Wireless Sensor Network Cloud Services: Towards a Partial Delegation

Duhart Clement*†, Cotsaftis Michel* Bertelle Cyrille†
*ECE Paris School of Engineering, LACSC, France <duhart,mcot>@ece.fr
†Normandie Univ, LITIS, FR CNRS 3638, Le Havre University, France cyrille.bertelle@univ-lehavre.fr

***Abstract*—Current emerging technologies in Wireless Sensor Networks are providing protocols and operating systems to build up the Internet Of Things which heads towards SaaS approach in the Cloud. However, this paper discusses about the impossibility to directly route all nodes from the Cloud. Hence, this paper proposes a new framework based on mobile agents to distribute and deploy applications over a CoAP middleware. This is mainly performed using a distributed publish-subscribe mechanism on each node. The solution allows agents to model distributed applications and their remote deployments.**

***Keywords*—Wireless Sensor Network, 6LoWPAN, CoAP, mobile agent, publish-subscribe, deployment, petri network.**

I. INTRODUCTION

For several years, Internet Of Things (IoT) development has put some emphasis to connect several billion of embedded devices in order to propose new kind of services around Smart Buildings, Smart Clothes, Transport, Logistics and other unknown future use cases. This ambition requires several considerations about service deployment, data heterogeneity, network with large scale, interoperable connectivities, user protection and Quality of Service (QoS). Main technological tendency is to estimate that the Cloud will solve most of these problems if the devices can be connected properly to Internet.

Q-1 *Are we sure to be able to connect all smart objects of a LoW Power Wireless Area Networks (LoWPAN) to the Cloud ?*

- Since the 90's, research communities evaluate a possibility to use Internet Protocol (IP) to connect all objects to the Internet Cloud. Some studies [3], [17] demonstrate its viability according to requirements and constraints of Wireless Sensor Network (WSN) through an IPV6 adaptation: the 6LoWPAN protocol.
- Today, routing protocols are constrained in their addressing capacity by hardware limitations. Indeed, if a packet contains routing path, it is limited by packet size. If a node stores the routing rules, it is limited by RAM storage such as in 6LoWPAN RPL protocol [9].
- Tomorrow, the number of connected objects will increase drastically due to technology miniaturization. According to microcontroller improvement, it will be possible to build smart clothings or materials on which several connected components should interact together. But the constant problem stares the energy constraint.

Finally, server-to-client paradigm should be doubted due to node routing capacity limitation which constrains WSN size.

Q-2 *How Cloud can manage data from hidden connected objects ?*

- Nodes should be configurable to send their data themselves to proxy according to application requirements such as it has been investigated around Publish-Subscribe paradigm [13]. But this paradigm generally requires a server to manage subscriptions.
- Moreover, some basic services can require that some nodes exchange data without complex process such as in data collection or actuator control. By considering distributed publish-subscribe system on which each node asks to an other node to send its data in a given context, data flow management become a critical part of WSN design [7].
- Finally, if several studies demonstrate interest of random data transmission to reduce bandwidth usage [18], the approach becomes very difficult when several services are distributed on a set of common nodes. Data flow should be controlled and managed by a structural design of service data flow to ensure QoS.

Cloud should not be the service host but the supervisor which is able to distribute efficiency services over WSN by configuring properly smart objects.

Q-3 *Which mechanisms can allow an entity to configure an other unknown entity ?*

- CoAP REST-based protocol [10] is an interesting candidate to be the application layer for 6LoWPAN because it includes mechanisms of resource discovery through the Constrained Environments Link Format defined on RFC 6690 [15] to be easily integrable with WEB design system. Moreover, it implements a Publish-Subscribe mechanisms called subject/observer design pattern.
- However, this CoAP feature is only considering variable state to push data value to observers. This is useful if a proxy is collecting all data to drive all nodes but not enough, for example if a sensor should drive directly an actuator according to a threshold.
- By extension, model should be considered the set of subscriptions, such as the set of elementary node behaviors to produce a distributed service over the WSN. This external behavior is depending of internal resources like tropic agents defined by a set of Event-Condition-Action configured according to desired services.

Cloud should deploy configurations in order to delegate deployment processes for inaccessible nodes by visible node.

Q-4 *How can data heterogeneity be managed between nodes and applications without intermediate proxy ?*

- Traditionally, a proxy asks or receives data from nodes and forwards them with the right format for the application. But if there is no proxy, the data should be initially in the right format. So, nodes should be able to adapt the data for the application consumer.
- An interesting approach is to use a packet preprocessor which prepares data format and CoAP requests according to data consumers. This is a common approach in web-based systems which have the same heterogeneity problematics.
- Finally, if proxy should be replaced such as previously presented, nodes should manage heterogeneity problem themselves by using a Publish-Subscribe to determine when a data should be sent, and a preprocessor to generate data in proper consumer format.

Proxy can be replaced by distributing its functions of heterogeneity management and data flow control, but what is the cost of this delegation on constraint nodes?

These different issues are addressed in this paper which is organised as follows. In Section II, Environment Monitoring and Management Agent (EMMA) system is presented with some implementation discussions in Section III and evaluation results in Section IV. After a brief positionning with related works in Section V, conclusions appear in Section VI.

II. DISTRIBUTED EMMA SYSTEM

In Wireless Sensor Network (WSN), nodes are consumers or producers of data but also can be computing nodes for particular treatments. So, each node can host several services like sensor output, actuator control or complex services like mathematical operations. In this scenario, each node has several applications which should be linked together locally or remotely to produce distributed services.

In Distributed System (DS), service interfaces require an abstraction layer called middleware in order to facilitate the component compositions which are the set of processing and message passing to bind them on physical nodes. Moreover, this abstraction layer allows supervisors to monitor and control the whole system independently of component characteristics.

In EMMA system, all components are abstracted by CoAP resource interfaces including node services, resources binding and distributed processing. This web-based middleware facilitates its integration with others local IoT services or remote Cloud servers.

As previously discussed, WSN are strongly constrained and cannot allows a supervisor to monitor and control each Distributed System (DS) directly. Delegation policies can be deployed to allow a master supervisor to drive some slave supervisors. However, the fault-tolerance problem becomes the main concern in which a sub-network will be orphan in case of one supervisor failure. EMMA uses Passing Smart Message (mobile agent) to control, drive and configure distributed components instead of supervisors to increase the system fault-tolerance in case of physical node failures.

A. EMMA middleware

Each node has its own Constrained Application Protocol (CoAP) client and server which allows them to interact together.

1) Web based Integration: A node's server contains a set of resources which are represented using JSON format:

- A static resource is a data produced or required by a node's service such as a sensor, actuator or complex treatment.
  Available actions: GET and PUT
- A dynamic resource can be created or deleted by the middleware for a distributed application.
  Available actions: POST, DELETE, GET and PUT.

The hierarchical representation of this resources uses Uniform Resource Identifier (URI) to access them but also to determine their type. The first URI segment defines a service container called ressource root with its preprocessor and its parser such as illustrated on Table I.

URI	Root	Root description	Type	Resource
/A/light-control	/A	Agent resource	JSON	/light-control
/L/light	/L	Local resource	Integer	/light
/S/routes	/S	System resource	JSON	/routes

Table I. RESOURCE EXAMPLES

EMMA middleware has by default the three types of root resources presented on Table I. Resources of type L are simple local numerical data, the S types are system data string resources and resources of type A are agents.

2) Distributed Publish-Subscribe: An agent is a JSON object which allows a data consumer to subscribe to a given resource. It is defined by boolean interest condition, a set of data formats to send and a set of targets.

```
{
   "NAME": "AgentSensor",
   "PRE": "L#brighness<50 && S#time%10 == 0",
   "POST": [
        "R#light+1",
        "L#brighness"
    ],
    "TARGET": [
        "PUT[aaaa::2]:5683/L/light",
        "PUT[aaaa::1]:5683/database/light"
    ]
}
```

JSON Agent 1. An agent hosted on a sensor transmits its brightness value to a database and orders a light to increase its value each 10 seconds if the mesured brightness is lower than 50.

The Agent 1 is posted on root ressource /A for the agent evaluator service to check if the condition PRE is true. In this case, it will send CoAP requests after preprocessing POST payloads by replacing the variables and by computing the value to send to the targets contained in TARGET set. This agent services stores them in permanent memory and process them by block which introduces execution costs evaluated in Section IV-B.

3) Subscription Deployment Delegation: Agents are also resources and can be created, deleted, read or edited like any resource. Consequently, an agent can create or delete an agent including itself. A game of Matroska allows an agent to be composed of other agents which will bind node's resources together to produce distributed services.

```
{
    "NAME": "DiscoverDeployer",
    "PRE": "S#rand%2==0",
    "POST": [
        "A#DiscoverDeployer",
        {
            "PRE": "S#rand%5==0",
            "POST":[
                "{'resources':S#resources}"
            ],
            "TARGET": [
                "PUT[aaaa::1]:5683/NetworkInfo"
            ]
        }
    ],
    "TARGET": [
        "POST[ff02::2]:5683/A/DiscoverDeployer",
        "POST[0::1]:5683/A/DiscoverNotifier"
    ]
}
```

JSON Agent 2. The agent DiscoverDeployer is a self-deployer agent which is sent periodically and randomly to its neighbors and install on them the DiscoverNotifier agent. This agent will send periodically and randomly the resource list of the node to the proxy aaaa::1.

The Agent 2 contains an other agent DiscoverNotifier in order that DiscoverDeployer is carrying it. Moreover, this agent is an implementation example using EMMA system of a *Resource Discovery Mechanisms:*

- **If** a new node appears in the LoWPAN, the agent DiscoverDeployer will be deployed on it by its neighbors.
- **Then** the agent DiscoverNotifier will send service list to a proxy periodically.

The proxy don't need to ask data to nodes and so to have a route toward them because agents on the LoWPAN will configure nodes for it.

4) Heterogeneous Web Services Transcoding: This powerful feature allows an EMMA node to bind any other CoAP nodes together by using CoAP observer mechanism and EMMA agent preprocessor.

```
CoAP Node 1                EMMA Node              CoAP Node 2
     |                         |                        |
     |                         |   GET /temperature     |
     | (registration)          |     Observe: 0         |
     |                         |       Token: 0x4a      |
     |                         +----------------------->|
     |                         |     2.05 Content       |
     | (notifications)         |     Observe: 12        |
     |                         |       Token: 0x4a      |
     |                         |     Payload: 22.9 C    |
     |                         |<-----------------------+
     |                         +--+   /A/Transcoder     |
     |                         |  | PUT /L/t            |
     |                         |<-+ Payload:?value=L#t  |
     | PUT /heater             |--+                     |
     | Payload:?value=22.9 C   |  |     /A/Sender       |
     |<------------------------+<-+                     |
```

JSON Agent 3. An EMMA node contains an agent which subscribes to a node resource. When data is pushed on a temporary EMMA resource, a transcoder agent is fired to generate a CoAP packet for another node.

B. EMMA System

An EMMA system is the whole graph of connected EMMA nodes, this graph with all other nodes form the LoWPAN graph. It is composed of EMMA bundles which are sets of static and dynamic resources to produce a distributed application. The system dynamics are event-oriented. Indeed, agents modify target resources using CoAP requests when they are notified by the modification of their depending resources. Some main common EMMA applications are:

- Transcoding functions for binding other CoAP nodes
- Deployment process of EMMA applications
- Node configuration for data collection
- Distributed control-command schemes
- Network management

Because all of these EMMA applications can run together on the same nodes and transmit on the same wireless medium, it is necessary to distribute computations and dataflows with a scheduling policy over the set of nodes to guarantee system well-functioning.

1) Data Flows Analysis: Petri Network is used on EMMA application design to check system properties and to analyse the event scheduling for ensuring application consistency. The tokens represent the network events and the data flow is modeled by an Ordinary Petri Network. EMMA model differs from traditionnal Petri Network concerning network dynamic. A transition is fired if a condition PRE(X) on the set of input places marking X is true, then the transition produces token for output places marking Y according to an output algebraic function POST(X,Y) such as given in Equation 1.

$$\text{If } PRE(X) \text{ is } true : Y \underset{action}{\longleftarrow} POST(X,Y) \qquad (1)$$

Resources of EMMA model are associated to Petri Network places and agent executions to transitions. As agents are resources, each transition has an associated place which represents it. This double representation allows agent (transition) to modify itself (place): so EMMA system can be self-adaptive. Output arcs are the POST(X,Y) fonctions associated to a Rest operation such as illustrated on Figure 1.

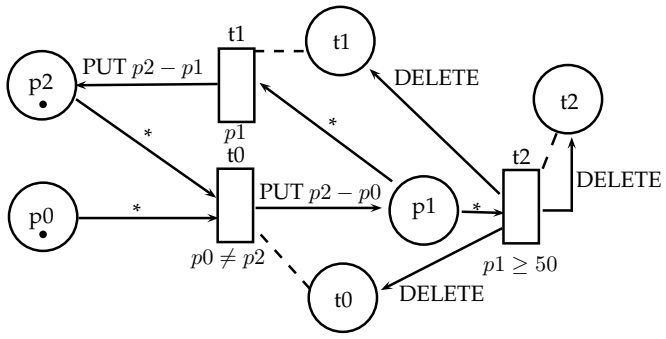

Figure 1. This distributed application computes the differential value $p1(t) = p0(t-1) - p0(t)$. If reaching 50 then the application uninstalls itself.

2) Resource Tuple Space: The set of resources over EMMA system is a tuple space of distributed shared memory. Amongst these resources, the agents consume, produce, edit or delete resources including themselves. Although there is an abstraction through the node Rest CoAP interfaces, resources are stored on nodes memory. On the hand, nodes can store a limited number of resources due to RAM limitation and permanent memory size. On the other hand, if a resource used by an application is far distant of consumer agents, its access time will introduce intensive latency.

3) Distributed Application Deployment: An EMMA application is a set of resources which interact together through some agent resources. When the application mapping is realized according to nodes capacity and the deployment path determined according to network topology and current data flows of other applications, the deployment application can be defined to produce a deployment agent such as on Figure 2.

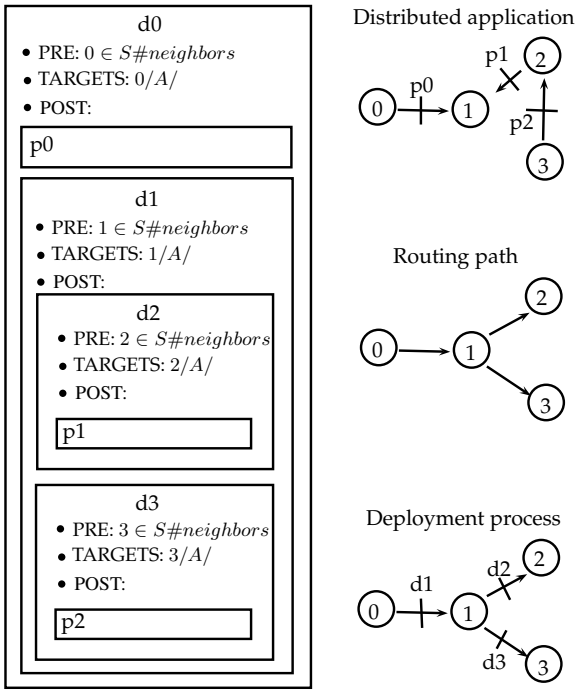

Figure 2. The distributed application has to be deployed on the nodes 0 to 3 according to the routing path which determines the deployment application. The composed agent d0 is sent to the first node which sent the two other agent d2 and d3 to the corresponding neighbor nodes. Each of the deployment agent install its associated application agent p0, p1 and p2.

The algorithms used are SATisfiability problem (SAT) approaches for establishing possible mappings of resources according to nodes capacity and back propagation of agent composition on the routing spanning tree to build the deployment process. This agent composition is relevant in case of deep network on which the supervisor cannot have a routing path to each node due to RAM lack on intermediate routers, but is very expensive about agent size evaluated on Section IV-C.

III. IMPLEMENTATION

EMMA is implemented over a set of Contiki applications which can be runned as standalone to facilitate debugging and system evaluation:

- emma-server is responsible to manage the incomming CoAP transactions and right access.
- emma-client sends the CoAP transactions for emma-resources such as the agents.
- emma-preprocessor manages EMMA resources on RAM and permanent memory and also the API which allows developpers to add new kinds of root resources (or data representations like XML) without modifying EMMA core which only manages text file.

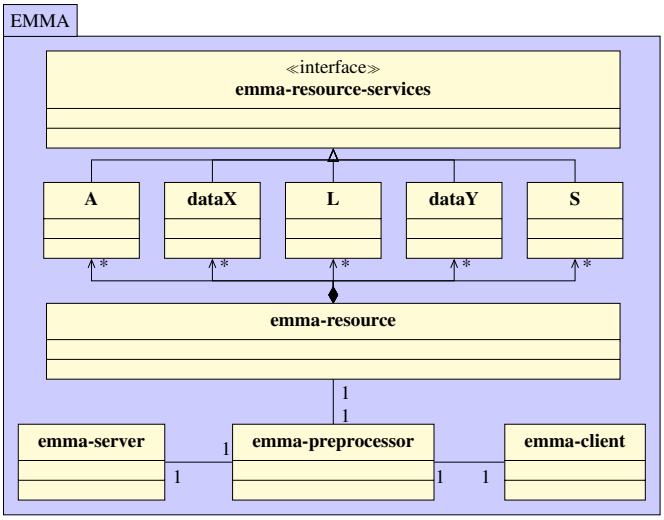

Figure 3. The EMMA architecture is composed of three Contiki applications: emma-resource, emma-preprocessor and emma-client. All EMMA resources are managed by a service container which implements the interface emma-resource-root (a file oriented POSIX API)

The counterpart is that data are not serialized in EMMA server and client but only on the service container which introduces overcomputing cost such as discussed in Section IV. Conversely, the architecture saves memory because resources (including agent) are never completely stored on RAM but are evaluated by blocks. When an agent sends a request to other resources, the payload is parsed for replacing local variables at the same time and if it is a logical-arithmetic formula, the computing stack computes the value (or partial value) by block of CoAP payload. Memory footprint of the different EMMA components are resumed on Table II.

Modules	RAM	Program memory
emma-client	381 Bytes	8267 Bytes
emma-server	456 Bytes	4528 Bytes
emma-resource	648 Bytes	4108 Bytes
emma-JSONparser	0 Bytes	382 Bytes
emma-preprocessor	95 Bytes	4116 Bytes
Total	1.6 KBytes	21.4 KBytes

Table II. EMMA MEMORY FOOTPRINTS

IV. RESULTS ANALYSIS

A. Setup

All following simulations have been processed with COOJA simulator by emulating 13 avr-atmega128rfa1 microcontrollers which have 16 Kbytes RAM, 128 Kbytes flash memory and 4KBytes of EEPROM with a clock at 16 Mhz on a row grid. Results are mean values over 100 simulations.

B. Self-deployment agent

A self-deployment agent allows common agents to be deployed on all nodes. When it arrives on a node, the self-deployment agent sends itself on all node's neighbors before to deploy locally the common agent. The simulation results of previous deployment Agent 2 are plotted in Figure 4.

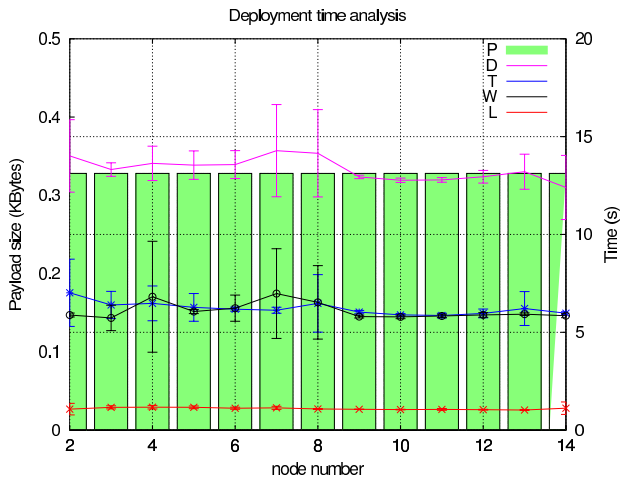

Figure 4. Self-deployment agent evaluation with the deployment time D equal to the agent writing time W required to store the agent of size P, the agent evaluation time and transmission time (transmission process time T to the next node minus its writing time) and the local agent deployment L to evaluate and store locally the configuration agent deployed.

$$D(i) = W(i) + (T(i) - W(i+1)) + L(i) \qquad (2)$$

Figure 4 shows that deployment time is mainly affected by the writing time W which also delays agent transmission (T(i) - W(i+1)). Indeed, a packet can only be sent when the previous packet has been processed due to RAM lack. This writing time is important because resources are stored on EEPROM.

However, the mean deployment time required to deploy configuration agent on all nodes is 89 seconds (ie less than 7 seconds by node). Indeed, the agent is sent to next node before to process the writing of application agent which avoids to loose time during its diffusion. The interesting point is that transmission time and agent evaluation time are very short in comparison to the writing time on permament memory.

The deployment time can be drastically reduced by using other memory space like the RAM or Flash memory which is unwanted for energy saving purpose.

C. Composed deployment agent

The interest of evaluating agents from the permanent memory instead of RAM is the capacity to manage large agents such as the composed deployment agents presented in Section II-B3. This simulation evaluates the effect of agent size on processing, transmission and writing time of agents between neighbors. Composed deployment agent is used to deploy dedicated agents on particular nodes along a determined path. This simulation evaluates a Matroska agent composed of 13 deployment agents which deploys one application agent on each node. When a composed agent is evaluated, it sends its internal composed deployment agent to the next node before locally deploying the application agent. At each node, the deployment agent is lightened by the previous deployment.

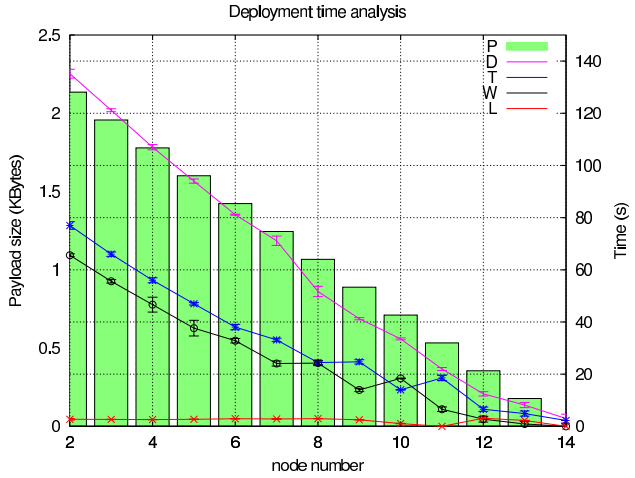

Figure 5. Composed-deployment agent evaluation has the same legend that the Figure 4. This results illustrate that the writing time is proportionnal to the agent size.

Figure 5 shows that the deployment time is decreasing along deployment path. This composed agent is like a Matroska which reduces its size at each deployment. Consequently, the deployment on the next nodes is faster than previous one due to reduction of agent writing cost which requires most of the time. This approach is very slow due to the important size of the agents and is limited by node capacity to host the agent in permanent memory along the deployment path such as presented in Section II-B3.

However, composed deployment agent approach is very interesting when nodes in deep network cannot be addressed directly by the supervisor. Because agent requests are sent from the last node which hosts the agent, it is just required to have a route to next node (and not from the supervisor). Another interest is that its deployment path is determined initially and scheduling policy can be used to avoid any loss of Quality of Service (QoS) for applications that are already deployed.

This kind of deployment using composed deployment agent allows supervisor to deploy agents and configurations on remote nodes unaccessible from the LoWPAN supervisor.

V. RELATED WORK

States of the art on middleware [19], [12] for Wireless Sensor Network (WSN) are very important according to large challenges [14], [6] for the research community.

Among those developments, there are the mobile agents of Agilla [4] or Limone projects which process data locally by moving software toward data instead of the opposite. This can significantly reduce network usages if processes are lightweight but these projects require a proxy-gateway to manage heterogeneity [5]. To deal with it, some other works are oriented on services composition by using SOAP and/or publish-subscribe model [2], [8], [16]. By extending it, the set of distributed resources is considered as a shared tuple place like TeenyLIME [1] to eventually produce a fully distributed system like OASIS system [11].

But there is no open discussion about online distributed node service binding and heterogeneity management.

Environment Monitoring and Management Agent (EMMA) project has been inspired by these different presented approaches but with the purpose to permit its incorporation in current IETF normalizations.

VI. CONCLUSION

This contribution has begun by introducing four questions concerning current protocols normalization for the WSN by the IETF. These questions raise an issue about the capacity for the supervisor like the Cloud to access directly to all nodes of a LoW Power Wireless Area Networks (LoWPAN) due to RAM limitation of nodes. Consequently, if proxy cannot access to all nodes:

How application heterogeneity can be managed ?

The proposed EMMA system allows data to be exchanged, processed and used directly inside the LoWPAN by using a distributed publish-subscribe model over Constrained Application Protocol (CoAP) which is the current application layer proposed by the IETF for 6LoWPAN. The EMMA architecture is based on file processing which allows new kind of data to be managed by EMMA core through new service containers. Among them is included the agent service which allows smart messages to transcode CoAP requests if there are nodes with different webservice API. Moreover, these agents can carry other agents to deploy on inaccesible nodes from the supervisor and to build self-deployment agents in particular cases such as discovery services, rules in case of failures, etc. Finally, EMMA system defines a way to build Distributed System (DS) which connect together nodes services of a LoWPAN.

In future work[1], methodologies will be discussed to ensure consistency and efficiency of these distributed applications in complex networks. This developpement will include design techniques and algorithms which will ensure data exchanges scheduling, energy efficiency of the WSN and load balancing in terms of computation and data flows.

[1]The example materials proposed in this paper are voluntary simple for well-understanding purpose and will be complexified in future publications.

ACKNOWLEDGEMENT

The authors thank M. Mardegan for its contributions during the developpement and implementation of EMMA project.

REFERENCES

[1] Costa, P., Mottola, L., Murphy, A., Picco, G.: TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. ...on Middleware for sensor ...(2006)

[2] Delicato, F., Pires, P.: A flexible middleware system for wireless sensor networks. ...on Middleware (2003)

[3] Dunkels, A.: uIP-dunkels-2003. In ACM, ed.: Full TCP/IP for 8-bit architectures, San Francisco, CA, USA, Proceedings of the 1st international conference on Mobile systems, applications and services (2003) 85—-98

[4] Fok, C., Roman, G., Lu, C.: Agilla: A mobile agent middleware for sensor networks. ACM Transactions on Autonomous and ...(314) (2006)

[5] Hackmann, G., Fok, C., Roman, G., Lu, C.: Agimone: Middleware support for seamless integration of sensor and ip networks. Distributed Computing in Sensor ...(2006) 1–24

[6] Hadim, S., Mohamed, N.: Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks. IEEE Distributed Systems Online **7**(3) (March 2006) 1–1

[7] Hughes, D., Daudé, M., Coulson, G., Blair, G., Smith, P., Beven, K., Tych, W.: Managing Heterogeneous Data Flows in Wireless Sensor Networks Using a "Split Personality' Mote Platform. 2008 International Symposium on Applications and the Internet (July 2008) 145–148

[8] Khedo, K., Subramanian, R.: A service-oriented component-based middleware architecture for wireless sensor networks. ...of Computer Science and Network ...**9**(3) (2009) 174–182

[9] Ko, J., Gnawali, O., Culler, D., Terzis, A.: Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In: Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN 2011), Chicago 193—-208

[10] Kovatsch, M., Duquennoy, S., Dunkels, A.: A Low-Power CoAP for Contiki. 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems (October 2011) 855–860

[11] Kushwaha, M., Amundson, I., Koutsoukos, X., Neema, S., Sztipanovits, J.: OASiS: A Programming Framework for Service-Oriented Sensor Networks. 2007 2nd International Conference on Communication Systems Software and Middleware (January 2007) 1–8

[12] Molla, M., Ahamed, S.: A Survey of Middleware for Sensor Network and Challenges. 2006 International Conference on Parallel Processing Workshops (ICPPW'06) (2006) 223–228

[13] Moritz, G., Golatowski, F., Timmermann, D.: A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks. 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems (2011) 861–866

[14] Rahman, M.: Middleware for wireless sensor networks: Challenges and Approaches. IEEE Distributed System Online **7**(3) (2006) 2–6

[15] Shelby, Z.: RFC 6690. (2012)

[16] Souto, E., Guimares, G., Vasconcelos, G., Vieira, M., Rosa, N., Ferraz, C.: A message-oriented middleware for sensor networks. Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing - (2004) 127–134

[17] Vasseur, Jean-Philippe and Dunkels, A.: Interconnecting smart objects with ip: The next internet. Morgan Kaufmann (2010)

[18] Wang, J., Zhang, Z., Xia, F., Yuan, W., Lee, S.: An energy efficient stable election-based routing algorithm for wireless sensor networks. Sensors (Basel, Switzerland) **13**(11) (January 2013) 14301–20

[19] Wang, M., Cao, J., Li, J., Dasi, S.: Middleware for wireless sensor networks: A survey. Journal of computer science and ...(January) (2008)