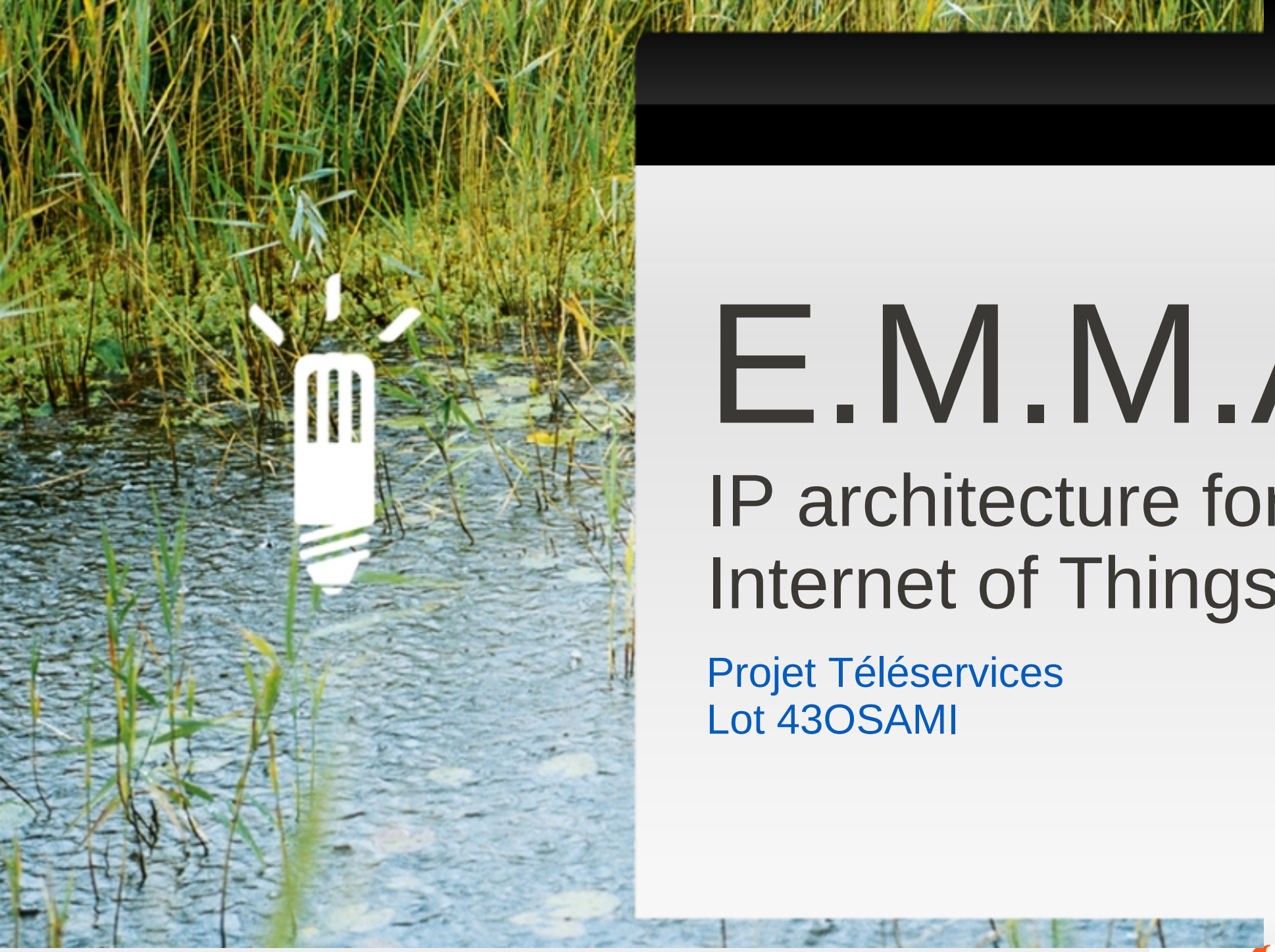


E.M.M.A

IP architecture for Internet of Things

Projet Téléservices
Lot 43OSAMI

Standardization context

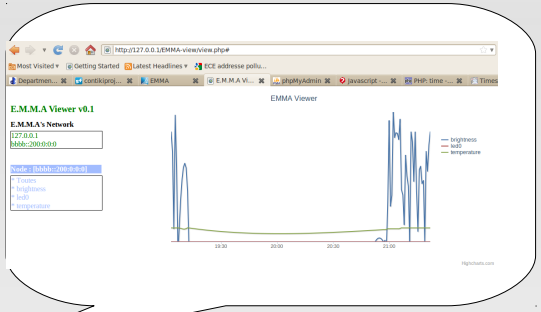
- HomePlug Alliance, Wifi Alliance and ZigBee Alliance promote IPV6 as a core communication protocol for the Smart Grid.
- On June 5th, ZigBee Alliance released the spec 0.7 describing ZigBee IP with Rest architecture.
- At the end of the year, we expect that IPV6 will be the common choice of NIST for smart grid network with SEP2.0 as data model.

Sommaire

- Overview EMMA
- Emma Network
- Emma Node
- Emma Proxy
- Emma Viewer
- Demonstration

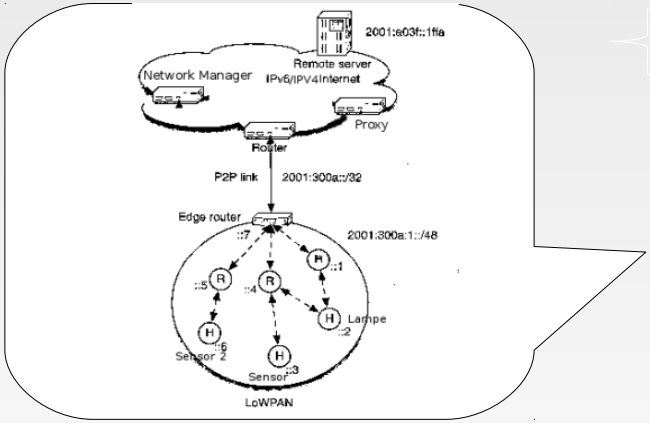
E.M.M.A v0.2

- Energy Monitoring & Management Agent
- Full solution for Internet of Things with 4 abstract separed stages



Final Users :
Consummers, Administrators, Machines...

Emma Network :
Full HTTP Apps (PHP,Javascript...)



Emma Viewer

Emma Gateway | Emma Proxy

Emma Node

Emma API

```
#include "ressources.h"
#include "Rest6-XML.h"

SENSOR_LIST_LOAD(sensors_list);
RESSOURCE_INIT(temperature);
RESSOURCE_INIT(brightness);
RESSOURCE_INIT(data);

int getLight(){
[...]
return light;
}

int getTemperature(){
[...]
return light;
}

void setData(int val){
[...]
}

int getData(){
[...]
return data;
}

void ressources_loader(){
        RESSOURCE_CONFIGURE(brightness,lux);
        RESSOURCE_CONFIGURE(temperature,celcius);
        RESSOURCE_CONFIGURE(data,%);

        RESSOURCE_GET(brightness, &getLight);
        RESSOURCE_GET(temperature, &getTemperature);
        RESSOURCE_SET(data, &setData);
        RESSOURCE_GET(data, &getData);
}
```

Emma Entity :
Simple way for designing & implementation sensor and actuator solution

Emma Network

- 6loWPAN TCP / IPV6 (802.15.4)
- Full HTTP

*** GET /**
*** GET /data/ { * | RessourceName }/**
*** PUT /data/{*|RessourceName}\/**

*** GET /data/ { * | RessourceName } /report/ { * | ReportId } /**
*** PUT /data/ { * | RessourceName } /report/ { IdReport }/**
*** POST /data/ { * | RessourceName } /report/**
*** DELETE /data/ { * | RessourceName } /report/ { * | IdReport }/**

*** GET /data/ { * | RessourceName } /log/ { * | LogId } /**

- RestFull JSON Publication

```
{
"host":"[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]",
"uri":"notify.php",
"port":"80",
"method":"GET",
"body":"",
"name":"brightness",
"period":"100",
"min":"30",
"max":"150"
}
```

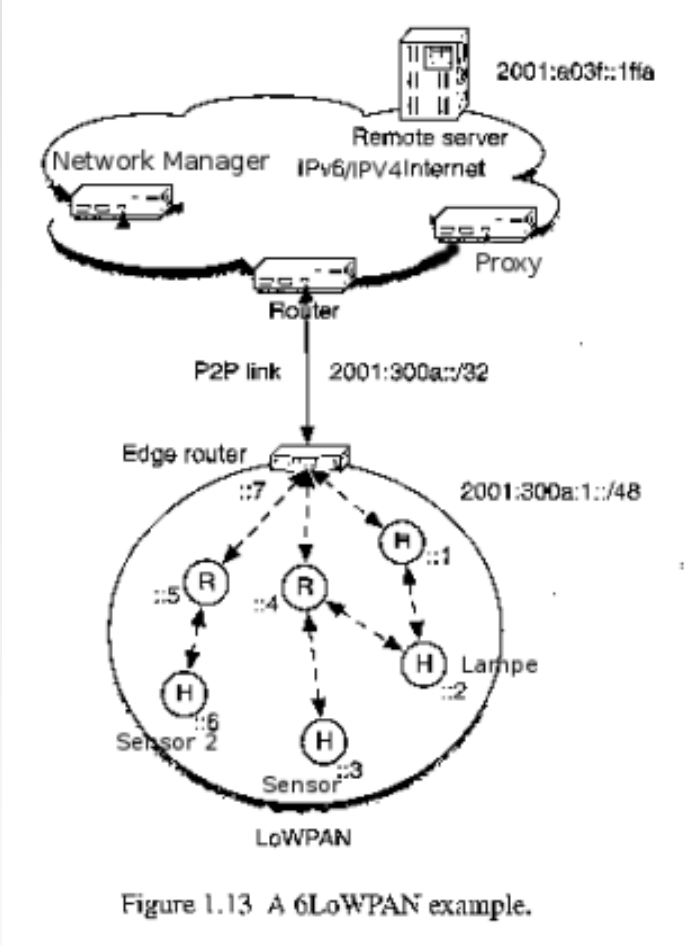


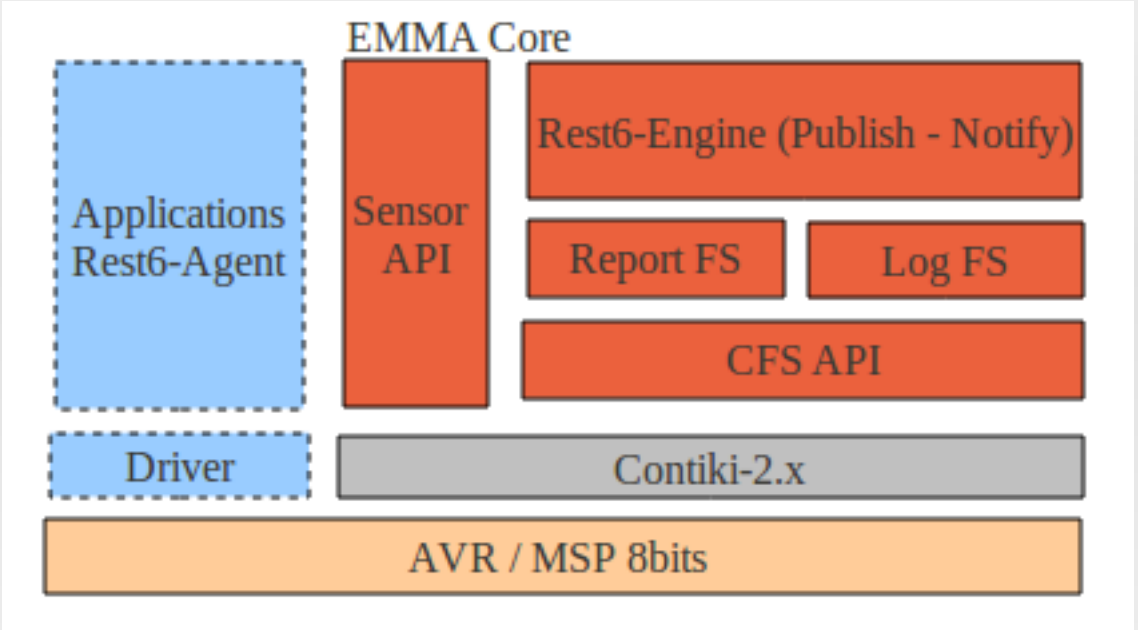
Figure 1.13 A 6LoWPAN example.

Emma Node

- Contiki OS

  Adam Dunkel's OS
  (http://www.sics.se)

- Fonctionnalities :
  - Internal Data Logger (space for 200 samples)
  - Data Publishing by a light internal webserver (Autonomous & Simultaneous connexion)
  - Delegated Task with report notification by a light internal webclient(Autonomous & Simultaneous connexion)
  - Easy way for build final terminal node : Emma API



Emma API

Very easy way for ressource declaration inside EmmaNetwork

```
#include "ressources.h"
#include "Rest6-XML.h"

SENSOR_LIST_LOAD(sensors_list);
RESSOURCE_INIT(temperature);
RESSOURCE_INIT(brightness);
RESSOURCE_INIT(data);

int getLight(){
[...]
return light;
}

int getTemperature(){
[...]
return light;
}

void setData(int val){
[...]
}

int getData(){
[…]
return data;
}

void ressources_loader(){
        RESSOURCE_CONFIGURE(brightness,lux);
        RESSOURCE_CONFIGURE(temperature,celcius);
        RESSOURCE_CONFIGURE(data,%);

        RESSOURCE_GET(brightness, &getLight);
        RESSOURCE_GET(temperature, &getTemperature);
        RESSOURCE_SET(data, &setData);
        RESSOURCE_GET(data, &getData);
}
```

Standard driver developpement with free standard bus API (I2C, PWM, ADC, DIGIT)

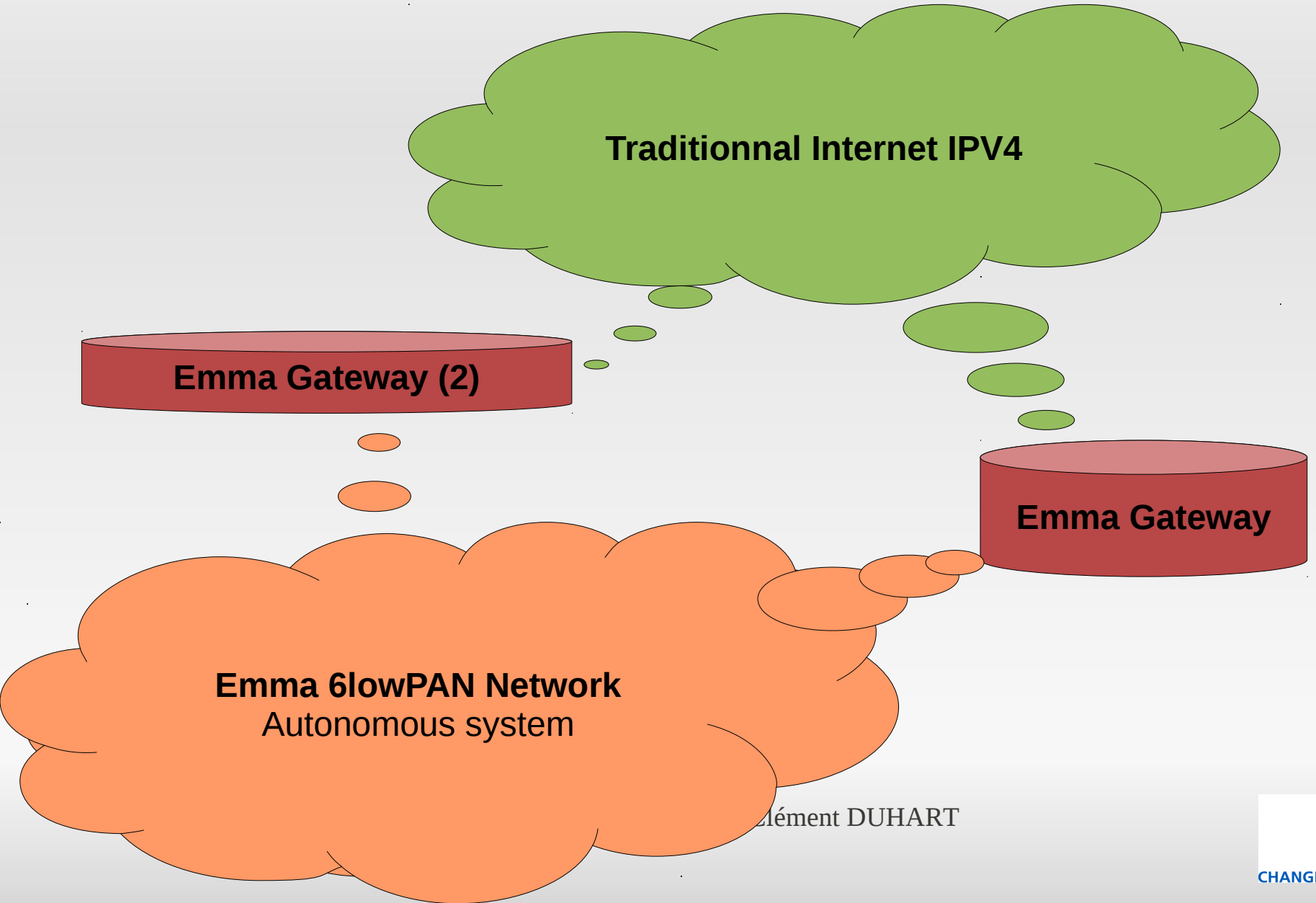
Auto logger for each ressource

Multiple Report Notification for each ressource

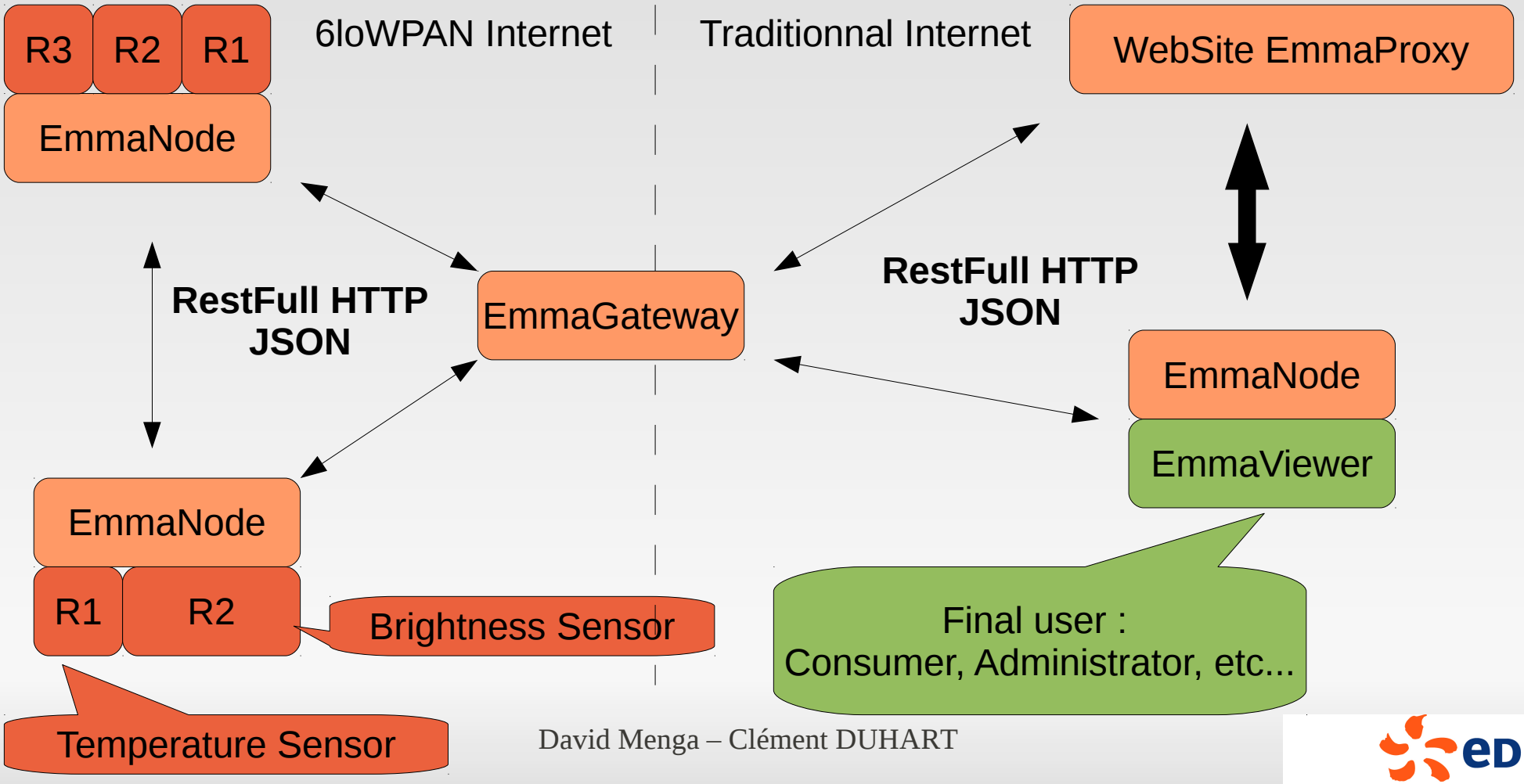
Associate and configuration for driver-ressource interface

Web Shell : Administrator interface

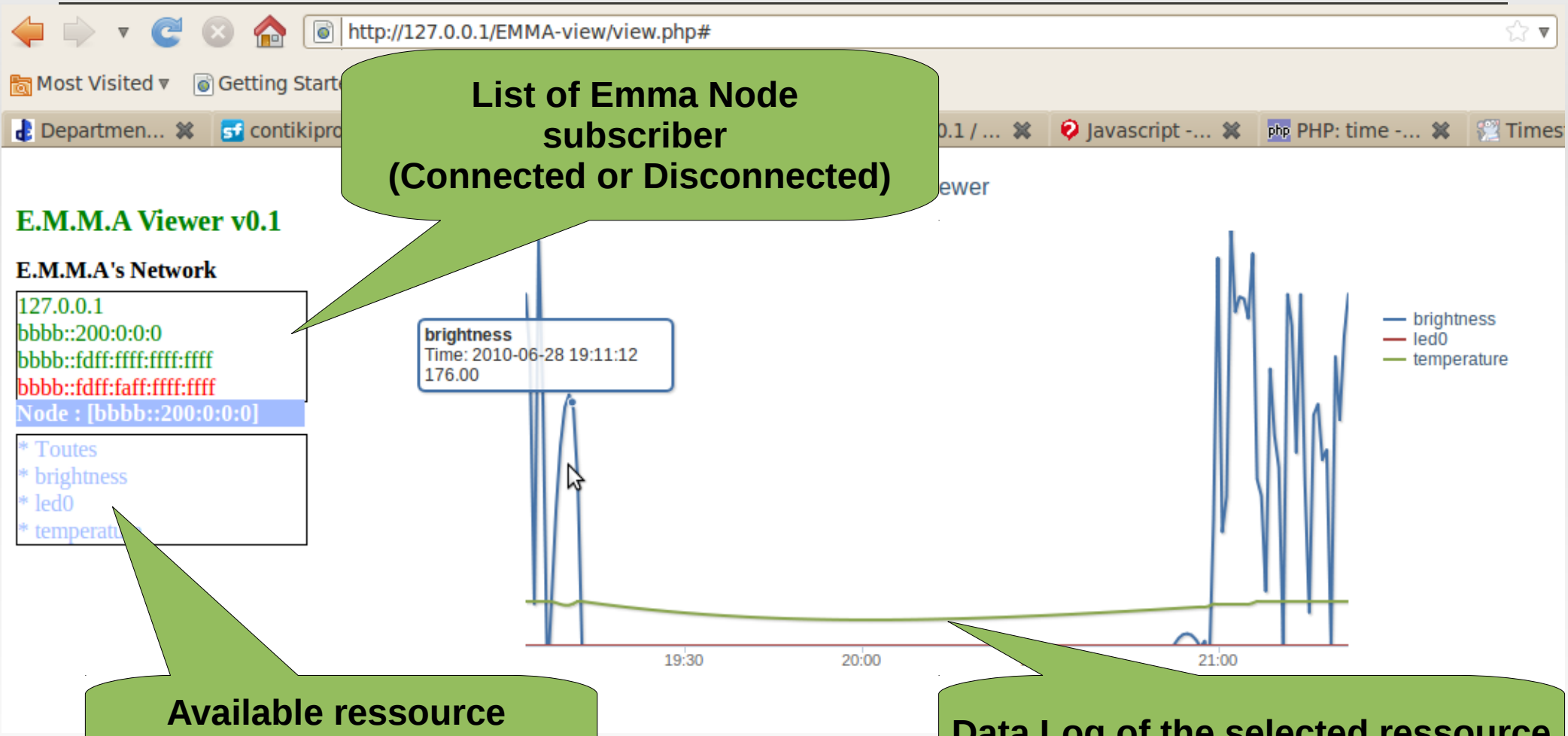
Emma Gateway



Emma Proxy



Emma Viewer



List of Emma Node subscriber (Connected or Disconnected)

Available ressource on the selected Emma Node

Data Log of the selected ressource

Thanks for your attention

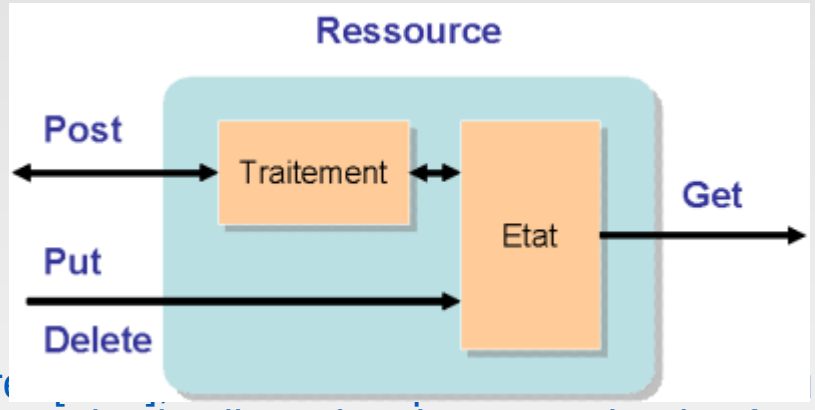
And now, demonstration …

Annexe : L’architecture REST (Representational State Transfer)

- REST est un style d'architecture, pas un standard. Il n'existe donc pas de spécifications de REST. Il faut comprendre le style REST et ensuite concevoir des applications ou des services Web selon ce style.
- Bien que REST ne soit pas un standard, il utilise des standards. En particulier :
- URI comme syntaxe universelle pour adresser les ressources,
- HTTP un protocole sans état (stateless) avec un nombre très limité d'opérations,
- Des liens hypermedia dans des documents (X)HTML et XML pour représenter à la fois le contenu des informations et la transition entre états de l'application,
- Les types MIME comme text/xml, text/html, image/jpeg, application/pdf, video/mpeg pour la représentation des ressources.
- REST concerne l'architecture globale d'un système. Il ne définit pas la manière de réaliser dans les détails. En particulier, des services REST peuvent être réalisés en .NET, JAVA, CGI ou COBOL.

Annexe 2 : Au cœur de REST, HTTP, GET et POST

- Le deuxième composant de l'architecture REST est le protocole HTTP. Ce protocole comporte deux méthodes principales GET et POST et deux manières de transmettre des paramètres, soit dans l'URI, soit dans les données d'un formulaire.
- Quand doit-on employer GET et quand doit-on employer POST ?



- L'utilisation de GET est "sûr[illegible]rce ne doit pas être modifiée par un GET. Ceci autorise les liens, la mise en cache, les favoris. Il faut donc utiliser GET pour des opérations qui ressemblent à des questions ou à des lectures de l'état de la ressource.
- En revanche, il faut utiliser POST quand la demande ressemble à une commande, ou quand l'état de la ressource est modifié ou quand l'utilisateur est tenu pour responsable du résultat de l'interaction.
- Deuxième principe de l'architecture REST : le HTTP GET est "sûr", c'est à dire que l'utilisateur ou son agent peut suivre des liens sans obligations.