Int J Wireless Inf Networks (2016) 23:173–186
DOI 10.1007/s10776-016-0316-1

# A Resource Oriented Framework for Service Choreography over Wireless Sensor and Actor Networks

**Clément Duhart[1] · Pierre Sauvage[1] · Cyrille Bertelle[2]**

Received: 20 July 2015 / Accepted: 2 July 2016 / Published online: 13 July 2016

**Abstract** Current Internet of Things (IoT) development requires service distribution over Wireless Sensor and Actor Networks (WSAN) to deal with the drastic increasing of network management complexity. Because of the specific constraints of WSAN, some limitations can be observed in centralized approaches. Multi-hop communication used by WSAN introduces transmission latency, packet errors, router congestion and security issues. As it uses local services, a model of decentralized services avoids long path communications between nodes and applications. But the two main issues are then to design (1) the composition of such services and to map (2) them over the WSAN. This contribution proposes a model for decentralized services based on Resource Oriented Architecture in which their communications are designed thanks to an adaptation of Petri Network (1). In addition, the problem of decentralized service mapping and its deployment over a WSAN is successfully resumed by a Pseudo-Boolean Optimization in order to minimize network communication load (2). These contributions are presented using a proposed EMMA middleware as unifying thread.

✉ Clément Duhart
duhart@ece.fr

Pierre Sauvage
sauvage@ece.fr

Cyrille Bertelle
cyrille.bertelle@univ-lehavre.fr

[1] Department of Computer Science and Engineering, LACSC, ECE Paris, Paris, France

[2] ULH, LITIS, FR-CNRS-3638, ISCN, Normandie Univ, 76600 Le Havre, France

**Keywords** Internet of Things (IoT) · Wireless Sensor and Actor Network (WSAN) · Middleware · Resource Oriented Architecture (ROA) · Service Choreography (SC)

## 1 Introduction

The development of the Internet of Things (IoT) has some emphasis about current scientist issues and future industrial applications. Among the IoT applications, the Responsive Environments (RE) are a part of a novel technological area. RE aims to transform our daily environments into intelligent spaces. Historically the domotic systems were automatic systems for controlling appliances. Nowadays the term of RE is referring to an environment connected to Internet. On one hand, the data collection is used by remote service providers to manage macro issues, i.e. the energy providers which have to regulate energy production. And on the other hand, the different appliances are managed by different service companies over Internet. An alarm system can be used simultaneously over Internet such as an health care system for older people and such as a security system. A major challenge for the IOT is the sharing of a common network infrastructure between all current and future services. Wireless Sensor and Actor Networks (WSAN) are used to connect locally the different appliances into a common wireless mesh networks. Appliances get an Internet access through others appliances. WSAN has important constraints in terms of bandwidth, throughput and payload. The network protocols proposed for WSAN, such as ZigBee, were incompatible with Internet Protocol (IP), the Internet standard.

Since the 90's, a lot of research works have experimented approaches around IP to facilitate WSAN incorporation into Internet. IPv6 LoW Power Wireless Area

Networks (6LoWPAN) protocol has been standardized to use IPV6 over IEEE 802.15.4 developed for energy constrained devices [7, 16, 25]. An HTTP based application layer called Constrained Application Protocol (CoAP) is currently investigated to provide WEB based communication transactions [17]. This protocol provides mechanisms for webservice interfaces like REpresentational State Transfer (REST) or Simple Object Access Protocol (SOAP) [21]. A lot of open discussions remain regarding software architectures for RE. Assuming that all appliances use a common network protocol, their applications stay heterogeneous which is addressed commonly by a central system. However in such situation, packet congestion appears around the routers according to network depth. Hence a new approach called Service Choreography (SC) is emerging to distribute locally the data exchanges. But the data heterogeneity is not still managed in framework found in literature.

This paper focuses on methodology of design, analysis and deployment of such distributed services over WSAN. Section 2 presents a literature review regarding the different programming approaches of middleware to extract current limitations for the development of Resource Oriented Architecture (ROA) solutions. The proposed framework Environment Monitoring and Management Agent (EMMA) is introduced in Sect. 3. This framework facilitates SC by providing a flexible distributed Publish-Subscribe mechanisms over CoAP. A methodology to distribute such data exchanges is presented in Sect. 4 according to node hosting capacity with the consideration of deployment processes. Section 5 presents results about association mapping of this rules and discuss about efficient deployment process. Finally, conclusion appears in Sect. 6.

## 2 Related Work

A middleware is a piece of abstraction software which provides advanced functionalities of hardware and network engines to applications. It is composed at least of a network stack, a multi-task manager and the drivers. Hadim and Mohamed [13], Rahman [22] detail the different challenges regarding the network (scalability, mobility and dynamic topology), node architecture (hardware abstraction, resource awareness and modular programming) and data management (aggregation, heterogeneity and quality of service). The design of middleware stays an intensive research domain because its technologies issues are inherent of large applications area. Rubio et al. [23] proposes a classification of programming middleware for WSAN: the *macro-programming* and the *node-centric*.

The *macro-programming* consider a WSAN like an integrated system. Each node is a processing unit which executes particular operations assigned at macro-level. The literature provides three main different approaches of *macro-programming* in which the system is considered indifferently like a cluster of processing units, a distributed database or a Multi Agent System (MAS). Kushwaha et al. [19] presents the OASIS architecture to design applications composed of different tasks distributed over the WSAN. They exchange directly their data in order to build computation flows over the WSAN. These tasks and their interconnections are designed offline and deployed remotely from supervisors. Hence the operations are executed directly inside the WSAN but they are managed remotely. Costa et al. [4] proposes an architecture which considers the WSAN like a distributed database. Instead of collecting all data into a database, the requests are directly transmitted by broadcasting over the WSAN. The nodes resolve locally the request in order to provide an aggregated response to the requester. Fok et al. [10] and Hackmann et al. [12] have developed another macro-programming approach based on mobile agents. Each node has a Virtual Machine (VM) in order to execute soft-coded applications. An application is composed of different role based agents which exchange data and perform operations onto a virtual tupple space. In MAS, each agent tries to satisfy its goals under its constraints in order to reach a global stationary point. Hence new constraints or goals can be added on runtime by the addition of agents in the virtual tupple space. This strong uncoupling between the application and hardware levels facilitates dynamic and online reprogramming of the WSAN such as there is no global problem formulation. Moreover, Liu and Chen [20] show that this approach is useful to load-balance energy consumption over the WSAN using mobile agent moving over the nodes according to the residual energy repartition.

The *node-centric* design considers the nodes like autonomous devices connected to WSAN. In such approach, the application term is referring to the software embedded into the nodes. They collaborate with others nodes or Internet services like a traditional distributed system. Hence the middleware provides mechanisms for networked applications including service discovery, protocol interfaces and data heterogeneity management. Dunkels et al. [8] has developed a complete micro-operating system able to execute such applications in parallel on a single node. These applications communicate with other local applications through an event-based messaging engine and remotely with others services with the uIP stack. They propose an advanced solution using standard protocol 6LoWPAN to deploy remotely the binary applications over the air like on traditional computer systems. However in WSAN, the nodes can not be configured manually and individually according to the large scale of the network. Delicato et al. [5], Eduardo et al. [9] and

Khedo and Subramanian [15] propose mechanisms based on Publish–Subscribe pattern to configure remotely the data exchanges between the applications. The applications subscribe to a class of data through their middleware in order to receive them when they are published on the network.

Both approaches have different interests according to the WSAN purposes. For example in data collection, a distributed database is more interesting than a *node-centric* approach because the system is homogeneous and do not require to collect systematically all data on a central database. In case of IoT applications, the devices and the services are produced and added by different manufacturers along the network lifetime. The lack of standard middleware forces the manufacturers to use *node-centric* middleware in their products. In such situation, the inherent heterogeneity of the applications and the network communications should be managed by supervisors in order to maintain network consistency and to preserve its resources. Kuorilehto et al. [18] conclude their survey that *Currently, they implement technologies and algorithms for application distribution but lack an approach combining a distributing middleware layer to OS providing a single node control.* Recently, Cherrier et al. [2] and [3] has proposed a new framework based on choreography of services for WSAN. They combine both approaches proposing a distributed *node-centric* based middleware with a high level language to describe macro-applications. The nodes provide web-services to produce or consume data which can be used simultaneously by different applications. Hence the authors propose a model of Finite State Machine (FSM) in order to guarantee the system consistency according to the different network exchanges between the node applications. The authors compare their work with centralized approaches based on the orchestration of services in which a gateway collects all the data and controls remotely the system. They demonstrate analytically and empirically that choreography model has better performances in terms of reliability, energy efficiency and scalability than orchestration. However the resource limitations in term of memory on node do not allows them to manage the heterogeneity of sequential protocols used at application layer such as SOAP. Hence, Guinard et al. [11] proposes a ROA framework for service choreography. This REST model uses the Publish–Subscribe mechanisms at resource level. Hence, the protocols which require sequential exchanges are implemented like a FSM of several Publish–Subscribe instead of a part of the webservice. An application is resumed by a graph of resource interactions described like Publish–Subscribe configurations. When a resource is changing, its content is transmitted to its dependent resources in order to form a cascading computation flow over the WSAN. In literature review, the establishment of choreography between node applications are operated manually from supervisor.

In this paper, the following questions are addressed to propose a Resource Oriented Architecture (ROA) middleware with self-reconfiguration mechanisms for service choreography over WSAN:

*How to map automatically resources on nodes?*

*How to guarantee choreography exchange consistency?*

*And how to deploy them in a large scale network?*

## 3 Decentralized Service Model

This Section focuses on Environment Monitoring and Management Agent (EMMA) middleware which is the implementation of the proposed decentralized service model based on Service Choreography (SC) over WSAN. Its Resource Oriented Architecture (ROA) encapsulates node services into containers in order to form a resource tupple space. Among the different services, a VM executes reactive agents which models an augmented Publish–Subscribe mechanism distributed over the WSAN. These agents have the ability to transcode CoAP requests in order to manage locally the heterogeneity with other middlewares or remote services using CoAP. Such as these agents are themselves resources, they have the capacity to be self-deployed with self-rewriting abilities.

A Service Choreography (SC) is formed by a set of services connected by the agents. The resource tupple space provided by the middleware forms a complete abstraction between the service choreography and its execution supports on node. A remote supervisor is responsible to define the best mapping of resources to deploy SC in order to preserve nodes and network load. This mapping takes in consideration the deployment process which is itself a SC composed of agents.

This Section presents the middleware architecture with the different basic EMMA services of agent executions, system interfaces and data storages. The proposed graphical framework based on an augmented Petri Network provides an easy solution to design complex SC with composition features. Its use allows mathematical background to be reused in order to analyse event diffusion of the SC in order to guarantee its consistency. Finally, the different strategies of resource deployment are presented with peer-to-peer deployment, composed deployment and self-deployment.

### 3.1 Resource Oriented Architecture

#### 3.1.1 Service Abstraction

EMMA middleware is based on REST architecture which publishes data through resources used by Constrained

Application Protocol (CoAP). These resources are managed by encapsulated services which can be a driver, a processing or a memory storage. These services are implemented through an internal POSIX file API which provides resource reading, editing, creation or deletion by external CoAP requests. By default, the middleware provides three types of services illustrated in Fig. 1:

- *Agent service (A)* An agent resource is a script executed on a node in order to transmit CoAP requests to other resources.
- *System service (S)* A system resource contains node information such as routing tables, sensor data, actuator state, energy consumption, etc. These resources are input and output interfaces of the system.
- *Local service (L)* A local resource contains variables produced and used by the agents.

### *3.1.2 Agent Resource*

The agent resources are used to model the service choreography by defining distributed configurations of Publish–Subscribe. An agent is a rule to transmit the contain of a resource to another one according to the conditional resource state of the node. If a sensor updates its value contained in a resource, the sensitive agents update other depending resources which forms a cascading computation flow over the WSAN such as illustrated in Fig. 2. The resource tupple space abstracts the network communication which allows the agent to update indifferently a local or a remote resource on another node. The resources are accessible through an Unified Resource Identifier (URI)

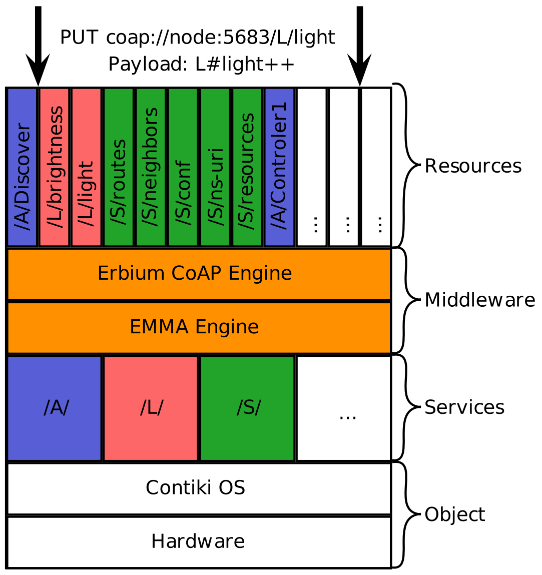


Fig. 1 EMMA Middleware architecture

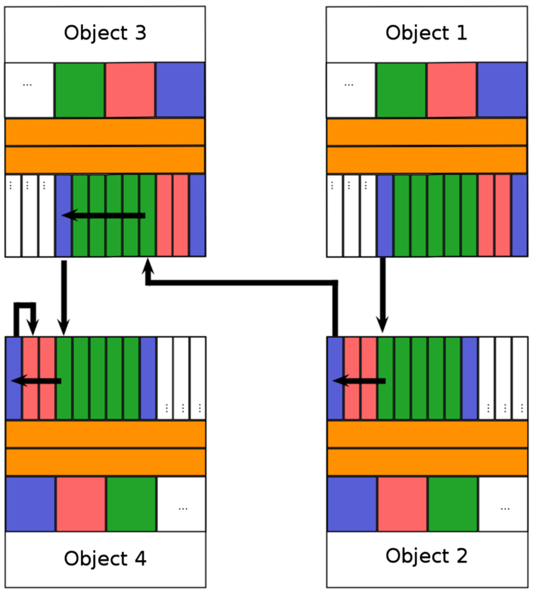


Fig. 2 Example of a Service Choreography

composed of the node IP, the EMMA listening port and its service name.

An EMMA agent $a$ is a JavaScript Object Notation (JSON) file stored on node $n$ which contains a set of resources denoted $X_n$. It is composed of three elements:

- A boolean activation function $PRE_a(X_n)$ Example: $/L/threshold < /S/brightness$
- A list of resource targets denoted $Y_a$ Example: *PUT[IPv6]:port/S/light*
- The associated payloads $\forall y \in Y_a,\ POST_a^y(X_n, y)$ Example: *{'value':'/S/light ++'}*;

When its boolean activation function $PRE_a(X_n)$ is true, it sends CoAP requests to target resource $y \in Y_a$ according to $POST_a^y(X_n, y)$ such as resume in Eq. (1).

$$\text{If } PRE_a(X_n){:}\forall \mathrm{y} \in \mathrm{Y_a}, \mathrm{y} \underset{\text{method}}{\longleftarrow} \mathrm{POST_a^y(X_n, y)} \tag{1}$$

Such as illustrated in following agent examples, the *PRE* field specifies the firing condition to send a request to each resource target stored in *TARGET* field. A target is defined by a CoAP method (GET/PUT/POST/DELETE) and an URI ([*IPv6*]:*port*/*resource*). The payload stored in *POST* field for each resource target is a template file which is processed to replace variables by their resource value. If the payload contains mathematical operations, they are performed before to transmit the request. This payload can contain unresolved variables which are replaced by the resource values of the target node. Because the agents are also resources, they can be created or deleted by other agents including themselves which allows agent design with a self-rewriting ability.

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

JSON Agent 1: This agent is hosted on a brightness sensor which orders to a light to increase its value before to transmit the measured brightness to a database each 10 s if the measured brightness is lower than 50.

```
{
"NAME": "DiscoverDeployer",
"PRE": "S#rand%2==0",
"POST": [
 "A#DiscoverDeployer",
 {
  "PRE": "S#rand%5==0",
  "POST":["{'resources':S#resources}"],
  "TARGET": ["PUT[aaaa::1]:5683/network"]
 }
 ],
  "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 installs on them the DiscoverNotifier agent. This agent will send periodically and randomly the resource list of the node to a database located on aaaa::1.

### *3.1.3 Network Heterogeneity*

The CoAP is based on Hyper Text Transfer Protocol (HTTP) which do not define data formatting nor URI specifications. Hence, this lack do not allows different CoAP middlewares to communicate directly. Traditionally, this issue is managed by a proxy server which translates the requests. In this sense, CoAP has an internal static Publish–Subscribe mechanism which allows the proxy to collect data from all nodes in order to ensure translations.

The EMMA middleware allows this translation to be operated directly by the agents. Because their requests are fully specified through the fields *POST* and *TARGET*, they can send natively any kind of payload with any URI. For example, if a remote middleware uses eXtensible Markup Language (XML) formatting language, the *POST* field of the agent should contain the required XML template. Moreover the agent can generate requests to subscribe to CoAP Observer mechanism in order to collect the data of another middleware. The combination of these two types of agent allows an EMMA node to subscribe data to another middleware in order to generate CoAP requests for another one such as illustrated in Listing 3.

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

JSON Agent 3: An EMMA node contains an agent which subscribes to Observer mechanism of the node 2. When a data is pushed by this node on a temporary EMMA resource, a transcoder agent is fired to generate a CoAP request for another node. This example illustrates heterogeneity management by an EMMA node between a temperature sensor and an heater which cannot communicate directly without the mediation of a proxy.

The management of data heterogeneity for service choreography is a crucial issue which has not be found in literature review such as it requires to centralize data on proxy. The proposed specification of agents provides this feature on each node executing EMMA middleware.

## 3.2 Service Choreography

Service Choreography (SC) is a set of node web services interconnected in order to exchange their data in peer-to-peer fashion. They are configured through the augmented Publish–Subscribe of EMMA to distribute the deployment processes, the control-command loops between sensors and actuators, the service discovery mechanisms and the management of data heterogeneity. The design of SC is a challenge regarding problem complexity of concurrent accesses on distributed resources during event diffusion over the WSAN. The proposed framework uses an augmented Petri network to model them at an abstraction level and to analyse their logical properties.

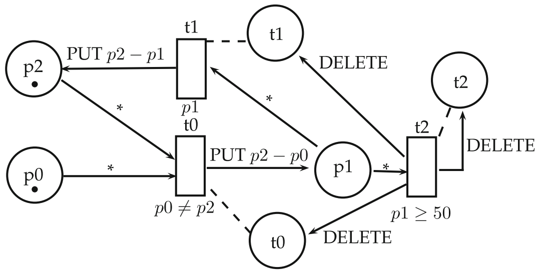


**Fig. 3** This Service Choreography (SC) computes the differential value $p1(t) = p0(t-1) - p0(t)$ through agent t1 and t2. If reaching the value 50, the agent t2 is fired and uninstalls application including itself

### 3.2.1 Model Based on Petri Network

In EMMA, the SC are modeled by an augmented Petri Network in which requests are referred by tokens, agents by transitions and resources by places. A transition is fired if these two conditions are satisfied: (1) a token appears in any input places and (2) the agent boolean condition returns true. This transition activation produces a token for each output places and changes target resource values by corresponding pre-processed payload. Agents are also resources then, each transition is associated to a place. If this kind of place is deleted, the associated transition is destroyed, in the same way for creation or edition. Therefore, this Petri Network model is dynamic and can change during its execution. This model illustrated in Fig. 3 allows SC to be simulated independently of its execution supports. Its behavior is validated thanks to classical algorithms found in literature of Petri Network such as safety, liveness, reversibility, determinism, termination, output-correctness and input-dependence [1]. Moreover, classical patterns can be reused directly such as Sequence, Parallel Split, Synchronization, Exclusive Choice, Simple Merge, Multi-choice, Structured Synchronizing Merge, Multi-Merge, Arbitrary Cycles, Multiple Instances [24].

### 3.2.2 Deployment Agent Examples

Deployment process consists to map typed place on typed resources, including agents, on nodes. There are several ways to generate deployment process according to SC complexity, network topology and already deployed applications:

- *Direct deployment* sends directly agents to each node from supervisor. This approach is interesting for configuration adjustments but not efficient if there are a lot of resources to deploy in deep networks and moreover unavailable in case of hidden node problem [6].

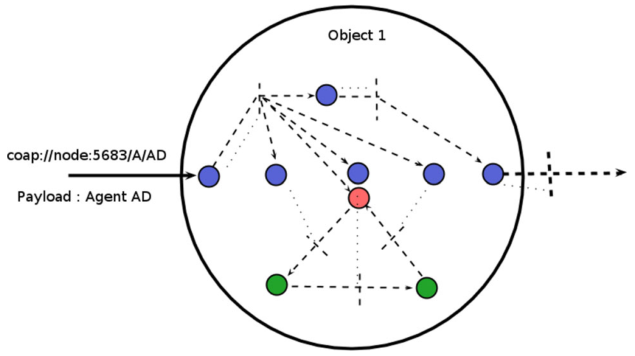


**Fig. 4** Example of a Matroska deployment agent AD arriving on a node to install resources before to sent next deployment agent to the next target node

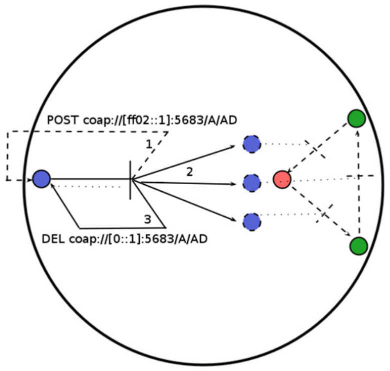


**Fig. 5** Example of a self-deployment agent which duplicates itself to its neighbors before to install the agents and to delete itself

- *Composed deployment* illustrated in Fig. 4 uses agents to carry other agents. They are sent on a node in order to install locally the resources and also to launch other deployment agents in a particular region of the WSAN. These deployment agents produce deployment chains like a Matroska game. This ability allows deployment process to be distributed over the WSAN, however the overhead produced by these agents is important according to the number of contained deployment agents.
- *Self-deployment* is a flooding approach in which a deployment agent is broadcast to all neighbors. It contains all resources to deploy over the WSAN for a SC. When it arrives on a node, it deploys resources required by this node according to its resource context. Then it moves to next ones such as illustrated in Fig. 5. This kind of agents have a very large size but they are interesting in deploying common SC like the Service Discovery mechanism.

These deployment chains are themselves SC. They are designed and validated by EMMA Petri Network and deployed by one of the above deployment way. Therefore, the application deployment process has to be considered during the mapping process of SC.

## 3.3 Methodology for Service Choreography Mapping

Mapping process consists to associate for each required places of a SC an empty resource space on a node in order to minimize network communication costs. This mapping process is composed of three specification stages: functional design, instantiation and deployment.

### *3.3.1 Functional Design*

The functional design uses previously presented Petri Network to model the SC. The different input-output resources produced by node services are connected through agent place $A$ (modeled by transitions) in addition of temporary resource $L$ (corresponding to places). This specification stage introduces the concept of scope which manages structural dependencies such as illustrated in top of Fig. 6. For implementation reason, a transition resource $a$ must be on the same node $n$ that the resources of its input places $X_n$ required by its activation function $PRE_a(X_n)$. Otherwise for efficiency reason such as high-frequency communication exchanges, a SC designer would like to force several resources to be located on the same node. Hence the term of *scope* is a mapping specification which refers to a group of resources that must be mapped on the same node. In addition the scopes can be constrained in their mapping over the WSAN by:

- *Scope multiplicity* A scope which requires several identical scopes is connected by a link of multiplicity $M$. For example, an agent of data aggregation requires $M$ values produced by sensor resources. Hence the scope containing the agent is linked to the scope which models the sensor resources by a multiplicity parameter equal to $M$.

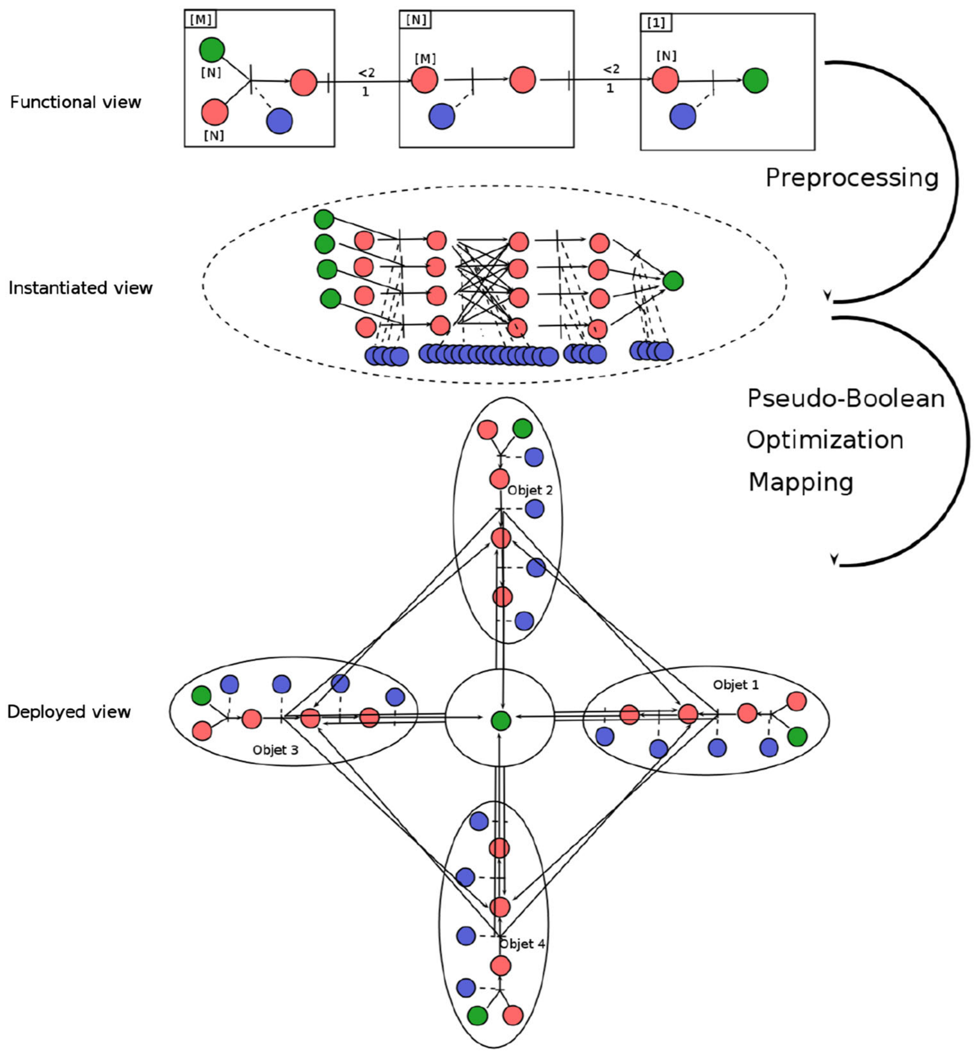


**Fig. 6** Overview of the three steps of mapping process: Functional Design, Instantiation Process and Choreography Deployment

- *Scope dependency* The SC design is operated independently of the target network. However it is possible to constraint the resource mapping in respect of network constraints such as the maximal number of communication hops between two scopes.

#### *3.3.2 Instantiation Process*

The instantiation process generates the global SC graph according to a target WSAN. The list of required resources produced by node services is established in order to determine the scopes which can be mapped according to their multiplicity parameters and the network constraints. The mapping problem presented in Sect. 4 evaluates the set of mapping permutations in order to minimize an objective function representing global network load. In addition of the SC graph, the mapper adds the SC of the deployment process to keep free resource for the composed agents responsible of the agent $A$ and temporary $L$ resource installations.

#### *3.3.3 Choreography Deployment*

SC deployment process generates for each node an agent which contains all resources to deploy on it. These agents are included into one to another by back propagation along a deployment path which is by default the routing tree from supervisor. Their composition is limited by their size according to the memory capacity of the nodes along the deployment path. This procedure is reiterated until that the WSAN coverage is reached in order to send the generated deployment agents to the corresponding nodes from supervisor.

Next Section presents the proposed formalization to address the mapping problem from instantiated graph to the deployment of resources on a WSAN target.

## 4 Theoretical Mapping Problem Formalization

Application mapping problem consists to determine an efficient distribution of resources on the nodes to minimize communication load. Based on the functional constraints, the network topology and the hosting capacities of nodes, the instantiated graph must be mapped over the WSAN with its deployment process.

### 4.1 Model Definitions

#### *4.1.1 Network*

A WSAN is composed of a set of nodes $N = \{n_1, n_2, \ldots\}$ modeled by a distance matrix $D$ in which $d(n_1, n_2)$ represents the cost metric between $n_1$ and $n_2$. By default, the cost metric is determined by the routing algorithm according to the number of hops between two nodes. It can model any others parameters such as the bandwidth, the link quality or an aggregation of them. If and only if the communications are bi-directional, $\forall n_1, n_2 \in N : d(n_1, n_2) = d(n_2, n_1)$. Because of node memory limitation, the routing tables of nodes are partial, a node $n_1$ can be unreachable from $n_2$. By convention, it will be written $d(n_2, n_1) = -1$.

#### *4.1.2 Resources*

A resource $r$ is an allocated space of memory on a node associated to a unique access path *URI*($r$) defined by */type/ressource_name*. Each resource has a type $j \in T$ corresponding to its service ($T = \{A, S, L, \ldots\}$, see Sect. 3.1.1). $\forall n \in N, \forall \mathrm{j} \in \mathrm{T}$, $R_n^j$ defines the set of resources of type $j$ on node $n$. Then $R_n$ denotes the set of all resources on node $n$, $R_n = \bigcup_{j \in T} R_n^j$.

**Definition 1** card $_j(n)$ refers to the maximum number of resources of type $j$ on node $n$ which is defined during the software compilation. Therefore it is considered fixed for the mapping process.

#### *4.1.3 Scopes*

A scope $s$ is a set of places $P_s$ which must be mapped on the resource spaces of a same node. $M_s$ refers to the multiplicity of the scope $s$ (see Sect. 3.3.1). The set of scopes $S = \{S_1, \ldots, S_m\}$ represents the whole SC to deploy on WSAN. Several scopes can be mapped on a same node, as long as the node capacities are sufficient. Two scopes are called linked if at least one place of the first scope interacts with one of the second. A communication $a$ is modeled by a data exchange function $f_a(t)$ and a weight $w_a$ which represents number of packets required to transmit payload. As the function $f_a(t)$ cannot be assessed a priori, we will rather consider an estimated frequency $f_a$.

The set of all communications from scope $s_1$ to scope $s_2$ is denoted $A_{s_1,s_2}$ such as the sum of all place communication costs defined in Eq. (2) between scopes $s_1$ and $s_2$. Communications between two scopes are generally asymmetric, hence $c(s_1, s_2) \neq c(s_2, s_1)$.

$$c(s_1, s_2) = \sum_{a \in A_{s_1,s_2}} c_a = \sum_{a \in A_{s_1,s_2}} f_a \times w_a \tag{2}$$

#### *4.1.4 Places*

A place $p \in P_s$ is a requirement defined by an access path of *URI*($p$) such as $/type/place\_name$ for the mapping of the scope $s$ on a node. There are two subsets such as $P_s = \bigcup_{j \in T} \{P_s^j \bigcup \dot{P}_s^j\}$.

- $\forall j \in T$, $P_s^j$ defines the set of places which requires an available resource space of type $j$ on the node. *Example:* The place $p \in P_s^L$ requires an empty resource space of type $L$ on the node.
- $\forall j \in T$, $\dot{P}_s^j$ defines the set of places which requires a resource of type $j$ on the node. *Example:* The place $\dot{p} \in \dot{P}_s^L$ specified with the $URI(p) = /L/temp$ requires a resource with the same $URI$ on the node.

## 4.2 Problem Formulation

### 4.2.1 Knapsack Problems

The formulation of SC mapping problem is composed of two variants of the Knapsack problem:

- *Multiple Knapsack Problem (MKP)* Each node is considered like a Knapsack in which places should be mapped according to the node capacity.
- *Multiple Choice Knapsack Problem (MCKP)* The nodes have a finite partition for each resource type. The number of elements by type is limited on each node because of node hosting capacity.

Finally, the problem formulation is resumed by:

*How to map scopes which requires different number and types of places over a set of nodes which do not have the same hosting capacities in terms of resource type and memory size in order to minimize the network communication load?*

### 4.2.2 Service Choreography Mapping

**Definition 2** The operator *size*() is defined such as:

- *size*($r$) refers to the memory space used by a resource $r$.
- *size*($R$), $R \in \{R_n, R_n^j\}$ refers to the memory space used by all resources in $R$.
- *size*($P_s^j$) refers to the memory space used by all places in $P_s^j$.
- *size*($n$) refers to the total memory space of node $n$.

**Definition 3** $\forall s \in S, N_s \subseteq N$ denotes the set of nodes on which the scope $s$ is mappable. A scope $s$ is mappable on a node $n \in N$ if and only if:

- Constraint (3) defines that node $n$ has a free resource for each place $p \in P_s^j$ in scope $s$ of type $j$.

$$\forall j \in T,\ |P_s^j| + |R_n^j| \leq \text{ card }_j(n) \tag{3}$$

- Constraint (4) defines that node $n$ has a resource of type $j$ for each $\dot{p} \in \dot{P}_s^j$.

$$\forall \dot{p} \in \dot{P}_s^j,\ \exists r \in R_n^j / URI(\dot{p}) = URI(r) \tag{4}$$

- Constraint (5) defines that node $n$ has enough hardware memory to contain the places of scope $s$.

$$\sum_{j \in T} \text{ size } (P_s^j) \leq \text{ size } (n) - \text{ size } (R_n) \tag{5}$$

**Definition 4** A Service Choreography (SC), defined by its set $S$ of scopes, is not mappable if one of its scopes is not mappable according to its multiplicity:

$$\exists s \in S, |N_s| < M_s \tag{6}$$

**Definition 5** The set of scopes which can be mapped on the node $n$ is denoted $\forall n \in N, S_n \subseteq S$.

## 4.3 Pseudo-Boolean Optimization

The Pseudo-Boolean Optimization (PBO) consists to minimize a Pseudo-Boolean function under a set of Pseudo-Boolean constraints expressed by equations or inequations. The best SC mapping consists to associate the places of scopes to the available resource spaces on nodes in order to minimize network load over the WSAN.

**Definition 6** The mapping of a scope $s \in S$ on its possible hosting nodes in $n \in N_s$ is denoted by the boolean vector $x_s^n$. Then the set $X$ determines the mappings of all scopes among their possible hosting nodes.

$$\forall s \in S, \forall n \in N_s : X = \{x_0^0, x_0^1, \ldots, x_1^0, x_1^1, \ldots, x_s^n\}$$

Then, the Eq. (7) resumes the total number of boolean literals for the PBO such as the number of possible mapping permutations on the nodes.

$$|X| = \sum_{s \in S} |N_s| \tag{7}$$

### 4.3.1 Communication Cost Function

The cost function $z(X)$ evaluates the impact of the mappings $X$ on the network communication load. The Pseudo-Boolean Optimization (PBO) solver determines the best combinations of scopes and nodes among the set $X$ of permutations in order to minimize the communication costs between the linked scopes. The communication cost of a SC mapping is defined such as the sum of its scope link costs $c(s_1, s_2)$ defined in Eq. (2) times the network distance $d(n, n')$ between their respecting hosting nodes. Then the Pseudo-Boolean function of general communication costs over the WSAN, which must be minimized, is defined in Eq. (8).

$$z(X) = \sum_{s_1 \in S} \sum_{n \in N_{s_1}} \sum_{s_2 \in S} \sum_{n' \in N_{s_2}} c(s_1, s_2) d(n, n') x_{s_1}^{n} x_{s_2}^{n'} \tag{8}$$

### 4.3.2 Constraint Set

The minimization of the function $z(X)$ is constrained by the following set of (in)equations to define available mappings of the SC.

- The Constraint (9) defines that each scope must be mapped several times according to its multiplicity parameter $M_s$.

$$\forall s \in S : \sum_{n \in N_s} x_s^n = M_s \tag{9}$$

- The Constraint (10) forces the mapping of linked scopes to have a network route between their hosting nodes. If there are communications between $s_1$ and $s_2$ ($c(s_1, s_2) > 0$), they can be mapped respectively on $n$ and $n'$ if and only if $d(n, n') \geq 0$.

$$\forall (s_1, s_2) \in S^2, \forall n \in N_{s_1}, \forall n' \in N_{s_2} : c(s_1, s_2) d(n, n') x_{s_1}^{n} x_{s_2}^{n'} \geq 0 \tag{10}$$

- The Constraint (11) defines that the total available resource space required by all scopes mapped on a node $n$ do not exceed its capacity.

$$\forall n \in N, \forall j \in T : \sum_{s \in S_n} |P_s^j| x_s^n \leq \text{card}_j(n) - |R_n^j| \tag{11}$$

- The Constraint (12) limits the total memory usage by the mapped resources on a node to its available hardware memory.

$$\forall n \in N : \sum_{s \in S_n} \sum_{j \in T} \text{size}(P_s^j) x_s^n \leq \text{size}(n) - \text{size}(R_n) \tag{12}$$

## 5 Procedure and Evaluations

This section resumes the installation procedure of Service Choreography (SC) on an heterogeneous WSAN composed of EMMA and others CoAP nodes. The deployment process is evaluated for the two kinds of deployment agents proposed in EMMA framework. The experimental support of self-deployment and composed agents is a SC for *Network and Service Discovery* mechanism. The results explain the choice of composed agents for SC deployment in EMMA mapping engine. Then, the resolution time of mapping problem is investigated on a classical problem in distributed systems: the *Philosopher Dining*. This example offers a SC enough complicated in order to provide a representative benchmark. These evaluations are not compared with other solutions because authors were not able to find in literature review similar approaches for automatic mapping and distributed deployment of SC.

### 5.1 Installation Procedure

1. *Network and Service Discovery* consists to recuperate the lists of nodes and their resources in order to determine the map of the WSAN composed of the network topology and the node services.
   (a) *EMMA nodes* are discovered by the Agent 2 previously presented which self-deploys on them an agent in order to push periodically and randomly the list of contained resources of its hosting node to the supervisor. Because the neighbors and route tables are included in system resources, all 6LoWPAN nodes are discovered.
   (b) *Other CoAP nodes* are directly requested from the supervisor on their *Resource Discovery* (coap://[IPv6]/.well-known/core) to get the list of their resources with their meta descriptions.
2. *Service Choreography Deployment* maps and deploys the different required SC on the WSAN target.
   (a) *Common SC* are self-deployed by an initialization agent such as presented in Sect. 3.2.2 in order to deploy log collection mechanism, configurations of the network and energy management.
   (b) *Mapping Process* determines the best mapping of SC in order to minimize the network communication load and built the composed deployment agents presented in Sect. 4.
   (c) *SC Deployment* launches the different deployment agents on their WSAN area. This deployment is terminated when the Discover Notifier agents notify that all resources are deployed.
3. *CoAP Node Integration* consists to build and launch manually the agents in order to connect them to deployed SC. The data heterogeneity at CoAP layer is managed by translator agents such as presented in Sect. 3.1.3.

### 5.2 Network and Service Discovery Deployment

EMMA agents allow deployment process to be performed by three different approaches: *direct deployment*, *composed deployment* and *self-deployment*. Direct deployment is the common approach in most of contributions for SC middleware. However, it produces an important load in deep network because all deployments are transmitted from the supervisor. Below results compares the two proposed strategies by EMMA between a self-deployment Agent 2 (Fig. 8) and a composed Agent 1 (Fig. 7) deployed on a 14-hop network. The experimentation evaluates the best strategy to deploy the *Service Discovering* mechanism.

Figure 7 prints deployment time $D$ equal to agent writing time $W$ to store agent contained in payload of size $P$, transmitted in $T$ ms to node and executed in $L$ ms on it. Because the agent execution is processed by block, transmission time of deployment agent $i$ is equal to total transmission time minus writing time on next node which is resumed in Eq. (13).

$$D(i) = W(i) + (T(i) - W(i+1)) + L(i) \quad (13)$$

Following figures shows the impact of cumulated agent overhead along the deployment path. For each node, the deployment agent contains SC agents and the composed agents for the next node. This strategy is interesting in distributing the deployment process over WSAN areas which avoids network congestion on routers close of the supervisor. However the deployment of identical SC by this approach produces a useless redundancy.

Fig. 8 presents the deployment of a self-deployment agent which is broadcast over the WSAN. It deploys the SC on the node at its arriving before moving on the next nodes. Its constant overhead is low regarding contained SC, however its use for the deployment of whole SC implies that its payload is very large. Finally, the *self-deployment* is efficient for installation of common SC at initialization in order to avoid redundant compositions whereas the *composed deployment* is used for resource deployment delegation to a local node in the WSAN area of interest.

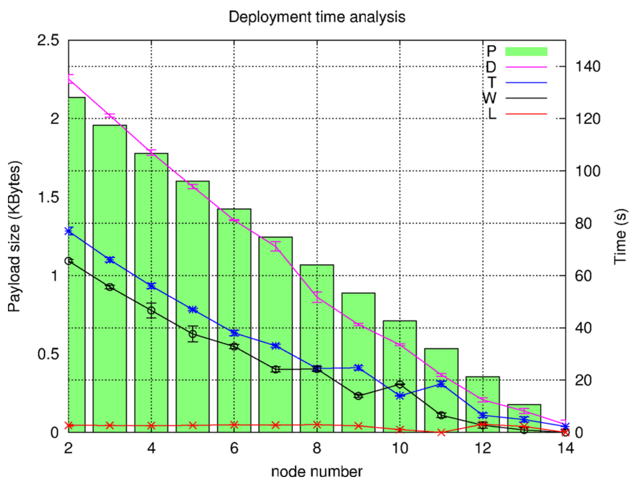


**Fig. 7** Benchmark of a composed-deployment for *Network and Service Discovery Deployment*

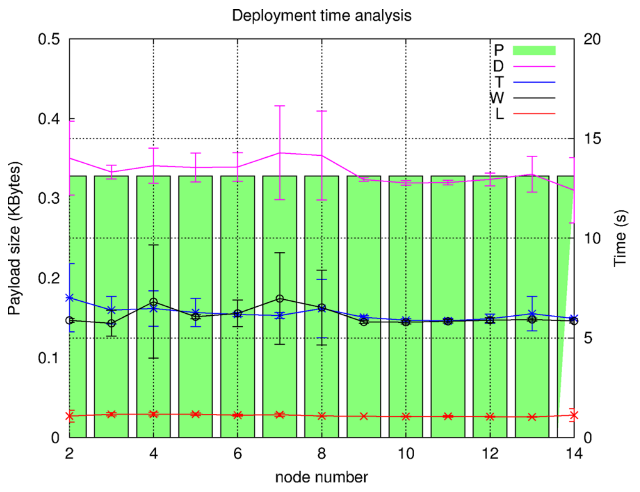


**Fig. 8** Benchmark of a self-deployment for *Network and Service Discovery Deployment*

## 5.3 Philosopher Dining Mapping

The mapping process of *Philosopher Dining* is a classical problem of distributed systems for synchronization issues in concurrent resource access. In the use case of a Smart Home under energy contract of power delivery limitation, the different appliances must not consume simultaneously electricity. Therefore they must be scheduled. One distributed approach consists that each appliance negotiates with the others the permission to be turned on. For example the electric car and the hot water tank alternate their scheduling.

The Fig. 9 presents the functional design on EMMA framework for the Petri Network of *Philosopher Dining* detailed in [14]. Each appliance is represented by a

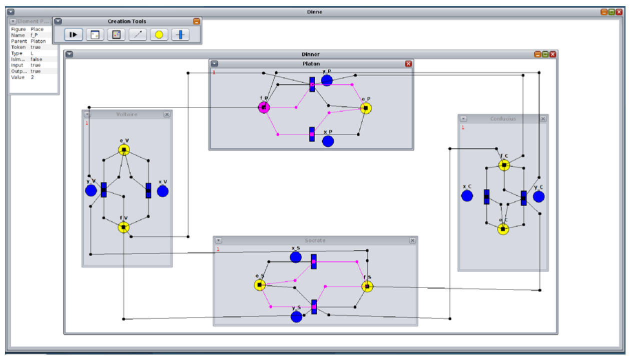
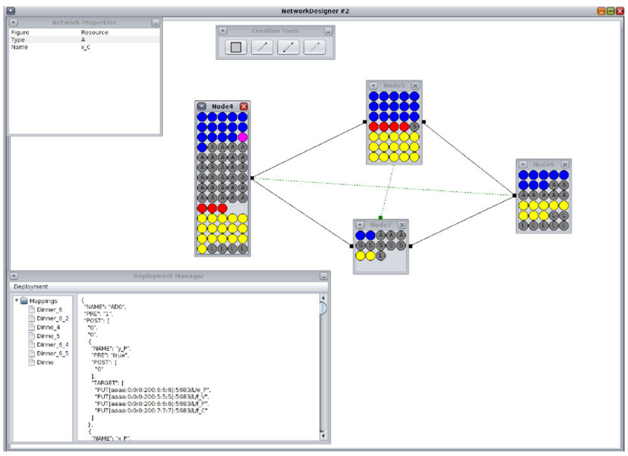


**Fig. 9** Functional view based on EMMA Petri Network for the SC of *Philosopher Dining*



```
{
 "NAME": "AD0",
 "PRE": "1",
 "POST": [
  "0",
  "0",
  {
   "NAME": "y_P",
   "PRE": "true",
   "POST": [
    "0"
   ],
   "TARGET": [
    "PUT[aaaa:0:0:0:200:6:6:6]:5683/L/e_P",
    "PUT[aaaa:0:0:0:200:5:5:5]:5683/L/f_V",
    "PUT[aaaa:0:0:0:200:6:6:6]:5683/L/f_P",
    "PUT[aaaa:0:0:0:200:7:7:7]:5683/L/f_C"
   ]
  },
  {
   "NAME": "x_P",
```

**Fig. 10** Mapping result of the *Philosopher Dining* SC with the deployment agent in down-left corner

philosopher and then a scope. They must exchange energy tokens in order that an appliance can consume if it has the tokens of the other appliances which should not be turned on simultaneously.

The Fig. 10 presents an example result of mapping process for 20 philosophers on 4 nodes with the composed agent of deployment.

The purpose of this experimentation is the evaluation of the resolution time of mapping process according to the number of scopes and nodes. The Fig. 11 illustrates the number of generated constraints depending of the literals $x_s^n \in X$. They increase linearly and mainly according to the number of nodes whereas the number of scopes has a lower impact. The Fig. 12 shows that the resolution time increases drastically according to the number of nodes which is an expected result such as the Knapsack problem

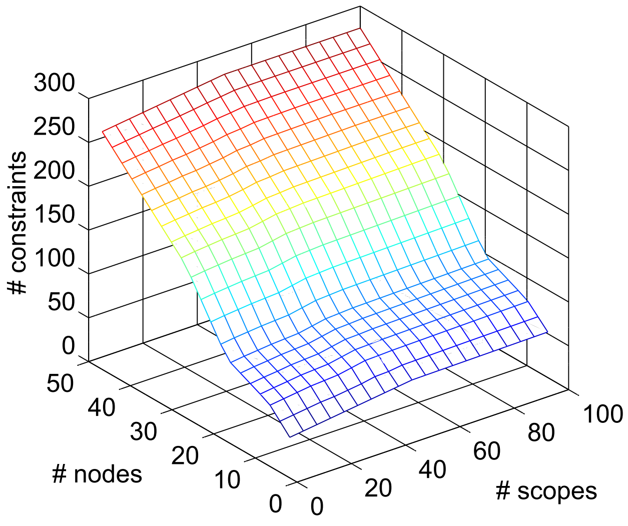


**Fig. 11** Generated constraints according to number of nodes and scopes for *Philosopher Dining* Mapping

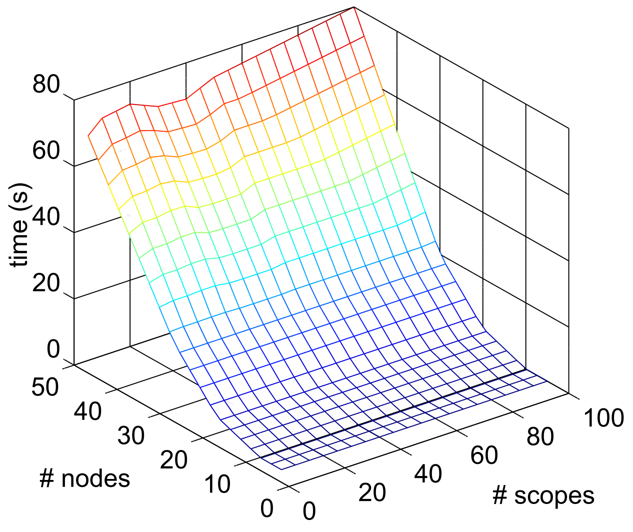


**Fig. 12** Resolution time according to number of nodes and scopes for *Philosopher Dining* Mapping

is NP-complete. Hence the mapping process can be performed simultaneously for a lot of SC but only on a WSAN partition in order to limit the number of nodes. This limitation is not strongly penalizing because the deployment process is already partitioned by the composed agents.

## 5.4 Implementation

The EMMA middleware is implemented on Contiki OS by a set of standalone module applications. It is composed of the Erbium CoAP server-client, a File System (FS) for resource management with a JSON parser and a preprocessing engine for variable parsing. These modules communicate by an event messaging engine in order to take advantage of micro-controllers sleeping mode for energy saving purposes. The different component footprints are provided in Table 1.

**Table 1** Memory footprints of EMMA modules

| 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 |
| emma-service-system | 60 Bytes | 2845 Bytes |
| emma-service-local | 10 Bytes | 576 Bytes |
| emma-service-agent | 210 Bytes | 6586 Bytes |
| Total (EMMA) | 1.9 KBytes | 31.4 KBytes |
| Contiki OS | 6.6 KBytes | 71.1 KBytes |
| Total | 8.6 KBytes | 102.4 KBytes |

The EMMA mapper is a JAVA application with Human Computer Interface (HCI) to design Service Choreography (SC) and to print graphically their mapping on the WSAN. It is composed of a CoAP proxy based on Californium framework to collect network and service informations and a PBO solver based on SAT4J framework. The different results have been experimented on COOJA simulator using ATMEL avr-raven board composed of a radio transceiver IEEE 802.15.4 and an ATmega1284PV micro-controller 8-bits at 8 Mhz with 16 KBytes RAM and 128 KBytes Flash memory.

## 6 Conclusion

This paper presents EMMA framework which provides a set of tools to design distributed architectures for Responsive Environments (RE). Its Resource Oriented Architecture (ROA) provides an abstraction layer between the WSAN and its networked applications. The different services provided by the nodes are interconnected by a distributed Publish–Subscribe mechanism in order to form a Service Choreography (SC). Instead of proposing a new model, this work is original through the adaptation of well-known mathematical models to design and validate SC. Moreover, the framework implementation is based on standard technologies in order to propose an automatic solution to map, deploy and execute SC over heterogeneous WSAN. Hence this framework should be a first step toward distributed IoT applications with self-reconfiguration features.

**Acknowledgments** The authors thank Dr. Courbin Pierre for the discussions and its reviews during the writing of this document and also Mr. Mardegan Nicolas for its contribution on the implementation of EMMA framework.

## References

1. Billington J, Wheeler G, Wilbur-Ham M (1988) Protean: a high-level petri net tool for the specification and verification of communication protocols. IEEE Transactions on Software Engineering 14(3):301–316. doi:10.1109/32.4651
2. Cherrier S, Ghamri-Doudane YM, Lohier S, Roussel G (2012) Services collaboration in wireless sensor and actuator networks: orchestration versus choreography. In: Symposium on Computers and Communications (ISCC), IEEE, Cappadocia, Turkey, pp 411–418
3. Cherrier S, Salhi I, Ghamri-Doudane Y, Lohier S, Valembois P (2014) Bec 3: Behaviour crowd centric composition for iot applications. Mobile Networks and Applications 19(1):18–32. doi:10.1007/s11036-013-0481-8
4. Costa P, Mottola L, Murphy AL, Picco GP (2006) Teenylime: Transiently shared tuple space middleware for wireless sensor networks. In: International Workshop on Middleware for Sensor Networks (MidSens), ACM, New York, NY, USA, MidSens '06, vol 1, pp 43–48. doi:10.1145/1176866.1176874
5. Delicato Flvia Coimbra PL Pires PauloF, da Costa Carmo Luiz Fernando Rust (2003) A flexible middleware system for wireless sensor networks. In: Endler M, Schmidt D (eds) Middleware 2003, Lecture Notes in Computer Science, vol 2672, Springer Berlin Heidelberg, pp 474–492. doi:10.1007/3-540-44892-6_24
6. Duhart C, Cotsaftis M, Bertelle C (2014) Wireless sensor network cloud services: Towards a partial delegation. In: International Conference on Smart Communications in NetworkTechnologies (SaCoNeT), IEEE, Vilanova i la Geltru, Spain
7. Dunkels A (2003) Full tcp/ip for 8-bit architectures. In: International Conference on Mobile systems, Applications and Services (MobiSys), ACM, San Francisco, CA, USA, vol 1, pp 85–98
8. Dunkels A, Gronvall B, Voigt T (2004) Contiki-a lightweight and flexible operating system for tiny networked sensors. In: International Conference on Local Computer Networks (LCN), IEEE, IEEE, Clearwater, Florida, USA, pp 455–462
9. Eduardo S, Germano G, Glauco V, Mardoqueu V, Nelson R, Carlos F (2004) A message-oriented middleware for sensor networks. In: Workshop on Middleware for Pervasive and Ad-hoc Computing (WMPAC), ACM, New York, NY, USA, MPAC '04, pp 127–134. doi:10.1145/1028509.1028514
10. Fok CL, Roman GC, Lu C (2009) Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Transaction on Autonomous and Adaptive Systems 4(3):16–26. doi:10.1145/1552297.1552299
11. Guinard D, Trifa V, Wilde E (2010) A resource oriented architecture for the web of things. In: Internet of Things (IOT), 2010, IEEE, pp 1–8
12. Hackmann G, Fok CL, Roman GC, Lu C (2006) Agimone: Middleware support for seamless integration of sensor and ip networks. In: Gibbons P, Abdelzaher T, Aspnes J, Rao R (eds) Distributed Computing in Sensor Systems, Lecture Notes in Computer Science, vol 4026, Springer Berlin Heidelberg, pp 101–118. doi:10.1007/11776178_7
13. Hadim S, Mohamed N (2006) Middleware: Middleware challenges and approaches for wireless sensor networks. IEEE Distributed Systems Online 7(3):1–1. doi:10.1109/MDSO.2006.19
14. Holliday M, Vernon MK, et al (1987) A generalized timed petri net model for performance analysis. IEEE Transactions on Software Engineering (12):1297–1310
15. Khedo K, Subramanian R (2009) A service-oriented component-based middleware architecture for wireless sensor networks. International Journal of Computer Science and Network Security 9(3):174–182
16. Ko J, Gnawali O, Culler D, Terzis A (2011) Evaluating the performance of rpl and 6lowpan in tinyos. In: Extending the Internet to Low power and Lossy Networks (IP+SN 2011), ACM, Chicago USA, vol 1, pp 193–208
17. Kovatsch M, Duquennoy S, Dunkels A (2011) A low-power coap for contiki. In: International Conference on Mobile Ad-Hoc and Sensor Systems (MASS), IEEE, Valencia, Spain, vol 1, pp 855–860. doi:10.1109/MASS.2011.100
18. Kuorilehto M, Hannikainen M, Hamaainen TD (2005) A survey of application distribution in wireless sensor networks. Journal Wireless Communication Network (EURASIP) 2005(5):774–788. doi:10.1155/WCN.2005.774
19. Kushwaha M, Amundson I, Koutsoukos X, Neema S, Sztipanovits J (2007) Oasis: A programming framework for service-oriented sensor networks. In: International Conference on Communication Systems Software and Middleware (COMSWARE), IEEE, Bangalore INDIA, pp 1–8. doi:10.1109/COMSWA.2007.382431
20. Liu W, Chen B (2011) Optimal control of mobile monitoring agents in immune-inspired wireless monitoring networks. Journal of Network and Computer Applications 34(6):1818–1826. doi:10.1016/j.jnca.2010.12.004, http://www.sciencedirect.com/science/

article/pii/S1084804510002158, control and Optimization over Wireless Networks
21. Moritz G, Golatowski F, Timmermann D (2011) A lightweight soap over coap transport binding for resource constraint networks. In: International Conference on Mobile Adhoc and Sensor Systems (MASS), IEEE, Valencia, Spain, vol 1, pp 861–866. doi:10.1109/MASS.2011.101
22. Rahman MA (2009) Middleware for wireless sensor networks: Challenges and approaches. In: Seminar on Internetworking, Helsinki University of Technology, Finland, vol 124
23. Rubio B, Diaz M, Troya J (2007) Programming approaches and challenges for wireless sensor networks. In: Second International Conference on Systems and Networks Communications, ICSNC 2007., pp 36–36. doi:10.1109/ICSNC.2007.63
24. Russell N, Arthur, van der Aalst WMP, Mulyar N (2006) Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22
25. Vasseur JP, Dunkels A (2010) Interconnecting Smart Objects with IP: The Next Internet. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

**Clément Duhart** received two M.Sc. degrees in 2012 from the University of Pierre et Marie Curie Paris VI in Artificial Intelligence and Decisions and from ECE Paris Graduate School of Engineering in Embedded Systems. He is received his Ph.D. degree in Computer Science at Le Havre University in 2016 applied on Organic Computing for Wireless Sensor and Actor Networks.

**Pierre Sauvage** received his M.Sc. degrees in Information Systems in 2015 from ECE Paris Graduate School of Engineering. He is currently employed in industry for design and deployment of large scale architectures for Big Data and Internet of Things applications.

**Cyrille Bertelle** is full professor in Computer Sciences in Normandie University, Le Havre, France since 2005. His activities concern complex systems modelling: their conceptual formalization, their distributed implementation and their applications in various domains: ecosystems, game theory, logistics, urban dynamics and territorial intelligence. He focuses his studies on emerging computing using collective intelligence methods and complex networks. He is co-founder of a regional institute on complex systems in Normandie (ISCN) and co-representative of Complex Systems Digital Campus (CS-DC) UNESCO UniTwin. He is currently Vice President for Research and Development in Le Havre University.