection SLIP	Voir tableau III
Démarer la simulation	Bouton START dans la fenêtre "Simulation Control"
Ouvrir Firefox	TODO
Entrer l'adresse du noeud E.M.M.A.	Par défaut :
Découvrir les ressources du noeud	Cliquer sur DISCOVER
Créer une première ressource	<ul style="list-style-type: none"> - Compléter l'adresse du noeud avec l'URI choisie (rattaché à la racine R) - Cliquer sur POST
Créer une seconde ressource	<ul style="list-style-type: none"> - Compléter l'adresse du noeud avec l'URI choisie (rattaché à la racine R) - Dans l'onglet "Outgoing", écrire une valeur initiale pour la ressource - Cliquer sur PUT
Découvrir les ressources créées	Cliquer sur DISCOVER
Récupérer le contenu des ressources créées	Cliquer sur GET pour chacune des deux ressources
Modifier la valeur de la première ressource	<ul style="list-style-type: none"> - Dans l'onglet "Outgoing", écrire une nouvelle valeur pour la ressource - Cliquer sur PUT
Récupérer le contenu pour vérifier	Cliquer sur GET
Supprimer les ressources créées	Cliquer sur DELETE pour chacune des deux ressources

TABLE IV. PROTOCOLE DE TEST DES REQUÊTES COAP DEPUIS FIREFOX VERS LE RÉSEAU SIMULÉ.

Action	Commande associée
Activer la connection SLIP	Voir tableau III
Démarer la simulation	Bouton START dans la fenêtre "Simulation Control"
Ouvrir l'utilitaire de chargement d'agents	Tools, EMMA Loader
Charger l'agent	Parcourir l'installation EMMA, emma-agent-loader, run-env, LOAD.txt
Rafraîchir la liste des noeuds	Bouton Refresh
Déployer l'agent	Bouton Launch
Créer une ressource RLoop de valeur initiale 10	<ul style="list-style-type: none"> - Ouvrir Firefox, rentrer l'adresse du noeud dans la barre d'adresses. - Dans l'onglet "Outgoing" de COPPER, écrire une nouvelle valeur pour la ressource - Cliquer sur PUT, dans COPPER
Récupérer la valeur de la ressource	Bouton GET, dans COPPER

TABLE V. PROTOCOLE D'EXÉCUTION D'UN AGENT SUR UN NOEUD E.M.M.A..

Action	Commande associée
Activer la connection SLIP	Voir tableau III
Ajouter un noeud E.M.M.A.	Motes, Add mote, emma mote
Démarer la simulation	Bouton START dans la fenêtre "Simulation Control"
Ouvrir l'utilitaire de chargement d'agents	Tools, EMMA Loader
Charger un agent de déploiement	Parcourir l'installation EMMA, emma-agent-loader, run-env, LOAD2.txt
Envoyer l'agent sur le noeud 2	Bouton Launch
Créer une ressource r,start sur le noeud 2, de valeur initiale 1	<ul style="list-style-type: none"> - Ouvrir Firefox - Entrer l'URI complète de la ressource dans la barre d'adresses. - Dans l'onglet "Outgoing", écrire le chiffre 1 - Cliquer sur PUT
Découvrir l'agent créé sur le noeud 3	<ul style="list-style-type: none"> - Entrer l'URI complète de la ressource dans la barre d'adresses. - Cliquer sur DISCOVER - Sélectionner l'agent A.deployed dans la liste des ressources du noeud - Cliquer sur GET

TABLE VI. PROTOCOLE DE DÉPLOIEMENT D'UN AGENT IMBRIQUÉ ENTRE DEUX NOEUDS E.M.M.A..

B. Résultats

a) *Protocole 1:* Le protocole 1, voir tableau III, permet de vérifier la création d'un tunnel c'est-à-dire la création d'une interface réseau virtuelle sur le PC. Pour s'assurer de l'existence de ce tunnel, utiliser la commande 'ifconfig' dans une console : l'interface "tun0" doit être présente, voir 21.

```

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
              UP LOOPBACK RUNNING MTU:16436 Metric:1
              RX packets:665 errors:0 dropped:0 overruns:0 frame:0
              TX packets:665 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:82504 (82.5 KB)  TX bytes:82504 (82.5 KB)

tun0    Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:127.0.1.1 P-t-P:127.0.1.1  Mask:255.255.255.255
        inet6 addr: fe80::1/64 Scope:Link
        inet6 addr: aaaa::1/64 Scope:Global
              UP POINTPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
              RX packets:0 errors:0 dropped:0 overruns:0 frame:0
              TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:500
              RX bytes:0 (0.0 B)  TX bytes:144 (144.0 B)

wlan0   Link encap:Ethernet HWaddr c8:bc:c8:e0:82:65
        inet addr:10.4.184.210 Bcast:10.4.184.255 Mask:255.255.255.0
        inet6 addr: fe80::c8bc:c8ff:fee0:8265/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              RX packets:6166 errors:0 dropped:0 overruns:0 frame:0
              TX packets:4709 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:6635751 (6.6 MB)  TX bytes:658345 (658.3 KB)

nicolas@macbook:~$ 

```

FIGURE 21. Captre d'écran de la commande "ifconfig" montrant la création d'un tunnel.

b) Protocole 2: Le protocole 2, voir tableau IV, permet de vérifier le fonctionnement d'un noeud E.M.M.A. en tant que serveur. Il démontre la possibilité de créer plusieurs ressources, de les modifier et de les supprimer par l'intermédiaire de requêtes CoAP (voir plugin COPPER, figure 22). Ce protocole met en évidence la grande souplesse des standards 6LowPAN et CoAP qui permettent de faire interagir un micro-contrôleur avec un navigateur WEB quelconque.

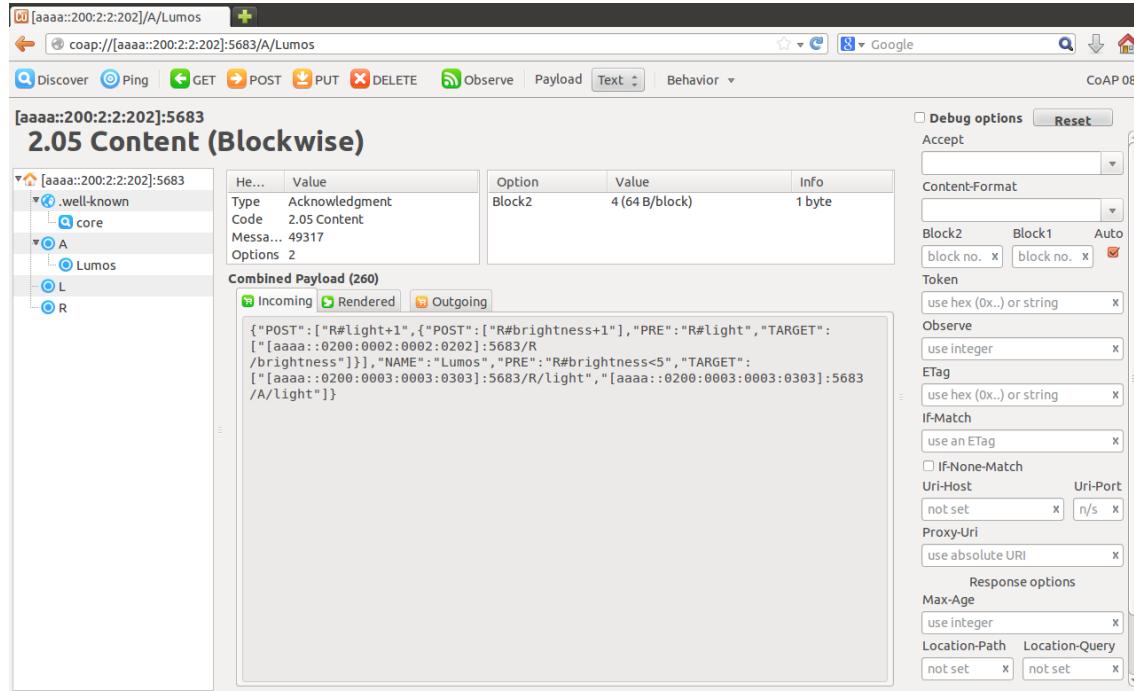


FIGURE 22. Capture d'écran du plugin Copper (présence des requêtes DISCOVER, GET, PUT, POST et DELETE).

c) Protocole 3: Le protocole 3, voir tableau V, permet de vérifier le fonctionnement de l'utilitaire de chargement d'agents. Ce dernier, intégré à l'interface graphique du simulateur COOJA, est visible sur la figure 23. Le chargement de l'agent peut être

vérifié de visu en utilisant le plugin COPPER (figure 22) pour effectuer une requête de type "GET" sur l'agent à vérifier.

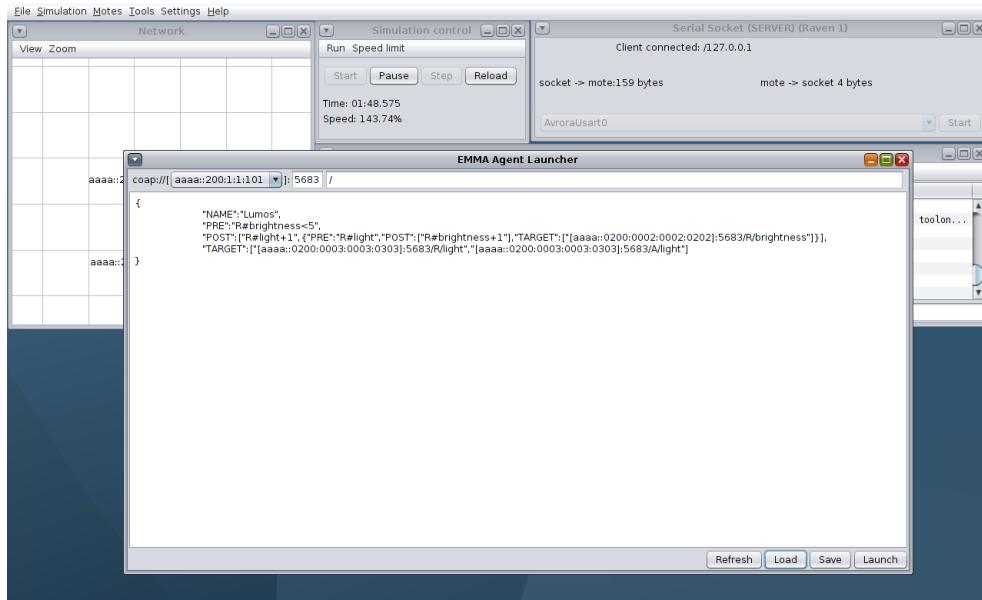


FIGURE 23. Capture d'écran de l'utilitaire de chargement d'agents, intégré à l'interface graphique de COOJA.

d) Protocole 4: Le protocole 4, voir tableau VI, présente un fonctionnement avancé du framework E.M.M.A. : le déploiement d'agents. La réussite du déploiement peut, encore une fois, être vérifiée en utilisant le plugin COPPER pour effectuer les requêtes adéquates à la vérification. Dans le cas d'un déploiement d'agents multiples, l'utilisation du simulateur (voir figure 23) prend tout son sens car il permet une quasi-omniscience dans le réseau simulé. D'autre part, il permet d'effectuer les tests à moindre coûts car il n'est plus nécessaire d'acheter le matériel physique pour tester le fonctionnement d'une application.

C. Interprétation

1) Validations: Les objectifs présentés ci-dessus ont tous été validés au travers des quatre protocoles expérimentaux menés. Des expériences complémentaires pourront être effectuées ultérieurement notamment sur des réseaux de plus grande taille. Ceci permettant de vérifier la fiabilité du framework E.M.M.A. appliquée à de grandes structures mais aussi de mesurer et comparer certains paramètres avec ceux d'un réseau centralisé. Les points qui semblent pertinents à ce niveau sont :

- Temps de déploiement.
- Quantité de paquets réseau échangés pour plusieurs applications type (déploiement, collecte de données, ...).
- Consommation moyenne d'énergie pour une application donnée.

D'autre part, un point qui n'a pas été abordé dans ce rapport car ne faisant pas partie du Cahier des Charges mais toutefois extrêmement important dans le domaine des réseaux de capteurs est la sécurité. L'implémentation actuelle d'E.M.M.A. n'est pas sécurisée car ce sera l'occasion de développements futurs. Dans ce cadre, des tests visant à déterminer les failles actuelles du framework sont envisageables. Ceci afin d'adapter la réponse, en terme d'implémentation, à apporter au framework dans ses prochaines versions.

2) Limitations: Les expériences menées dans ce rapport ont montré certaines limitations du framework E.M.M.A. qu'il est important de prendre en compte pour les développements futurs. Ces limitations sont de trois natures :

- Coût processeur lors de l'exécution de simulations.
- La mauvaise synchronisation de la liaison série au sein du simulateur.
- L'aspect sécurité du framework.

Bien que la simulation soit un outil très pratique pour développer rapidement et identifier facilement les sources d'erreurs, sa limite est atteinte lorsque le nombre de capteurs émulés augmente. Le simulateur COOJA est codé en JAVA et utilise donc une première machine virtuelle pour son fonctionnement. Ajouté à cela, chaque noeud émulé pour la simulation nécessite une machine virtuelle supplémentaire. Les ralentissements les plus notables proviennent du synchronisme de JAVA qui oblige le simulateur à attendre que chaque noeud ait fonctionné pour avancer d'une étape : une gestion asynchrone du réseau simulé permettrait un meilleur rendu et la simulation de réseaux à plus grande échelle.

La limitation citée concernant la mauvaise synchronisation de la liaison série est en partie liée à ces mécanismes. Un paramètre au sein du code du simulateur permet de régler le délais de synchronisation. Cependant, ce délai dépend de la nature du code

Amélioration	Framework	Fichier(s)
Traitement généralisé des chaînes de caractères	EMMA-node	-
Evaluateur arithmétique en notation post-fixée	EMMA-node	emma-eval.c
Synchronisation de la liaison série	EMMA-cooga	AvroraUsart1.java
Vérification du JSON	EMMA-loader	-
Sécuriser les échanges sans fil	EMMA-node	-

TABLE VII. TABLEAU DES AMÉLIORATIONS À APPORTER AU FRAMEWORK.

compilé : un code utilisant beaucoup la liaison série n'aura pas le même délais qu'un code ne l'utilisant pas. C'est une chose qui peut rapidement devenir agaçante lorsque l'on souhaite déterminer une erreur dans le code compilé.

L'absence, à l'heure actuelle, de mécanismes de sécurité internes au logiciel **E.M.M.A.** le rend inutilisable en production ou commercialisation. C'est donc un point important à souligner et à placer dans les priorités des développements futurs.

3) *Améliorations*: Cette section concerne des améliorations à apporter au framework **E.M.M.A.** en termes d'implémentation, d'optimisation, d'architecture ainsi que de potentielles nouvelles fonctionnalités. Le tableau VII référence cette liste d'améliorations.

CONCLUSION ET PERSPECTIVES

Le projet **E.M.M.A.**, présenté tout au long de ce rapport, permet de connecter des capteurs hétérogènes dans le but de déployer et d'exécuter des applications simples distribuées de type contrôle-commande. Le framework développé s'appuie sur des standards, **6LowPAN** et **CoAP**, qui assurent l'interopérabilité entre plateformes hétérogènes sur le long terme. Les expériences menées donnent un premier aperçu des capacités du framework **E.M.M.A.**, notamment en ce qui concerne l'efficacité des applications distribuées et concernant le coût matériel de déploiement. Les livrables se comptent au nombre de trois : le logiciel exécuté sur chaque capteur, le simulateur adapté à la plateforme utilisée et l'utilitaire de chargement utilisé lors du déploiement d'une application. Chacun de ces livrables est documenté, aussi bien au travers de ce rapport qui en présente l'architecture qu'au détours du code source qui documente chaque fonction. Cette documentation permet un développement du projet sur le long terme, potentiellement par d'autres équipes. Le projet se résume donc à l'heure actuelle en un ensemble d'outils utilisables dans le cadre de la recherche et du développement de solutions distribuées pour les réseaux de capteurs. Certains axes d'amélioration ont été pointés du doigt et ne seront pas en reste :

- La sécurité au sein des applications distribuées est actuellement à l'étude et fera prochainement l'objet d'un papier de recherche au sein du **LACSC**. Les résultats pourront être implémentés dans le logiciel **E.M.M.A.**.
- L'implémentation d'un outil de création haut-niveau d'applications distribuées pour **E.M.M.A.** est le sujet de projet de fin d'études d'un groupe d'étudiants de l'**ECE**. Cet outil fera ensuite partie intégrante du framework **E.M.M.A.**.

C'est à la suite de ces améliorations que le framework **E.M.M.A.** pourra réellement faire partie intégrante des bâtiments de demain.

REMERCIEMENTS

Je tiens à témoigner ma reconnaissance aux enseignants-chercheurs du **LACSC** pour leur accueil, leurs conseils et leur humour tout au long de ce stage. De la même façon, je remercie toute l'équipe de MyRobotics pour les conversations dans le domaine de l'électronique et du logiciel embarqué. Enfin, je remercie M.Duhart de la confiance qu'il m'a témoigné au cours de ce stage et de ses conseils avisés, ce qui m'a permis d'expérimenter les vastes possibilités qu'offre le monde des logiciels embarqués.

RÉFÉRENCES

- [1] C. B. Z. Shelby, *6lowPAN The Wireless Embedded Internet*, Wiley-Blackwell, Ed. Wiley-Blackwell, 2009.
- [2] D. Clement., "Environment monitoring and management agent," vol. 1, 2013.