IDM: An Inter-Domain Messaging Protocol for IoT

David Villa*, Oscar Aceña*, Felix J. Villanueva*, Maria J. Santofimia*,
Soledad Escolar†, Xavier del Toro García‡ and Juan Carlos Lopez*
*School of Computing Science, University of Castilla-La Mancha, Ciudad Real, Spain.
†Institute of Technology and Information Systems, Ciudad Real, Spain.
‡Institute of Energy Research and Industrial Applications, Ciudad Real, Spain.
Email: {david.villa, oscar.acena, felixjesus.villanueva, mariajose.santofimia,
soledad.escolar, xavier.deltoro, juancarlos.lopez}@uclm.es

***Abstract*—The Internet of Things (IoT) paradigm envisions a world of interconnected objects in which every object or *thing* can be univocally addressed and accessed, independently of its inherent technology and location. To the date, most of the state-of-the-art solutions proposed to realize the IoT vision employ a cloud-based approach. This means that, independently on where objects are physically located, their interconnection relies on a cloud-based solution. In a daily life situation it is like requesting a call-center operator to get us in contact to someone that is nearby us. Common sense tells us that it is not only simpler, but more efficient, to directly talk to him/her minimizing delays, failing points, and unnecessary intermediaries. This paper presents a protocol for IoT, named IDM (Inter-Domain Messaging), intended to overcome the limitations of cloud-based solutions for IoT. The IDM approach consists in offering an abstraction layer upon which different technologies can interact in a seamless and efficient manner. This paper also provides an experimental validation consisting in the implementation of a prototype interconnecting five different domains, with technologies like ZigBee, Bluetooth, RS485 or WiFi.**

I. Introduction

Actuation for the Internet of Things (IoT), more specially for Industrial IoT, is gaining attention as it remains challenging to effectively operate on IoT objects under requirements of real time, scalability, security, or interoperability [1] [2]. Traditionally, efforts have been addressed to solve the *sensing* aspect overlooking the importance of supporting *acting* and *smart behavior* capabilities. In other words, we can say that, to date, IoT has mainly been applied to data collection purposes [3]. When applied to actuation or smart behavior, there exist ad-hoc solutions in which gateways assume the responsibility of translating cloud invocations into actuator commands [4]. Needless to say, such solutions are highly coupled to the technology and the purpose they are devoted to, which is not what the IoT vision is on about. The reason why actuation still remains a challenge is because seamless integration of technologies is far from being a reality [5]. Despite efforts like CoAP [6] or ZigBee [7] connectivity issues still remain unsolved mainly because these type of solutions are built over IP. Resorting to IP is not always the best solution because of restrictions imposed by scarce-resource devices, timing or constrained networks.

The IoT paradigm is mainly grounded in the capability of the network to support seamless interoperability among *objects*, independently of their technology or network details [8]. A revision of the state-of-the-art solutions for IoT shows that most of them are mainly based on cloud-like approaches. In these solutions, the management of both the IoT network and the data/information is delegated to a remote server. In this scenario, the Cloud typically assembles a set of repositories in which sensors publish their values, and then the server applications retrieve and understand these data and take a centralised decision. Such architecture enables the interoperability between different IoT networks only at the cloud level without any direct communication between the information sources, consumers or actuators.

By itself, the use of cloud-based solutions for IoT is not bad. It makes sense when data collected from different sources has to be aggregated and therefore a comprehensive view is required. Also, the Cloud offers a simple way (although not the most efficient one) of addressing the heterogeneity (in terms of devices and networks) in IoT. However, there are situations in which the interconnection of IoT objects should be accomplished in a peer-to-peer manner, without having to resort to third-party intermediaries. This new trend is being referred as an effort to *bring the Cloud to the Edge* [9] [10] as a way of overcoming some of the limitations of the cloud-based approaches, as listed underneath:

- High latency. Delegating all communications to cloud-based applications, working as intermediaries, entails unnecessary delays. On the contrary, a more efficient approach would consist in the producer handling the data directly to consumers (actuators, for example). The high latency involved in cloud-based solutions might have severe consequences in actuation and device management.
- Not a network. We cannot say that this cloud-based solution is really enabling a network of interconnected objects. On the contrary, this type of solution does not support one-to-one communication between the platform nodes. For this reason, we claim that cloud-based IoT platforms, by itself, are not actually *networks of interconnected objects*.
- Many simple services support their responses on local information, without needing aggregated or historic data. For these services, it makes no sense for the information sources to communicate the local data to a remote repository, and then transfer them to the service which will

conveniently command the actuator. The most sensible approach is to directly connect the information source to the service that employs it (like watering and humidity sensors).

- Low fault tolerance. Using the Cloud for transferring data between two nearby objects might increase the risk of being affected by a temporal network unavailability (and also by privacy concerns). If a direct connection of objects is supported, these in-site services will not be affected.

All these inconveniences eventually lead to *poor* solutions to address actuation and smart behaviour in IoT. Ideally, the aforementioned limitations of current solutions for IoT could be overcome by a universal network protocol, meaning that all network parties should implement that protocol (this is the 6LowPAN approach). This does not, however, seem plausible in the light of devices being manufactured to date (not all support IP). Additionally, non-IP protocols are integrated through stateful bridges. These bridges have to be hand and hard-coded to address the nodes (different protocols) after the bridge. This is a big limitation for dynamic deployment, configuration and fault tolerance.

An alternative consists in providing a mechanism for *object* interconnection and awareness, aimed to be totally decoupled from the communication technology aspects. This feature is especially important for industrial protocols like CAN bus, RS485, I2C, etc., because these protocols are normally implemented by scarce-resource devices in which the implementation of the TCP/IP stack is not feasible. Moreover, these protocols have not been designed considering domain interconnection supports.

Our vision is that every IoT network should support the construction of real "inter-networks", thus horizontal frameworks, enabling the collaboration of different-technology devices that can intercommunicate in a direct and symmetric manner, without requiring any transformations or the use of intermediaries or handlers.

In this vision, every Internet resource or IoT "Thing" can be seen as part of the same inter-operable environment. Figure 1 depicts how we envision the network topology, accordingly to the IoT paradigm exploiting an "Inter-Domain layer". This layer is based on a virtual-network protocol, called Inter-Domain Messaging (IDM). IDM supports peer-to-peer communication between devices connected using any network technology. It is shaped to be similar to IP, since it enables the addressing of connected elements seamlessly integrating heterogeneous networks, thus enabling the IoT network interoperability at every logical point of the proposed architecture and the deployment of distributed in-network algorithm. In this scenario, IDM routers are the managers of the interaction and inter-operation between different protocols at different levels.

II. Background

IDM is built upon conceptual ideas discussed in the following subsections.

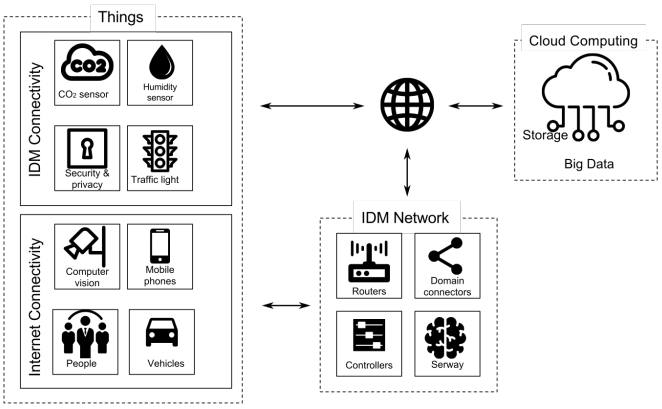

Fig. 1. Sensor and Actuator and Inter-Domain layer

A. Cross layering

Generally, every layer of a conventional protocol stack is conceived with a very specific purpose in mind. For example, in the OSI model [11] the data-link layer is in charge of enabling neighbor communication whereas the network layer supports communication between any network nodes. The fact that the protocol stack is implemented by the operating systems implies that, independently of the application, all the features of the whole protocol stack will have to be included. The local delivery, for example, despite not requiring the network-layer features will have to consider them because it is how a stack-based protocol works.

IDM is envisioned to work as an abstraction layer that interconnects different network domains. However, despite using the term *layer*, IDM is not coupled with any layer of a stack-based protocol. As it can be seen in Figure 2, the IDM layer can be projected over any of the host-to-network layers (transport, inter-net, or data link). In fact, IDM can use the services provided at the different layers of the stack. For example, IDM is capable of delivering a message to a LAN neighbor by encapsulating the message directly over Ethernet. This saves the router from having to handle layers that do not add any additional feature. For example, when sending a message to an Ethernet neighbor, the IP encapsulation is useless.

The main disadvantage of this traditional approach of stack-based protocols comes into light when limited-resource devices have to implement such protocol stack. Normally, the most common solution consists in weight-up the most complex or dynamic features, generating a light version of the protocol stack, like microIP or 6lowPAN. The main limitation of this solution is that, despite being conceived as a light version, it is, usually, a different protocol stack. For this reason, bridges are needed to translate messages from the light version to the standard one. This solution basically moves the overhead from embedded devices (resource constrained) to the bridges.

IDM offers a more efficient way of doing so by encapsulating its messages directly over the needed layer. For example, Figure 3 represents the interconnection of a ZigBee domain

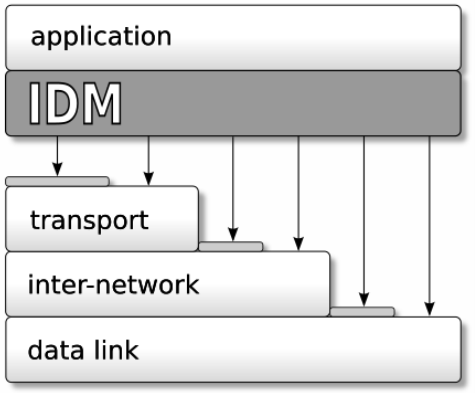

Fig. 2. IDM encapsulation over any layer of the stack

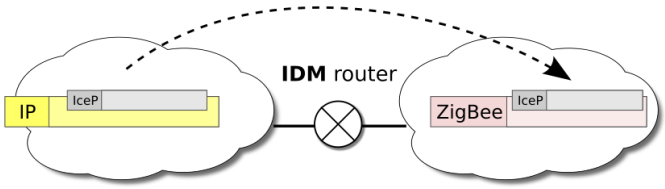

Fig. 3. IDM router forwarding an IDM message

with a TCP-based one. In the ZigBee network, IDM can encapsulate its messages directly over the data-link layer, rendering unnecessary the network and transport layers or the need of ad-hoc bridges that do the protocol conversion. On the contrary, the IDM router is the one that encapsulates the input IDM message (ZigBee) into an output message (TCP) accordingly to the next hop and without applying any change or modification to the original IDM message.

Additionally, thanks to the cross layering feature, IDM can also avoid unneeded or unwanted features which, sometimes, increase complexity. In this sense, if only a specific feature of an upper layer is required, like delivery assurance, there is no need to unnecessarily support the rest of features. On the contrary, a simple confiability protocol like RATP [12] could be implemented directly over the data-link layer.

This is the idea that Figure 2 tries to capture. IDM is capable of being encapsulated in any protocol of the stack. This feature therefore enables IDM to provide additional functionality over the underlying layer. Figure 2 uses the gray bars over the stack layers to represent that any additional functionality can be provided there, like the confiability over the data-link layer. The only requirement is that the message recipient (normally an IDM router) also implements the same extension.

It is important to highlight that this additional functionality is only provided where it is needed and not elsewhere in the network. For this reason, only the parties involved in the communication need to know about the use of such extensions.

B. IoT objects

The different layers of a network stack hold specific information to "address" different conceptual entities (IP addresses, TCP ports, HTTP resources, etc.). Under the IoT vision it would make sense to identify objects (rather than any of the aforementioned entities) and address them, individually. Ideally, every object will provide an API that will determine the terms of the interaction.

Under the IoT vision, an object or *thing* is the minimal-addressable unit. In constrast, in IP the minimum-addressable unit is the host. The main implication of having an object network, rather than a host network, is that different objects can be addressed, in a peer-to-peer manner, independently on whether they are physically deployed in the same or different resource. This means that we can address different object (services, for example) in the same sensor node since each of the services are considered individual objects.

C. Virtual networks

A virtual network is a distributed application in which the services somehow resemble the devices of a physical network. The fact that IDM is a virtual network entails a flexibility in terms of software deployment. For example, when a router is required at a certain location where software can be deployed, it is more convenient to do so by software rather than adding new hardware.

An additional implication of being constructed over a virtual network is that the existing network infrastructure can be reused (without having to deploy specific network hardware). Moreover, the physical network topology is also a detail that can be overlooked.

D. Software-defined networking

The main implication behind the idea of a Software-Defined Networking (SDN) [13] is the possibility of using production networks for experiments and tests of new protocols, routing mechanisms, etc. When this idea was implemented, it was soon evident that it offered a tremendous flexibility for different type of network users who could adapt the network behavior to their preferences.

This flexibility is achieved thanks to the interconnection devices that replace gateways, routers and commuters. The interconnection devices not only offer these functionalities but they can also offer more advanced functionalities. It is the network user who determines the routing rules stating.

The way how these interconnection devices are managed has inspired the idea of the IDM routers. Based in the OpenFlow [14] philosophy, the following listing shows the simplified interface for remote management of IDM routers.

```
module Routing {
    enum ActionType {Forward, Drop, Log};

    struct Matcher { string address; };

    struct Action {
        ActionType action;
        dictionary<string, string> arguments;
    };

    struct Flow {
        int identifier;
        Matcher m;
```

```
        sequence<Action> actions;
    };

    interface RouterAdmin {
        void flowAdd(Flow f);
        void flowDelete(int identifier);
        sequence<Flow> flowList();
    };
};
```

In our case, the IDM router holds a set of flow specifications (installed by the controller) determining what to do with every message. A flow is comprised of a *matcher* and a set of actions. When a message satisfies a *flow matcher*, the router executes the corresponding actions.

The most basic action is to forward the message to the next hop or destination. Other actions could be to drop the message, log the event, etc.

III. Protocol specification

This section will describe the technical details of the IDM protocol. It is important to highlight that IDM does not replace IP or any other standard protocol. On the contrary, it supports device interconnection, independently of their network technology. Moreover, IDM enables the creation of a tailor-made network without the overheading of using all the existing underlying protocols. This solution, indeed, just uses what it is required for the specific purpose at which it is being applied.

A. Addressing and routing

Any network protocol intended to provide peer-to-peer connectivity requires a common and unique addressing scheme. In other words, every node of the network has to implement the same addressing rules (address size, specific-purpose addresses, etc.). Addressing schemes like IP or IPv6 implements a hierarchical approach based on the use of network masks. These masks can be used to divide any network into smaller sub-networks. The division criteria might respond to different administrative reasons.

Similarly to IP, IDM uses hierarchical numeric addresses (expressed as hexadecimal strings) to identify source and destination. The only difference is that the IDM addresses are object identifiers instead of host identifiers. Hence, in the IDM address the prefix (net-id) determines the network location whereas the suffix (object-id) identifies the object itself, independently on whether the objects are physically located in the same computational device or in a different one.

The addressing scheme implemented by IDM has very important implications for object migration. Objects can be migrated from one node to a different one (as long as they are in the same network), in a very straightforward manner, as the IDM router will eventually know the new node address. It is important to highlight that the IDM address is not coupled to the node and that there is no correspondence between the object and the node address. For this reason, a new object can use its previous IDM address on a new node.

The current implementation of the IDM protocol adopts a manual address-assignment approach. Every object register itself in the local router and, automatically, the router knows the underlying addressing scheme. The IDM address, that works as a logical address, should be mapped to the specific node address, that plays the role of a physical address in a typical protocol stack. For example, in a ZigBee network the IDM addresses will be mapped into ZigBee hardware addresses. The IDM router will eventually use these *hardware addresses* when delivering message to these objects. IDM routers are, therefore, capable of managing very different protocols: TCP, UDP, Bluetooth, ZigBee, etc., with the sole restriction of having the specific hardware transceiver.

Finally, another interesting feature of the IDM addressing scheme is its implementation of variable-length addresses (from 8 to 128 bits). Constrained networks can benefit from having small addresses, saving message space, that can be therefore employed for the message payload.

B. Message format

The wide spectrum of IoT applications requires more flexible communication mechanisms that, under certain circumstances, support the omission of optional features for performance purposes. IDM considers the following request-message format:

- **One-way message**: This is the most basic message, employed for those situations in which an answer is not required. For example, a sensor notifying its state or an actuator receiving a command. Under this approach there is no reply nor error notification. This message has the following fields:
  1) *Object identifier*: The destination IDM address.
  2) *Method name*: The name of the remote object method, according to the user API.
  3) *Parameters*: The invocation arguments, if any.
  4) *Hop count*: The maximum number of allowed routers to reach the destination.
- **Two-way message:** This message is employed whenever the source expects an answer from the destination or an error report. In addition to the previous format fields, the *Source Address* has to be added to the message.
- **QoS-enabled message:** Advanced-routing decisions are supported on one of the following fields:
  1) ToS profile: This is a number that identifies a type of service. The router will provide a tailored behavior based on that identifier. For example, when a high-priority message is handled to the router, it will try to prioritize it over others. The only requirement is that the router should know, beforehand, how to react to that identifier.
  2) QoS profile: This is an alternative way of stating the expected router behavior. The only difference is that this field includes the information regarding the message management. This is more convenient for sporadic messages.

IV. Experimental validation

A. Description of a validation prototype

IDM can be used to tackle different problems. Nevertheless, the proposed prototype has been specifically designed to

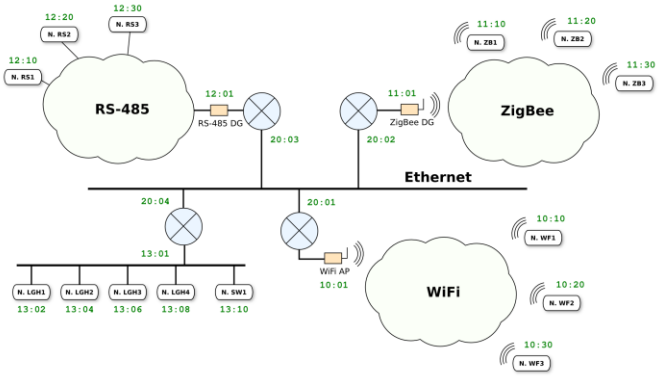

Fig. 4. Prototype IDM topology

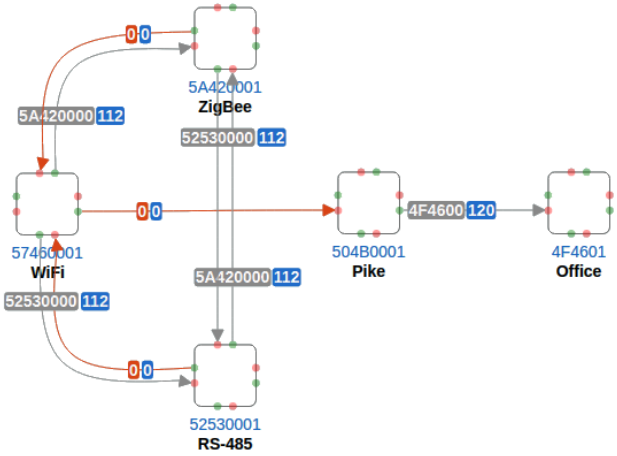

Fig. 5. IDM Controller (actual user interface)

evaluate its homogenization capabilities. As a result of such homogenization process, IoT objects are considered virtually equal inside an IDM infrastructure, although their communication and architecture details are different.

This prototype encompasses 5 domains:

- RS: 3 RS-485 Moth nodes
- ZB: 3 ZigBee Moth nodes (Arduino FIO + XBee)
- WF: 3 WiFi Moth nodes and one SonOff (ESP8266)
- ITSI-office: 5 SonOff nodes (ESP8266)
- The Internet (just for communication purposes)
- The Node-RED cloud

The first three domains are located at our laboratory, referred as ARCO, and the fourth domain at the ITSI laboratory.

We have designed the Moth prototypes and, despite having different controllers and network interfaces, they all have the following sensors/actuators:

- Relay for controlling a power load
- Red, green, and yellow LEDs
- PIR sensor (for detecting presence)
- LDR sensor (light sensor)
- Temperature sensor
- Microphone

Every domain has its own IDM router. The IDM routers for the ZB, RS, and ITSI-office domains are run on a Raspberry Pi whereas the router for the WF domain is run on a conventional PC. The Figure 4 shows the logic topology of the described prototype.

B. Clients and objects

Both sensors and actuators can work as clients, therefore sending invocations.

- Sensors invoke a designated receptor when their state change (a new value has been read).
- Actuators can also invoke a third party when their state change as result of an incoming invocation.

Sensors and actuators are also acting as objects, and can, therefore, receive invocations. Both of them count on an interface that will provide a method for setting the IDM address of the object that is to receive such state changes (its *observer*).

C. Routing

IDM routers forward messages based on the hierarchy addressing, similarly to how an IP router works. However, its management complies with the SDN approach, highly inspired by OpenFlow. IDM routers are also IDM objects that share a management interface. The IDM Controller is in charge of implementing the routing scheme by adding description flows into the routers.

A description flow is comprised of a predicate and one or more actions. When the predicate is satisfied, the router executes the action/s. A *forwarding* flow is quite simple: if the input message is targeting an address whose prefix matches the predicate, the action is executed, which eventually involves forwarding the message to the indicated address.

This way of managing routers is extremely flexible and, most of the times, simplifies their implementation. It entitles routers to do things like:

- Discarding certain type of messages (as a firewall would do)
- Informing the Controller that an incoming message does not match any rule
- Associating a time-to-live to rules
- Recording errors and statistics
- etc.

Figure 5 corresponds to a snapshot of the graphical interface of the IDM Controller. This supports an intuitive and visual editing of the routing flows.

D. Homogeneous integration

Once the infrastructure is deployed, objects can address invocations between themselves by only knowing their IDM address.

For example, the object that controls the PIR sensor on the ZB1-Moth node (address 10:17) can be configured to send its state changes to one of the LEDs of the RS2-Moth node (for example, the red one has the address 12:22). This

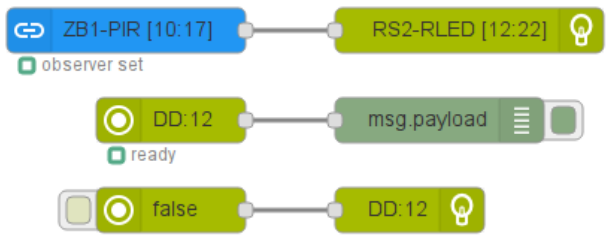

Fig. 6. Node-red IDM

configuration will result in switching on the red LED when the PIR sensor detects presence. This proves that the invocation was originated in a device connected to a ZigBee network, going through a TCP/IP network, reaching the device (red LED) connected to an RS485 network.

It is important to highlight that objects can also be implemented in conventional PCs, smartphones, or any other computing platform that we can consider. In all these cases, the peer-to-peer communication works in the same way.

Additionally, we have created Node-RED [1] components that supports the visual configuration of event receptors (see Figure 6). Moreover, clients and objects can be virtually created, hosted in the Node-RED server and therefore constituted as a new IDM domain connected to the infrastructure.

The previous figure represents three different Node-RED flows, with the following meaning:

- The first flow represents the configuration of the object, with address 10:17, which corresponds to the PIR sensor on the ZB1-Moth node, in the ZigBee domain. This flow is being configured to forward the state-change notifications to the object with address 12:22 (the red LED of the RS2-Moth node, in the RS485 domain).
- The second flow involves the generation of a virtual object, with address DD:12, which is registered in the local IDM router. This object can be invoked from any point of the IDM inter-net. When the invocation is received, Node-RED prints out the output on the screen.
- The third flow can be used to send invocations with a boolean value to that very-same object with address DD:12. Similarly, invocations could be sent to any other object (real or virtual) in any other place just by stating its IDM address.

V. Conclusion and Future Work

This paper describes a protocol for enabling the IoT vision. The main feature of the IDM protocol is its capability to directly interconnect objects from different, and initially non inter-operable, network technologies. The main strength of this protocol is that it can be directly deployed over the existing network infrastructure. The only requirement is the deployment of the IDM routers in the interconnection nodes.

[1]Node-RED: http://nodered.org/. Accessed: 2016-11-14.

The experimental validation shows that this a viable solution for IoT, characterized by its low latency and its support for peer-to-peer communication. In contrast to cloud-based solution, our approach provides a flexible solution for enabling actuation in IoT.

Despite the fact that no specific security aspects have been considered in this work, the underlying technologies that IDM uses provide basic security features such as SSL, WSS, etc. Moreover, ciphering extensions can be easily added thanks to the cross layering support of the protocol.

Future works should be addressed to consider transport of other types of messages, like HTTP or CoAP.

Acknowledgment

This work has been funded by the Programme for Research and Innovation of University of Castilla-La Mancha, co-financed by the European Social Fund (Resolution of 25 August 2014) and by the Spanish Ministry of Economy and Competitiveness under project REBECCA (TEC2014-58036-C4-1-R) and by the Regional Government of Castilla-La Mancha under project SAND (PEII_2014_046_P).

References

[1] J. Delsing, *IoT Automation: Arrowhead Framework*. CRC Press, 2017.
[2] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, "Enabling iot automation using local clouds," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 502–507.
[3] P. P. Ray, "A survey of iot cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 12, pp. 35 – 46, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2314728816300149
[4] P. Persson and O. Angelsmark, "Calvin merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210 – 217, 2015, the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915008595
[5] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb 2017.
[6] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.
[7] A. Zigbee, "Zigbee specification," *ZigBee document 053474r13*, 2006.
[8] R. Want, B. N. Schilit, and S. Jenson, "Enabling the internet of things," *Computer*, vol. 48, no. 1, pp. 28–35, Jan 2015.
[9] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2014, pp. 346–351.
[10] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, "Edge computing enabling the internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 603–608.
[11] D. Wetteroth, *OSI reference model for telecommunications*. McGraw-Hill Professional, 2001.
[12] G. Finn, "Reliable asynchronous transfer protocol (ratp)," Internet Requests for Comments, RFC Editor, RFC 916, October 1984.
[13] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, vol. 2, pp. 2–6, 2012.
[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746