Fog computing for the integration of agents and web services in an autonomic reflexive middleware

Manuel Sanchez, José Aguilar, Ernesto Expósito

Service Oriented Computing and Applications
https://doi.org/10.1007/s11761-018-0238-0

Fog Computing for the integration of agents and web services in an autonomic reflexive middleware

Manuel Sánchez, Jose Aguilar, and Ernesto Exposito

***Abstract*— Service Oriented Architecture (SOA) has emerged as a dominant architecture for interoperability between applications, by using a weak-coupled model based on the flexibility provided by Web Services, which has led to a wide range of applications, what is known as cloud computing. On the other hand, Multi-Agent System (MAS) is widely used in the industry, because it provides an appropriate solution to complex problems, in a proactive and intelligent way. Specifically, Intelligent Environments (Smart City, Smart Classroom, Cyber Physical System, and Smart Factory, among others) obtain great benefits by using both architectures, because MAS endows intelligence to the environment, while SOA enables users to interact with cloud services, which improve the capabilities of the devices deployed in the environment. Additionally, the fog computing paradigm extends the cloud computing paradigm to be closer to the things that produce and act on the intelligent environment, allowing to deal with issues like mobility, real time, low latency, geo-localization, among other aspects. In this sense, in this article we present a middleware, which not only is capable of allowing MAS and SOA to communicate in a bidirectional and transparent way, but also, it uses the fog computing paradigm autonomously, according to the context and to the system load factor. Additionally, we analyze the performance of the incorporation of the fog-computing paradigm in our middleware and compare it with other works.**

***Index Terms*— Fog computing; Cloud computing; SOA; MAS; Intelligent Environment; Integration.**

I. INTRODUCTION

The new advances in information technology, in domains like cloud and ubiquitous computing, allow us to exploit all computation tools (devices, software, etc.) as a whole, in order to define systems at a high abstraction level. In this way, it is possible the development of new domains like Ambient Intelligence (AmI). Consequently, some of the biggest challenges in developing an AmI is how to dispose of the enormous and multiple sources of information and services in a given time and in the right way. Cloud Computing is an alternative in this context, to form a network for the storage of large amounts of data and for the utilization of services in the Internet. Particularly, in previous works, we have developed a middleware based on MAS, called AmICL, to support smart classrooms [1]–[3]. This Reflective Middleware allows managing an Intelligent Learning Environment (IE). The middleware proposes five levels, for the management of the multi-agent's community, the access to services, and the different components (software and hardware) of the smart classroom. Later, In [4] we have proposed an Autonomic Reflective Middleware for Smart Cities, called MiSCi. MiSCi is based on AmICL, and its architecture is based on web services, allowing its services to be consumed by the applications (in our case, agents), aware of context or not. Agents can create temporary or permanent emerging ontologies according to the context, which allow them solving particular situations.

From this perspective, AmICL and MiSCi (and other intelligent environments like [5]–[8]) mix the cloud computing paradigm with multi-agent systems (SOA-MAS), which makes necessary to deal with the integration of agents and web services, so they both can discover and invoke each other (communicate and interoperate) in a transparent way. However, because SOA and MAS use different standards and specifications [9]–[11], their communications are not possible in a natural and direct way. While a MAS uses FIPA protocols for communication, and more specifically, the FIPA-ACL language [12]–[14], in SOA the SOAP [15] communication protocol is generally used, which is a standard that defines how objects in different processes can communicate using messages written in XML [16], with a transport protocol such as HTTP. Thus, some solutions have been proposed that allow the integration of both architectures [17], [18], in such a way that they can be discovered and invoke instances of the other transparently, to take advantage of both technologies. Particularly, we have proposed a new architecture [19], which deals with the MAS-SOA integration, and allows agents and services to communicate in a bidirectional and transparent way.

On the other hand, in an intelligent environment, where there are multiple nodes and applications that interact to offer services to people, the high quality of services, real time and low latency, are of great importance. In a smart city, more and more data are generated each day, so it is very important to process it quickly in order to allow real time applications to

M. Sánchez is with Universidad Nacional Experimental del Táchira, San Cristóbal, Venezuela and with Univ Pau & Pays Adour/E2S UPPA, Laboratoire d'informatique de L'universite de Pau et des Pays de L'adour, EA3000, 64000, Pau, France (email: mbsanchez@unet.edu.ve).

J. Aguilar is with Universidad de los Andes, Mérida, Venezuela, Escuela Politécnica Nacional and Universidad Técnica Particular de Loja, Ecuador (email: aguilar@ula.ve).

E. Exposito is with Univ Pau & Pays Adour/E2S UPPA, Laboratoire d'informatique de L'universite de Pau et des Pays de L'adour, EA3000, 64000, Pau, France (email: ernesto.exposito@univ-pau.fr).

Tepuy R+D Group. Artificial Intelligence Software Development. Mérida, Venezuela.

respond to user needs in lower time. In general, the deployment of a classical middleware for smart cities based on a cloud computing paradigm can become a problem for applications that require real-time response, mobility support and/or geo-distribution. Essentially, a new platform is necessary to meet these requirements, and the Fog Computing paradigm can be the solution.

Fog Computing is a distributed infrastructure, in which certain applications, processes, or services are managed at the edge of the network by a smart device, but others are still managed in the cloud. Fog Computing enables a new breed of applications and services, and there is a fruitful interplay between the Cloud and the Fog, particularly when it comes to data management and analytics [20]. Fog Computing can offer data, compute, storage and service to the end-user in the edge of the network. This paradigm has characteristics that makes it an appropriate platform for critical services and applications, such as those presented in a Smart City. The distinguishing fog characteristics are its proximity to end-users, its dense geographical distribution, and its support for mobility. Consequently, in [21] we have shown how the incorporation of a Fog layer in MiSCi can deal with the real time and low latency issues in a Smart City.

Particularly, this paper discusses how we combine the MAS-SOA integration architecture with the fog computing paradigm in the context of an AmI, to deal the issues discussed previously. In addition, it presents a study of the performance of our proposal, showing that it allows dealing with real time and low latency issues. Finally, this paper makes a comparative analysis with previous works, which combine the three paradigms (SOA, MAS and Fog).

This paper is structured as follows, Section II presents the related works, Section III exposes the MiSCi architecture and the sub-components used for the communication between agents and web services in the cloud, transparently. In Section IV, the experiments that show the performance of the MAS-SOA integration are shown, as well as the comparison with previous works. Section V extends the results of this work in the context of the industry 4.0, finishing with some conclusions.

II. RELATED WORKS

This section shows a brief description of the works that propose the integration of MAS, fog-computing and cloud computing. The first subsection describes those works that address the issue of integration or combination of MAS with the fog computing paradigm, while in the second subsection are presented those works that combine MAS with SOA platforms or with the cloud-computing paradigm.

A. Integration MAS-Fog Computing

Amadeo et al. [22] present the Cloud of Things (CoT) platform that deals with some challenges in the smart home domain by leveraging two groundbreaking concepts: Information Centric Networking (ICN) and Fog Computing. The proposal ICN-iSapiens model is a three-layered architecture where an intermediate (Fog) layer, consisting of smart home servers (HSs), is introduced between the physical world and the remote cloud, to support real-time services and hide the heterogeneity of Internet of Things (IoT) devices [23], [24]. The ICN Physical Layer includes all the edge devices (EDs), which deploy sensing and automation tasks in the smart home. They are usually single function resource constrained devices, like temperature or motion sensors, or light actuators. The Intermediate Fog Layer includes the HSs, which implement the application logics to monitor and control the house according to (i) user preferences, (ii) inputs from stakeholders (e.g., service providers and regulation entities), and (iii) dynamic context-related factors (e.g., the energy market price). The HS interacts with the EDs, via ICN, and with the remote cloud, through standard Internet connectivity. In addition to the software for NDN communications, each HS hosts the multi-agent application and the virtual object abstraction to perform Fog services. Each ICN-ED is represented in the Intermediate Layer as a virtual object (VO) that is a high level standardized description of the device's functionalities. VOs expose EDs by hiding their heterogeneity in terms of technological and networking details, and make their resources easily accessible to the Agents which, in turn, perform the application logics. Agents use VOs methods to pull monitoring and action tasks, and to be asynchronously notified about some events, i.e., when a resource value changes. Moreover, they may subscribe to complex events over groups of functionalities. To access information through the VO abstraction, VOs and Agents must be co-located in the same HS. Therefore, instead of transferring data to a central processing unit, ICN-iSapiens transfers processes (Agents) towards the EDs. The Remote Layer includes a remote Cloud platform, which addresses all those activities that cannot be executed by the HSs, e.g., tasks requiring high computational resources or long-term historical data. The data analysis executed by the Cloud can be used for different purposes, including (i) to optimize the Agents' operations and their behavior, or (ii) to support the demands of external consumer applications (e.g., collected data can be used by energy companies for reliable forecasting).

Peng et al. [25] present a study of the collaborative, self-adaptive configuration management of virtual machine resources in a multi-agent organization structure, with virtual machines as the unit of granularity. The multi-agent organization structure with virtual machine clusters consists of four types of agents. The vms-agents and apps-agents are collaborative agents. One vms-agent and one apps-agent are established in each virtual machine cluster. The vms-agent in a virtual machine cluster is responsible for implementing the virtual machine resource configuration requests issued by the vm-agent in the cluster, proposing dynamic parameter configurations for the relevant application systems/components that are operating on the virtual machines to the apps-agent, and sending the real-time utility information on various types of resources to the apps-agent. The apps-agent is responsible for implementing the application system/component parameter configuration requests issued by the com-agent in the cluster, proposing dynamic resource configurations for the relevant virtual machines on which the application systems/components are operating, and sending various types of real-time

information regarding the performance of the application systems/components (e.g. response time and throughput) to the vms-agent. The vm-agents and com-agents are learning agents that automatically adapt to the cloud computing environment. One vm-agent and one com-agent are established in each virtual machine and each application system/component in a cluster. The vm-agent learns the optimal decision regarding the virtual machine resource configuration, and monitors and provides feedback on various types of virtual machine resources in real time. Then, the com-agent learns the optimal decision regarding the application system/component parameter configuration, and monitors and provides various types of feedback on application system/component performance in real time. Experimental results show that their method improves the efficiency in the resource utilization, and accounts the interests of both cloud users and providers, while ensuring the maintenance of the application service level agreement.

Similarly, Giordano et al. [26] combine agent technology with the concept of Fog computing to design control systems based on the decentralization of control functions over distributed autonomous and cooperative entities that are running at the edge of the network. Also, they present Rainbow, an architecture that allows the development of smart city applications. Rainbow is a three-layer architecture designed to bring the computation as close as possible to the physical part. The bottom layer is the one that is devoted to the physical part. It encloses sensors and actuators, together with their relative computational capabilities, which are directly immersed in the physical environment. In the Intermediate layer, sensors and actuators of the physical layer are represented as (Virtual Objects) VOs. VOs offer to agents a transparent and ubiquitous access to the physical part due to a well-established interface exposed as API. VO allows agents to connect directly to devices without caring about proprietary drivers or addressing some kind of fine-grained technological issues. In summary, all the devices are properly wrapped in VOs that, in turn, are enclosed in distributed gateway containers. The computational nodes that host the gateways represent the middle layer of the Rainbow architecture. The upper layer of Rainbow architecture concerns the cloud part. This layer addresses all the activities that cannot be properly executed in the middle layer, for instance, algorithms needing knowledge, tasks that require high computational resources, or when an historical data storage is mandatory. On the contrary, all tasks where real time access to the physical part is required, are executed in the middle layer. Communication between the nodes connected with the physical part and the nodes in the cloud occurs by means of message exchange. Agents located in the cloud nodes act as intermediary between the Rainbow MAS and cloud analytics services. The features and capabilities provided by the Rainbow framework are shown by running intelligence algorithms, in order to realize Cyber Physical System (CPS) applications [27], [28], owning properties such as additivity, fault tolerance and self-reconfiguration, among others.

Finally, Mohamed et al. [29] discuss how the service-oriented middleware (SOM) approach can help to solve some of the challenges of developing and operating smart city services, using CoT and Fog Computing. They propose a SOM called SmartCityWare, for the integration and utilization of CoT and Fog Computing. SmartCityWare services and components involved in the smart city applications are accessible through the service-oriented model. The main purpose of SmartCityWare is to provide a virtual environment to be used to develop and deploy smart city applications. SmartCityWare consists of a set of services and a multi-agent runtime environment. All functions of the SmartCityWare are viewed as a set of services that can be used to build and support the execution of different smart city applications. These services are classified into core services and environmental services. Core services are those developed specifically for the core operations of SmartCityWare, such as the broker, security, service invocation, and location aware services, to provide overall control for the whole system. Environmental services provide access to services provided by one or more cloud service provider, services provided by multiple distributed fogs, and services provided by multiple IoT devices including sensors, WSN [30], actuators, cameras, cars, robots, etc. Fog services can be control, processing, storage, communication, streaming, configuration, monitoring, measurement, and management services. SmartCityWare services can be used by smart city applications available on the cloud, fog, or IoT devices, such as a car asking for a certain service from a smart city application available on the cloud. On the other hand, SmartCityWare middleware infrastructure utilizes software agents to provide flexible and expandable middleware services, or high-performance distributed service-oriented environments. The main functions of the agents are to deploy, schedule, and support the execution of the service codes in different fogs, in addition to manage, control, monitor, and schedule the available resources on a single fog, or on a set of related distributed heterogeneous fogs. For the experiments, they used three computers; one represents the cloud and two represent two fogs. In addition, they used WAN emulators among the machines, to introduce the effects of using long distances and/or the Internet to connect them.

B. Integration MAS-SOA or MAS-Cloud Computing.

In this subsection are shown the studies that combine MAS with the cloud computing paradigm, or some type of software oriented architecture. In this sense, Archimède et al. [5] propose Semantic-SCEPSOA, which is an architecture that combines three kinds of technologies. First, SOA to facilitate the identification and to form relationships of actors on the internet, allowing technical and operational interoperability by gathering functionalities of enterprise applications. Second, ontology technology to facilitate the understandings for the exchange of information. Third, a multi-agent model is used to elaborate the planning of the project. The semantic interoperability strategy, necessary to ensure a better understanding and interpretation of the exchanged information between heterogeneous systems, is based on ontologies. Reasoning mechanisms are needed by customers and producers to transform data described according to the global ontology in data expressed according to the local ontology, and vice-versa. S-SCEPSOA is organized around

three kinds of actors: the SOAregister containing information about the provided services, the SOAproducer proposing the service, and the SOAcustomer consuming the discovered services by invoking them at the corresponding SOAproducers. Interaction between the different actors is achieved via messages. The management process is achieved by three steps. The first one is the identification of partners, which concerns services publication, and discovery phases in the SOA context. The second step deals with instantiation of SCEP (supervisor, customer, environment, and producer) components on the SOAcustomer side, and connection to SOAproducers selected in the previous step. The SCEP agent gets from the customer database (CDB) the projects' description and creates the shared environment, as well as the customer and ambassador agents. One customer agent is created for each project, similarly, one ambassador agent is created for each SOAproducer. The third/last step concerns interactions and cooperation between SOAcustomer and its SOAproducers, through the SCEP instantiation, to manage multi-site production projects.

Al-Ayyoub et al. [8] present a dynamic resources provisioning and monitoring (DRPM) system, which is a MAS, to manage the cloud provider's resources while taking into account the customers' quality of service requirements, as determined by the service-level agreement (SLA). The provider's resources include a set of datacenters, each with a large number of physical machines. Each physical machine has specific characteristics (in terms of CPU, RAM, storage and BW) to host multiple virtual machine (VMs). Moreover, it has sensors to measure the utilization of its resources and the execution of the customers' applications on it. The application layer consists of a set of customers having several jobs possibly spanning multiple VMs. Each job is associated with specific performance goals specified in the SLA (e.g., time to complete their tasks, number of requests for specific periods). The multi-agent components include a global utility agent and a set of local utility agents. While the global utility agent has the classical role of the "central broker" allowing it to manage all the system resources, the local agents are assigned to each customer with the objective of improving the resource utilization without causing SLA violations. Based on a regression analysis of its history, the local agent can estimate the amount of resources that will actually be used without causing SLA violations. After reformulating the requests, the local agent sends the new requests to the global agent, which is responsible for the actual provisioning of the resources, by communicating with the supervisors of the physical machines in the different data centers under the cloud provider's control.

On the other hand, García et al. [31] propose a model for automatic construction of business processes, called IPCASCI (Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration), in order to facilitates the business process construction on cloud computing environments. It deals with the development of a proposal that facilitates the creation of such processes, in the form of web services, from other semi-automatic functional services. The process of business process construction is guided by a multi-agent architecture based on virtual organizations, which is able to implement the intelligent behavior needed for process management by using ontology. The components that integrate the architecture are 1) Cloud system, is the Platform of cloud computing on which is supported their proposal. The platform provides an environment for running and storage 2) Web services, are the web services existing in the architecture that will be used in the process of web service composition 3) UDDI register, is the Universal register system where the web services are registered. This allows an open access to the web services of the architecture. This way, the web services implemented in the architecture may be reused 4) MAS, this system supports functionalities to make the discovery and the composition of web services, by analyzing the semantic content introduced by the user in order to structure it in computable items 5) Ontologies, distinct ontologies modeling the semantic knowledge that can be included in the web services 6) BPEL (Business Process Execution Language) [32], [33] file to Store the composition of the web services that meet the requirements indicated by the user to obtain the desired solution. The discovery and composition of services are carried out when the user introduces the requirements of the web service to build, by means of an assisted system. Later, an automatic search (made in syntactic and semantic form) for the web services that include the requirements of the module, is carried out. Next, the Business Process Diagram (BPD) is then displayed by the assistant software to the user, because could exist several web services implementing a single activity. Finally, from the BPD, a composition BPEL is carried out in such a way that the service specified by the user is built, and can later be invoked.

Particularly, the researches presented above do not deal with SOA-MAS integration in a natural way, but instead each agent must directly apply the necessary transformations to be able to access cloud services. In this sense, Fuksa [17] developed a library, called JADE Gateway Library, which acts as a bridge between a MAS and the services that are deployed in an Enterprise Services Bus (ESB). The library is composed of two fundamental elements, the first one is the Jade-Gateway, which is responsible for managing the messages received from and to the ESB; the second one is the Gateway Agent, which main objective is to retransmit the messages between the MAS and the ESB. When an agent wants to consume a WS, it must send a message to the GatewayAgent, which in turn relays it to the ESB; On the other hand, when a WS needs to send a message to an agent, it must do through the EntryPoint ESB Service, which it will pass it to the Jade-Gateway, and then to the GatewayAgent, which will relay it to the requested agent. During the communication process, the Jade-gateway must perform SOAP-ACL and ACL-SOAP conversions, so that each party receives the message in the appropriate language. In the same way, Jade-gateway solves the problem of synchronous communications integration of web services with asynchronous communications of a MAS, blocking the GatewayAgent thread until obtaining the MAS response.

Finally, Pinto [18] proposes SoMAS (Service oriented MultiAgent System) as a completely service-oriented architecture, where both agents and web services use SOAP as a communication language. In SoMAS architecture, agents

publish their services in the UDDI, by sending the WSDL file corresponding to the service offered. When an agent or web service requires the use of a particular service, it must be looked up in the UDDI, and once found, it is established the communication with it through SOAP messages. This architecture leaves aside the FIPA standard, since the system does not use DF or ACL communication language, although the author indicates that both communication interfaces can be used, which means doing additional work.

Mainly, the works shown in this section combine MAS-Fog or MAS-SOA paradigms, however, they do not allow the combination of the three paradigms in a transparent way, such that agents can communicate with services in a transparent way, activating the fog paradigm conveniently (see in section VI the comparison of these works). In this sense, works that deal with the MAS-SOA integration issues in a natural way [17], [18], do not make use of the fog paradigm, so they are not able to solve several issues of cloud computing, such as mobility, real time, low latency, etc. In addition, these works have weaknesses, such as the fact of not being FIPA-compliant, or not allow bidirectional communication of agents and services. That is why in [19], [21] has been proposed a new architecture to solve those issues. In particular, this article studies the performance of the incorporation of the fog-computing paradigm to our middleware, which allows agents not only to communicate with services transparently, but also to use the fog paradigm in an autonomous way.

III. Extended Architecture Of MiSCi

This section describes the architecture of the Middleware for Smart Cities called MiSCi, proposed in [19], [21].

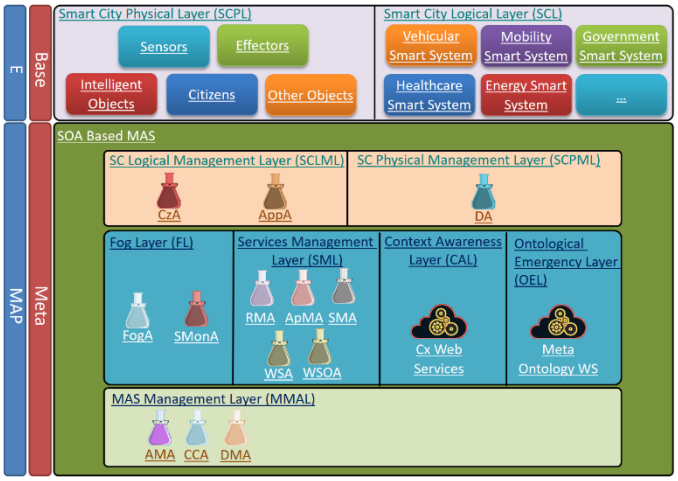

Fig. 1. Multilayer Architecture of MiSCi.

A. *Multilayer Architecture of MiSCi*

The core of MiSCi is based on a MAS composed by a multi-layer architecture. The agents of MiSCi use autonomic computing to deal with situations that arises in a smart city. They are capable of perceiving the interactions of the users by: a) Monitoring the environment (using the sensors and/or smart devices available on it). b) Analyzing the data to detect issues and find solutions (services to be offered to the user) according to the context. c) Planning and deploying the solution in the environment (using the effectors and/or smart devices), thus improving the activities carried out by the citizens of the city, with the main objective of improving the quality of life and comfort of the citizens, as well as performing a good management of the city.

The architecture of MiSCi contains nine layers (see Fig. 1). Each element of this architecture provides essential features to the middleware, enabling the ubiquity, the context awareness, the ontological emergence, smart decisions, fog and cloud computing, among others things (refer to [3], [4], [21] for more details). In general, the layers of MiSCi are:

1) MAS Management Layer (MMAL)
This layer is an adaptation of the FIPA standard *[34]* what defines the rules that allow a society of agents to coexist and be administered, encouraging the interoperability with other technologies. The Agents in this layer are: AMA (Agent Manager Agent), CCA (Communication Control Agent) and DMA (Data Management Agent), they are defined in detail in [34].

2) Service Management Layer (SML)
This is an essential layer in the architecture of MiSCi, because it makes possible the integration between the MAS and SOA paradigms in a bidirectional way. That means that agents can register, discover and consume web services in the cloud, and vice versa (agent's tasks are offered as web services). This feature makes possible to use the SaaS model of the cloud computing in MiSCi, which is fundamental in a Smart City. In this layer, the Services Management Agent (SMA), the Web Service Agent (WSA), the Web Service Oriented Agent (WSOA), the RMA (Resource Manager Agent) and the ApMA (Applications Manager Agent), are defined (see [35] for more details about these agents). In fact, SMA manages the UDDI, allowing agents and cloud services to discover each other. WSA allows agents and web services to communicate by translating messages in an appropriate way (is a proxy), and WSOA serves as a façade for agents, exposes their functionality's as services, and allows the communication from web services to agents of the platform (see Fig. 2).

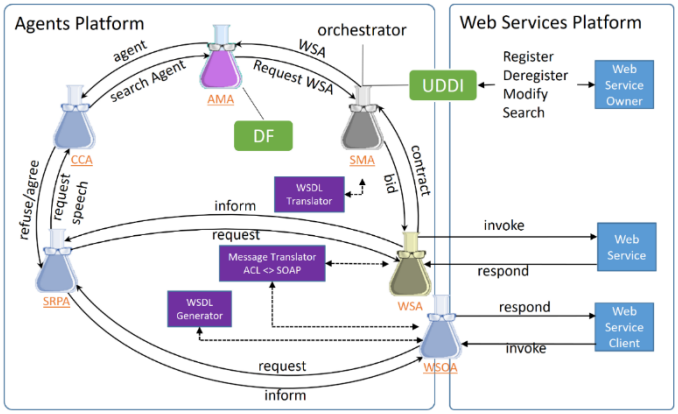

Fig. 2. Functionality of the SML.

3) Context-Awareness Layer (CAL)
The purpose of this layer is to offer context services, allowing agents of MiSCi to manage important information about location, time, and devices, among others. This information is managed in a cycle that is composed by the discovering and modeling of the context, the reasoning based on the context, and the distribution of the context. For this layer, we take as reference the services defined in [36], where is proposed a Context-Aware Reflective Middleware based on the Cloud Computing, with a range of services to manage the context information (see [4] for details about this layer).

4) Ontological Emergence Layer (OEL)
The objective of this layer is to provide a set of services with very specific tasks for handling ontologies. These services have been proposed in [37]. In this work has been defined group of services for Ontology Registration, for Ontology Searching, for the Integration of Ontologies, for Ontology Updating, among others. In addition, the structure of the meta-ontologies defined in this layer are the same proposed in the same work. They propose three meta-ontologies and the procedure to integrate them: a component meta-ontology, a context meta-ontology, and a domain meta-ontology. The meta-ontologies provide an adequate conceptual model of the context of a Smart City (see [37] for details about this layer).

5) SC Logical Management Layer (SCLML)
It is responsible for providing intelligence to the Smart City. This layer is where all the applications (software, virtual objects, etc.) and persons present in the Smart City are characterized. Each element/person corresponds to one agent (is an abstraction of it), which contains metadata that defines its properties. This layer contains agents like CzA (it characterizes each citizen in a Smart City) and AppA (it characterizes useful applications in the smart city, such as the Vehicular Smart System, the Healthcare Smart System, etc.). Those agents are coordinated and cooperate with each other, to take decisions that help to solve a particular situation, and to assist real people to perform their tasks in the smart city (this layer is detailed in *[4]*).

6) SC Physical Management Layer (SCPML)
It allows managing the physical devices in the Smart City. In this layer, all the physical elements of the environment are characterized thought the DA (Device Agent), allowing the interaction between agents and devices of MiSCi. Thus, each physical device is characterized by one DA (is an abstraction of it), which contains metadata that define its properties. Some of these physical devices are intelligent (smart objects), so that the properties of learning, autonomy, reasoning, among others, are critical to characterize them. This layer communicates with the real physical device that is in the SCPL layer, because is through SCPL that agents have access to the physical hardware of the devices (see [4] for details about this layer).

7) Smart City Physical Layer (SCPL)
This layer is the smart city itself. It is in this layer where all the physical components of the smart city are deployed, such as: a) Sensors, to capture the useful information for services and smart objects in the environment. b) Effectors, to modify the physical conditions of the environment. c) Smart Objects, which are components of the smart city that may adapt and respond to situations in the current context (see [4] for details about this layer).

8) Smart City Logical Layer (SCL)
This layer includes the main sub-systems of a smart city, which are responsible for managing the elements of the city in a global way, such as: Vehicular Smart System, responsible for control the traffic; Mobility Smart System, responsible of facilitating the mobility of citizens (public transport); Smart Healthcare System, in charge for facilitating the access to health services, among others. The agents of MiSCi can communicate with these systems through the AppA agents, because they characterize the applications of the smart city in the architecture. In this way, the global systems can be coordinated with the local systems, to meet the needs of the citizens in a given time.

9) Fog Layer (FL)
This layer enables the Fog computing paradigm in MiSCi. The agents in this layer help to decide whether the data might be processed locally or in the cloud, being the Fog Agent (FogA) responsible for this task. FogA uses a meta-ontology provided by the OEL Layer, some context information provided by the CAL layer, and some system information collected by the System Monitor Agent (SMonA), about the level of occupation in terms of processing and communication (bandwidth) of the agents and local web services. Using all that information, FogA can make a decision about whether or not the data should be processed locally or in the cloud (see [21] for details).

B. Motivation of the MiSCi extension.

In previous researches, we have shown how we can integrate both paradigms SOA and MAS [19], enabling MAS to communicate with cloud services in a natural way, and vice-versa. SOA-MAS integration is a crucial point in MiSCi, as well as in others agent based middleware, in order to take advantages of the cloud paradigm. Such is the case of the Cyber Physical Systems integrated with IoT, which cyber part can be associate with a MAS, and their service needs can be supplied in the cloud. In specific, the agents of the platform might use services in the cloud to process data (big data and data analytics services, linked data services, etc.), get recommendations, get context information, among other cloud services, facilitating intelligent decision making, context awareness, system recommendations, alerts, among other things.

The SML layer allows SOA-MAS integration on MiSCi, and it can be extended to other middlewares too. However, agents of the SML layer are not capable to deal with some issues related to cloud platform, smart environments, and IoT, like latency, real time, location aware, among others aspects (see [38], [39]). In this way, we have enabled the Fog Computing paradigm on MiSCi to deal with that issues, and this paper is focused in show the benefits of such integration. Fig. 3 shows the interactions of the Agents of the Fog Layer with other agents of the SMA.

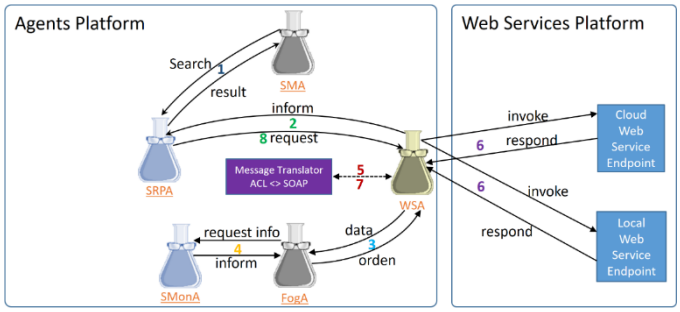

Fig 3. Fog interaction between agents.

Each time any agent of the platform is going to consume a service, it searches that service using the SMA, which responds with the id of the WSA that characterizes that service in the platform. Later, the requesting agent sends the information to the WSA, which contact the FogA, in order to know where will be the service (fog or cloud). The FogA uses information collected by the SMonA, and the Fog Ontology [21], to determine where the service will be executed, and returns the information to the WSA. The WSA translates the message into a SOAP message, and executes the service using the information returned by the FogA. After, the WSA receives the result from the web service, it translates that result to ACL, and sends the result to the requesting agent. In that sense, the FogA helps to deal with problems of quality of services, latency, real time, among others, of the cloud architecture, by combining real time information of the system, with information of the context on the Fog Ontology.

Fundamentally, in [19] we introduces MiSCi as an agent based Middleware for Smart Cities, which includes layers like SML to allow autonomous SOA-MAS integration; OEL to update and adapt the ontologies according to the dynamic of the environment; and CAL to manage the information about the context. Later, in [21] we have presented a modification to MiSCi, in which we have included the Fog Layer. In this paper, we describe the way in which the fog layer works, we present a comparison with other works, as well as some experiments to evaluate the performance of the Fog Layer in MiSCi at the level of the real-time response and latency issues.

IV. Experiments

In this section are presented a series of case studies, to verify if the incorporation of the fog computing paradigm in MiSCi improves the performance and makes it possible to deal with real-time problems, latency, quality of services, among others aspects. In this sense, three general case studies were designed, and other specific that is used to compare our approach with previous works.

A. General Considerations for simulation.

The variables to be taken into account in the simulations are:

- Number of MiSCi agents that offer their functionalities as services.
- Number of web services registered in the UDDI (Number of services usable by agents).
- Maximum number of WSA agents supported by the platform, which depends on the resources of the SMA.
- Average time of generation of new web service requirements by agents (Agent-> WS).
- Average time of generation of new requirements for agent services (WS-> Agent).
- Average response time of the web services. This variable always remains constant since it affects both proposals equally, and it does not influence the comparison of them.
- Standard deviation of the response time of the web services (this variable always remains constant since it does not influence the performance of the proposal).
- Average response time of the agents (this variable always remains constant since it does not influence the performance of the proposal).
- Standard deviation of the response time of the agents (this variable always remains constant since it does not influence the performance of the proposal).
- Average response time of requests for the SMA agent (this variable always remains constant since it does not influence the performance of the proposal).
- Average time for message translation time (SOAP-ACL, ACL-SOAP). This variable always remains constant because it affects both proposals equally, and it does not influence the comparison of them.
- Necessity of processing in the fog (it takes values between 0% and 100%, and it is only valid when the fog component is activated).

Each case study presents a variation of these variables, in order to study specific characteristics of the system. For all scenarios, a simulation time of 1800 seconds and a maximum of WSA agents of 2000 was established. That is, the SMA platform cannot create more than 2000 WSA agents; that means that if there are many concurrent service invocations and it is exceeded the maximum value of WSA supported, then the next requests will be rejected, until the WSA number drops again. In this way, we avoid oversaturating the system resources.

B. Simulation cases and results.

Case 1: In this case, we test the middleware with low (10), middle (200) and high (1000) quantity of services. Additionally, we are going to evaluate the performance of the middleware with fog and without fog. In addition, we set four levels of service requests: low level (10 requests each second), middle level (20 requests each second), high level (50 request each second), very high level (100 requests each second). Moreover, in this case, we are going to assume that we have low requirements of fog computing (need of real time is low, and the Internet latency is very low). In this sense, 20% of all generated requests are going to need real-time processing.

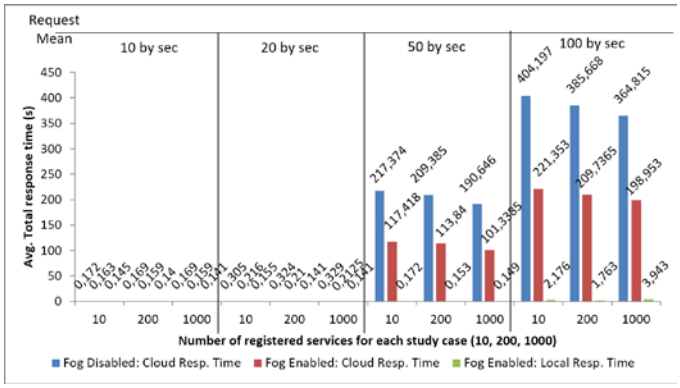

Fig. 4. Average total response time vs number of services.

Fig. 4 shown the average total response time (it includes lookup time of service, invocation of the service, message translation, delivery time for local message, and delivery time for cloud messages) respect to the number of services. Left bar indicates the average total response time for the request from the SMA to services in the cloud (fog is totally disabled, requests are processed in the cloud), the second bar corresponds to the total response time to consume services in the cloud (fog is enabled), and the last bar indicates the average total response time for requests with real time needs (processed in the fog).

From Fig. 4, when the Request Mean is 10 or 20 requests by second, then the average of the total response time is lower than 1 second in all cases; however, the average total response time for those requests that need real time process is always lower. On the other hand, when the request mean is 50 or 100 requests by second, then the avg. total response time for those request processes in the cloud when fog is enabled are always smaller than those request where fog is not enabled. In the same sense, and more important, it is the fact that when fog is enabled, then those requests that requires real-time or low latency have always a total response time very close to 1s. Based on that, we can say that enabling fog computing in the original proposal, give us good results for those requests that need real-time or low-latency, when the needs of real time is low. In this case, only 20% of all the generated requests on the system require to be processed in real-time, and our proposal allows doing that. This is very important for real time applications in the context of smart cities, cyber physical systems, industry 4.0, among others. For example, in a smart city, this allows providing healthcare services like first aids, or vehicular information like traffic and geolocation, to users in real time. In the context of the industry 4.0, this allows accessing production process or process mining as services, among others, in the cloud.

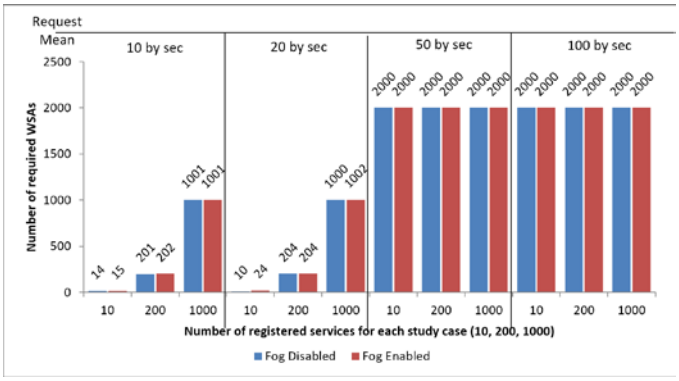

Fig. 5. Number of required WSA vs the number of registered services.

Fig. 5 shows the number of WSA required to attend all the requests of services generated on the system (left bar when fog is not enabled, right bar when fog is enabled). According to Fig. 5, the behavior is similar whether or not fog is enabled. That means that the inclusion of the Fog Layer on the system does not change the deployment of our middleware for the SOA-MAS integration.

Fig.6. Average total response time vs number of registered agents as services.

In Fig. 6 is shown how communication from services in the cloud with the agents' in the platform (web services consuming agents as services) is being affected when fog is enabled or not. We can see that there is not a high difference between both results, which means that the inclusion of the Fog Layer on the system does not affect the communication in our SOA-MAS integration approach.

Case 2: This case is an extension of the case 1, but now, we have high needs of real time because we have to solve a health emergency that have arisen in the system. That means that 70% of services are going to require real time processing.

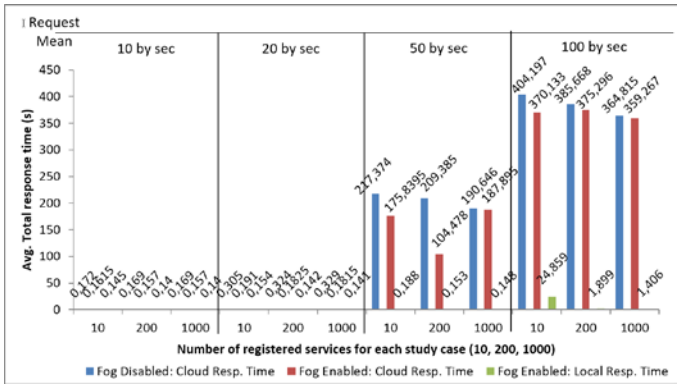

Fig. 7. Average total response time vs number of services.

Fig. 7 shows the same information as Fig. 4. From Fig. 7, we can see that in general, all those requests that require real-time when the fog paradigm is enabled in the systems, was resolved in much lower time, that those which require or not real time, when fog computing is not enabled Also, the difference is very considerable when we have 50 or 100 requests by second. In the same way, we can also notice that when fog is enabled, those requests processed in cloud take less time to be resolved that when the fog is not enabled. That is because the internet is less congested since more requests are being processed in the fog. In general, this figure shows that when we have a good internet speed, and high requests of real time, the system with fog enabled brings better results that when the fog is disable, respect to the total response time in average.

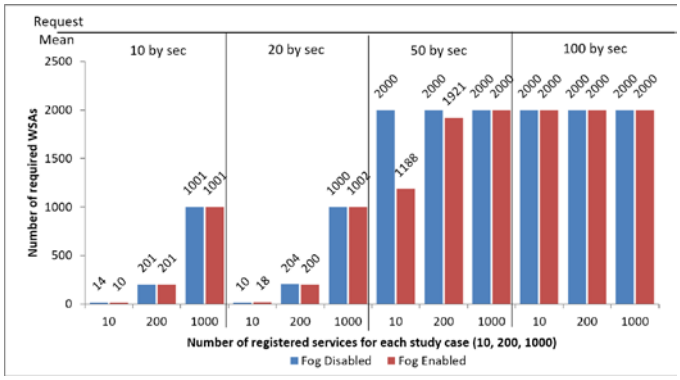

Fig. 8. Number of required WSA vs the number of registered services.

On the other hand, analyzing the requirements of the system respect to the number of WSA needed to process all the requests generated in the system, according to Fig. 8, there is not a high difference when fog computing is enabled in the system, that when fog computing is disabled. Moreover, in Fig. 9, we notice that the total response time in average for communication SOA-MAS (services in cloud consuming agent tasks as services) is lower when fog is enabled in the system for all cases. In that sense, we can say that enabling fog in the system, when internet connection is good and many services need real time, allows reducing the response time in all cases.

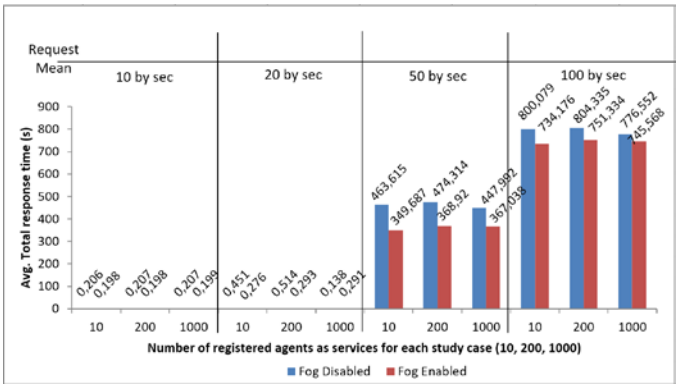

Fig. 9. Average total response time vs number of registered agents as services.

Case 3: This case is also an extension of the case 1, but now, the latency of the internet is high (200ms), and we have real time needs. That means most services (80%) are going to require low latency or real time processing.

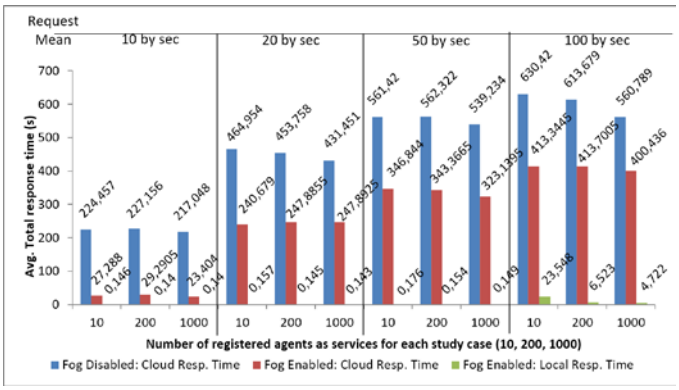

Fig. 10. Average total response time vs number of services.

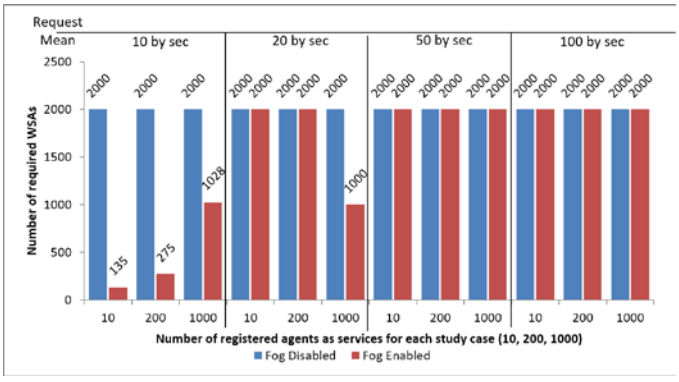

Fig. 11. Number of required WSA vs the number of registered services.

In this third case, we get similar results that in cases 1 and 2, From Fig. 10-12, we can deduce that when fog is enabled in the platform we get best results respect to the total response time in average (bidirectional), and the number of WSA agents required to process all the requests. The fact that more request is processed locally, allows the system to deal with low latency for those requests processed in the cloud, which helps to reduce the total response time when fog is enabled. In changed, when the fog is disable all traffics go thought internet, increasing the latency time, and increasing the total response time in both ways of the communication.

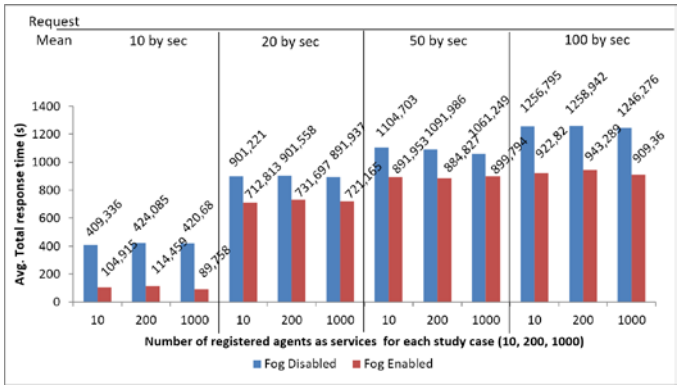

Fig.12. Average total response time vs number of registered agents as services.

Based on the previous results, we can say that adding the fog layer to our proposal for the SOA-MAS integration, brings better results and allows the system to deal properly with low internet latency and real time situations.

Case 4: This case was designed to allow the comparison of our proposal with previous works. Only Mohamed et al. [29] have presented quantitative metrics that can be used for comparison with this work. Same as Mohamed et al. [29], we are going to use 2 services, a very low latency of internet, and 1 request each second. In the same way, we are going to assume that each service is invoked each second, and we are going to simulated by 60 seconds ([29] only carried out 10 invocations to each service). On the other hand, also, we must assume that 50% of the request require real time processing. In this way, we reproduce the case study of Mohamed et al. [29] in MiSCi.

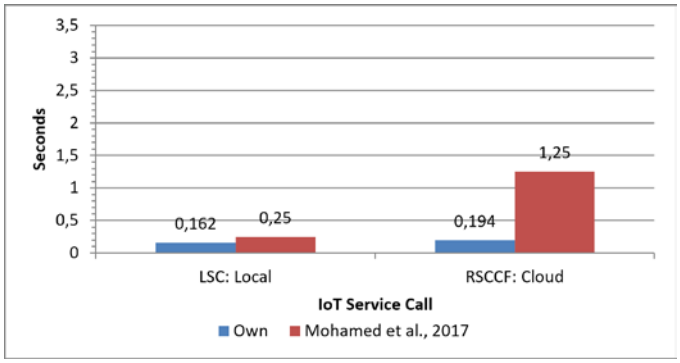

Fig. 13. System (fog-enabled) response times (own: blue, [29]: red)

In Fig. 13, we can see the results. In this figure, LSC refers to services with real time needs (processed locally) and RSCCF refers to services in cloud (don't requires real time), see [29] for more details. In our case, the response time for those services computed in the fog is 162 ms, while that the middleware proposed in [29] produced response times between 250ms and 500ms. On other the hand, services processed in the cloud give us response times of 194 ms, while that in [29] the response time for services in cloud was between 1250ms and 1500ms. Thus, our integration proposal gives better results that the middleware proposed in [29].

The difference between the response time of our middleware and the proposal in [29] is due to that our system combines semantic and context information from the OEL and CAL Layers, in order to identify if the data received by the FogA require real-time processing. MiSCi uses an ontology [21] to verify some properties, such as real-time needs, or to know whether previously the requested service was deployed locally or in the cloud and its performance. That allows, among other things, that FogA decides the correct place of the service (local or cloud) in a lower time.

V. SOA-MAS INTEGRATION IN THE CONTEXT OF THE INDUSTRY 4.0

In the context of Industry 4.0, several authors have proposed the use of MAS [40]–[43] to deal the decision-making challenges, as well as the autonomous coordination, cooperation and collaboration. In that sense, this research can bring huge benefits to the integration process in a production environment with cloud technologies, which at the same time need to solve latency and the real time problems, among others. To illustrate this idea, we will present the following case study.

Suppose a company that has several devices on an assembly line (see Fig. 14), where intelligent products control their own production process. Likewise, there are consumers who place orders for request customized products, and that need to have them at a specific time, for which the company must accept the elaboration of the product prudently. Smart products are the ones that coordinate their own production. The company requires several *integration mechanisms* in the 3C levels (coordination, cooperation and collaboration) [44]:

1. Smart products are the ones that coordinate their own production. It is necessary some appropriate *coordination mechanisms*, directed by the smart product, in such a way that in each phase of the production process the necessary elements are added to the product according to the requirements specific to each product, which may vary from one smart product to another.
2. In the same way, the physical elements (things) of the smart factory must use *cooperative mechanisms*, in order to allow them to carry out the production process in an efficient manner. Each Thing has its own objectives, for example, the objective of the assembly belt is to transport the product from an initial place to a destination one, knowing that there can be multiple origins and destinations. The objective of a robotic arm may be to add a layer to the final product, and so on. In this way, cooperation between all the actors will allow the final products to be created properly.
3. The objective of the whole production process is to produce smart products, in an efficient way, minimizing the production time, costs, as well as the resources or raw materials. This means that the elements of the production process must take into account this common goal, and *collaborate* among them to achieve it, without neglecting their particular objectives, that is, they must deal with multiple objectives.
4. On the other hand, the intelligent factory can *cooperate* with other organizations, in order to make automatic requests of the raw materials, in such a way that the production process is not stopped because of them. Finally,

the smart products can *cooperate* with the shipping organization, so that the products reach the final consumer appropriately and in time.

In this Industry 4.0 scenario, the 3C processes are fundamental to achieve the goals of the production process.

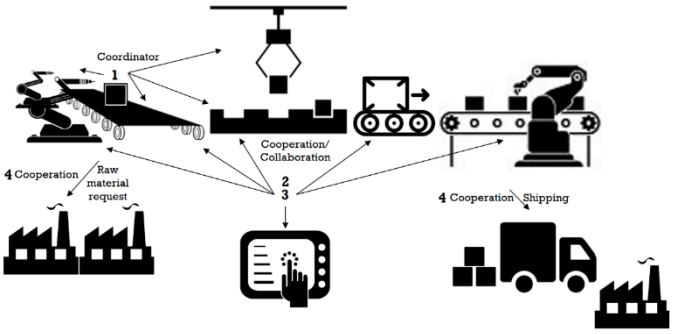

Fig 14. Industry 4.0 scenario with the 3C processes.

In order to carry out the coordination, cooperation or collaboration processes through a MAS, each device (thing) and smart product must be instantiated as agent. Nevertheless, because their capabilities are limited, many of their functionalities will be executed in the cloud, and eventually in the fog, for which it is required that the agents can establish communication with these services. It is here where the SML layer of MiSCi can be configured to help deal with that issue. Fig. 15 shows how would be the process of communication between the elements of the smart factory instantiated as agents, and the cloud or fog services.

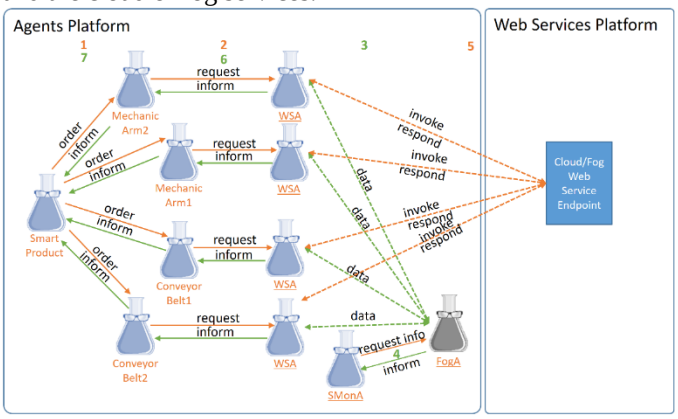

Fig. 15. Instantiation of the Industry 4.0 case study using MiSCi.

Fig. 15 shows how the coordination process led by a smart product would be carried out.

1. In this case, the smart product is the one that gives the orders to the other devices of the production system (arms, conveyor belt, etc.), according to the coordination plan that it handles.
2. These devices must perform certain tasks that allow adding layers to the final product, and for that, they benefit from the paradigm of cloud computing. In order to invoke the corresponding service, the devices must speak with the WSA agent (there is a WSA for each web service registered in the SMA) that characterizes the corresponding service in the system.
3. The WSA will communicate with the agent FogA, in order to know if the required service is located in the cloud or in the fog.
4. Then, the FogA requests the system information to the SMonA, as well as contextual information, using the context web services, which helps to determine what are the needs of real-time processing, low latency, among other things, in order to determine the location of the respective service, and returns the result to WSA.
5. WAS uses this information to adequately invoke the service (in the cloud or in the fog).
6. Once the WSA receives the results of the invoked service, it will return that information to the requesting device agent, who will be able to carry out its task.
7. Finally, the device agent informs about its task to the smart product agent, so that it can adequately adjust the coordination plan, if it is needed.

In the same way, we would proceed with the other cases for cooperate and collaborate. In that sense, this research is useful to introduce the MAS capabilities in the industry 4.0, by integrating them with the cloud computing paradigm, in order to deal with real-time needs, low latency, among other things, that are essential in the context of the Industry 4.0.

Particularly, as we explained before, all the devices on the production line are characterized as agents on the system, and they access services in the cloud or fog, to plan their tasks appropriately. For example, a smart product can access a process model as service, which has the knowledge base and specification on how to build the product, also, a scheduling service can be useful, in order to schedule the tasks of each device. On the other hand, device agents can request services that return the specifications on how to perform a specific task for a specific product (ex. trajectories, specific cuts, and 3d models), etc. In general, all the systems need to access services in the cloud, and it is very important to avoid real-time and low latency issues, in order to allow an appropriate production line, and to prevent to stop the production process.

VI. Comparison with previous works

In this section, we compare our approach with previous works. Table I resumes these differences using the next criteria:

M1. SOA-MAS Enabled. It indicates if the work allows the communication between agents and web services.
M2. Bidirectional SOA-MAS Integration. It indicates whether the agents are able to consume web services, and the web services are able to invoke the agent's tasks.
M3. SOA-MAS Enabled Autonomously. It indicates if the system administrator does not need to create any gateway or communication channel manually to allow SOA-MAS integration, and if agents/web services do not need to make neither message transformation. That means, all the communication is made transparently.
M4. Fog Enabled. It indicates if the work supports the Fog computing paradigm.
M5. Fog Enabled Autonomously. It indicates if the fog layer of the architecture is able to decide whether the data must be

processed autonomously (cloud or fog), according to the current context.

TABLE I
COMPARISON WITH PREVIOUS WORKS

Work	M1	M2	M3	M4	M5
[22]	✓	×	×	✓	✓
[25]	✓	×	×	✓	×
[26]	✓	×	×	✓	×
[29]	✓	×	×	✓	×
[5]	✓	✓	×	×	×
[8]	✓	×	×	×	×
[31]	✓	×	×	×	×
[17]	✓	×	×	×	×
[18]	✓	✓	✓	×	×
MiSCi	✓	✓	✓	✓	✓

As it can be observed from Table I, MiSCi is the only middleware that is able to integrate the SOA-MAS-Fog paradigms in an autonomous way. That means, the agents and web services can communicate in a bidirectional way, in order to solve the situations that arises in the environment, avoiding real time and latency issues by autonomously using the fog computing paradigm. In addition, our SOA-MAS-Fog integration architecture can be used in different AmI contexts, such as Smart Cities, Industry 4.0, among others.

VII. CONCLUSION

In this work, we have shown a component that allows the integration of the MAS, SOA and Fog computing paradigms in intelligent environments, to allow agents and web services to communicate naturally, and in turn, to be able to use the fog computing paradigm automatically. This allows taking advantage of the capabilities of both paradigms (MAS and cloud computing), while at the same time resolving typical problems of cloud computing based systems, like low latency, real time, geo-distribution, among others.

In particular, the integration architecture presented has shown good results in the cases studied, where we focus on solving real-time problems and low latency, demonstrating that this solution provides better results to deal with such problems, when the fog-computing component is enabled. For example, from Fig. 10, we can notice that when the fog layer is enabled, the total response time on average is reduced considerably, having in most cases response times lower to 1 second, and in all cases lower than 1 minute. That result is very desirable in real time systems. In the same way, the average response time for services on the cloud are also lower when the fog layer is enabled, because of having less traffic over internet.

An important remark since the Figures 4, 7 and 10 is that the local response time is always low, and which has not a lineal dependency with respect to the number of requests. That means that MiSCi guarantees low response time when data require real time processing.

As well, with respect to the proposal of Mohamed et al. [29], it has been shown that this proposal provides better results, both when invoking local services and when using services in the cloud. In general, the main difference of this research with respect to the works cited in section II is that our work does not only allow MAS and SOA to communicate in a bidirectional way, but also, in a transparent way. Thus, each agent does not need to worry about SOA specifications, and SOA services do not need to deal with the FIPA details. On the other hand, works on the section II do not deal with the real time and latency issues. They only consider the integration of the MAS and SOA paradigms, but, in most cases in a non-transparent way, and in others only in one direction (from MAS to SOA), which means that SOA services cannot start a conversation with the agents. Our approach autonomously uses the fog computing paradigm according to the context of the information, based on the real time and latency needs of the moment.

On the other hand, the case study presented in the context of Industry 4.0 shows that our integration proposal can easily be adapted to these systems, such is the case of cyber physical systems, Smart factories, among others. Particularly, the result of this research can be useful on the industry 4.0, with the objective of the invocation of everything mining tasks (process mining, services mining, people mining, data mining, etc.) as services [44], in order to provide knowledge that allow performing coordination, cooperation and collaboration processes autonomously.

In this way, a new middleware can be thought, based on the SML and Fog Layers, in order to facilitate the integration with different systems. Another future work is focused on the specific utilization of the Fog and SML Layers in the context of the industry 4.0, in order to allow autonomous coordination processes in a production environment. Finally, another future work is oriented towards the implementation of this component in the MiSCi architecture, which will allow including all the features of the fog computing paradigm, among others.

REFERENCES

[1] M. Sanchez, J. Aguilar, J. Cordero, and P. Valdiviezo, “Basic Features of a Reflective Middleware for Intelligent Learning Environment in the Cloud (IECL),” 2015, pp. 1–6. doi:10.1109/APCASE.2015.8.

[2] J. Aguilar, P. Valdiviezo, J. Cordero, and M. Sánchez, “Conceptual design of a smart classroom based on multiagent systems,” in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, Las Vegas, 2015, pp. 471–477. .

[3] J. Aguilar, M. Mendoça, M. Jerez, and M. Sanchez, “Emergencia ontológica basada en análisis de contexto, como servicio para ambientes inteligentes,” *DYNA*, vol. 84, no. 200, pp. 28–37, Jan. 2017. doi:10.15446/dyna.v84n200.59062.

[4] J. Aguilar, M. Jerez, M. Mendonca, and M. Sánchez, “MiSCi: Autonomic Reflective Middleware for Smart Cities,” in *Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings*, R. Valencia-García, K. Lagos-Ortiz, G. Alcaraz-Mármol, J. del Cioppo, and N. Vera-Lucio, Eds. Cham: Springer International Publishing, 2016, pp. 241–253. doi:10.1007/978-3-319-48024-4_19.

[5] B. Archimède, M. A. Memon, and K. Ishak, “Combining multi-agent model, SOA and ontologies in a distributed and interoperable architecture to manage

multi-site production projects," *Int. J. Comput. Integr. Manuf.*, vol. 30, no. 8, pp. 856–870, Aug. 2017. doi:10.1080/0951192X.2016.1224389.

[6] A. Adaldo, D. Liuzza, D. V. Dimarogonas, and K. H. Johansson, "Coordination of Multi-agent Systems with Intermittent Access to a Cloud Repository," in *Sensing and Control for Autonomous Vehicles*, Springer, Cham, 2017, pp. 453–471. doi:10.1007/978-3-319-55372-6_21.

[7] V. Jain and M. K. Madan, "Multi Agent Driven Data Mining For Knowledge Discovery in Cloud Computing," "*International J. Res. Comput. Appl. Manag.*, vol. 3, no. 1, pp. 111–117, Mar. 2017. .

[8] M. Al-Ayyoub, Y. Jararweh, M. Daraghmeh, and Q. Althebyan, "Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure," *Clust. Comput.*, vol. 18, no. 2, pp. 919–932, Jun. 2015. doi:10.1007/s10586-015-0449-5.

[9] D. Greenwood, P. Buhler, and A. Reitbauer, "Web service discovery and composition using the web service integration gateway," in *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service.*, 2005, pp. 789–790. .

[10] A. Sánchez, G. Villarrubia, C. Zato, S. Rodríguez, and P. Chamoso, "A Gateway Protocol Based on FIPA-ACL for the New Agent Platform PANGEA," in *Trends in Practical Applications of Agents and Multiagent Systems*, Springer, Cham, 2013, pp. 41–51. doi:10.1007/978-3-319-00563-8_6.

[11] C. A. Marín *et al.*, "A Conceptual Architecture Based on Intelligent Services for Manufacturing Support Systems," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK, 2013, pp. 4749–4754. doi:10.1109/SMC.2013.808.

[12] FIPA, "FIPA ACL Message Structure Specification," *Foundation for Intelligent Physical Agents*. [Online]. Available: http://www.fipa.org/specs/fipa00061/index.html. [Accessed: 20-Dec-2017].

[13] S. S. Singapogu, K. Gupton, and U. Schade, "The Role of Ontology in C2SIM," in *21st International Command and Control Research and Technology Symposium*, London, UK, 2016. .

[14] R. Geetha and K. L. Shunmuganathan, "Intelligent query processing from biotechnological database using co-operating agents based on FIPA standards and hadoop, in a secure cloud environment," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2017, pp. 1–4. doi:10.1109/ICACCS.2017.8014599.

[15] Oracle, "Simple Object Access Protocol Overview," *Oracle9i Application Server Oracle9iAS SOAP Developer's Guide*. [Online]. Available: https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm. [Accessed: 20-Dec-2017].

[16] W3C, "Extensible Markup Language (XML)." [Online]. Available: https://www.w3.org/XML/. [Accessed: 20-Dec-2017].

[17] M. Fuksa, "Methods and Tools for Intelligent ESB," Master, Czech Technical University in Prague, Prague, 2014.

[18] A. Pinto Pereira, "Towards robustness and self-organization of ESB-based solutions using service life-cycle management," Master Thesis, Polytechnic Institute of Bragança, Bragança, 2014.

[19] M. Sanchez and J. Aguilar, "Integrating SOA and MAS in Intelligent Environments," *DYNA*, p. Reviewing. .

[20] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, Helsinki, Finland, 2012, pp. 13–16. .

[21] J. Aguilar, M. B. Sanchez, M. Jerez, and M. Mendonca, "An Extension of the MiSCi Middleware for Smart Cities Based on Fog Computing," *J. Inf. Technol. Res. JITR*, vol. 10, no. 4, pp. 23–41, 2017. doi:10.4018/JITR.2017100102.

[22] M. Amadeo, A. Molinaro, S. Y. Paratore, A. Altomare, A. Giordano, and C. Mastroianni, "A Cloud of Things framework for smart home services based on Information Centric Networking," 2017, pp. 245–250. doi:10.1109/ICNSC.2017.8000099.

[23] F. Riggins and T. Keskin, "Introduction to Internet of Things: Providing Services Using Smart Devices, Wearables, and Quantified Self Minitrack," in *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017. doi:10.24251/HICSS.2017.166.

[24] S. Sengupta, N. Gupta, and V. (Advisor) Naik, "Firewall for internet of things," Indraprastha Institute of Information Technology, New Dehli, 2017.

[25] Z. Peng, B. Xu, A. M. Gates, D. Cui, and W. Lin, "A Study of a Multi-Agent Organizational Framework with Virtual Machine Clusters as the Unit of Granularity in Cloud Computing," *Comput. J.*, vol. 60, no. 7, pp. 1032–1043, Jul. 2017. doi:10.1093/comjnl/bxw042.

[26] A. Giordano, G. Spezzano, and A. Vinci, "Smart Agents and Fog Computing for Smart City Applications," in *Smart Cities*, 2016, pp. 137–146. doi:10.1007/978-3-319-39595-1_14.

[27] S. Zanero, "Cyber-Physical Systems," *Computer*, vol. 50, no. 4, pp. 14–16, Apr. 2017. doi:10.1109/MC.2017.105.

[28] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, 2014, pp. 1–4. doi:10.1109/AQTR.2014.6857843.

[29] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, and S. Mahmoud, "SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services," *IEEE Access*, vol. 5, pp. 17576–17588, 2017. doi:10.1109/ACCESS.2017.2731382.

[30] M. Faheem and V. C. Gungor, "Energy efficient and QoS-aware routing protocol for wireless sensor network-based smart grid applications in the context of industry 4.0," *Appl. Soft Comput.*, Jul. 2017. doi:10.1016/j.asoc.2017.07.045.

[31] J. A. García Coria, J. A. Castellanos-Garzón, and J. M. Corchado, “Intelligent business processes composition based on multi-agent systems,” *Expert Syst. Appl.*, vol. 41, no. 4, Part 1, pp. 1189–1205, Mar. 2014. doi:10.1016/j.eswa.2013.08.003.

[32] X. Fu, T. Bultan, and J. Su, “Analysis of Interacting BPEL Web Services,” in *Proceedings of the 13th International Conference on World Wide Web*, New York, NY, USA, 2004, pp. 621–630. doi:10.1145/988672.988756.

[33] G. Decker, O. Kopp, F. Leymann, and M. Weske, “BPEL4Chor: Extending BPEL for Modeling Choreographies,” in *IEEE International Conference on Web Services (ICWS 2007)*, 2007, pp. 296–303. doi:10.1109/ICWS.2007.59.

[34] J. Aguilar, A. Ríos, F. Hidrobo, and M. Cerrada, *Sistemas Multiagentes y sus aplicaciones en Automatización Industrial*, 2nd ed. Mérida: Talleres Gráficos, Universidad de Los Andes, 2013.

[35] M. Sánchez, J. Aguilar, J. Cordero, and P. Vadiviezo, “A Smart Learning Environment based on Cloud Learning,” *Int. J. Adv. Inf. Sci. Technol.*, vol. 4, no. 7, pp. 36–49, Jul. 2015. doi:10.15693/ijaist/2015.v4i7.36-49.

[36] J. Aguilar, M. Jérez, E. Exposito, and T. Villemur, “CARMiCLOC: Context Awareness Middleware in Cloud Computing,” 2015, pp. 1–10. doi:10.1109/CLEI.2015.7360013.

[37] M. Mendonça, J. Aguilar, and N. Perozo, “MiR-EO: Middleware Reflexivo para la Emergencia Ontológica en Ambientes Inteligentes,” *Lat. Am. J. Comput. Fac. Syst. Eng. Natl. Polytech. Sch.*, vol. 3, no. 2, p. 16, 2016. .

[38] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, Helsinki, Finland, 2012, pp. 13–16. .

[39] R. Mahmud, R. Kotagiri, and R. Buyya, *Fog computing: A taxonomy, survey and future directions*. Springer, 2018.

[40] D. Li, H. Tang, S. Wang, and C. Liu, “A big data enabled load-balancing control for smart manufacturing of Industry 4.0,” *Clust. Comput.*, vol. 20, no. 2, pp. 1855–1864, Jun. 2017. doi:10.1007/s10586-017-0852-1.

[41] Z. Huang, H. Yu, Z. Peng, and Y. Feng, “Planning community energy system in the industry 4.0 era: Achievements, challenges and a potential solution,” *Renew. Sustain. Energy Rev.*, vol. 78, no. Supplement C, pp. 710–721, Oct. 2017. doi:10.1016/j.rser.2017.04.004.

[42] D. Romero, T. Wuest, J. Stahre, and D. Gorecky, “Social Factory Architecture: Social Networking Services and Production Scenarios Through the Social Internet of Things, Services and People for the Social Operator 4.0,” in *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing*, 2017, pp. 265–273. doi:10.1007/978-3-319-66923-6_31.

[43] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, “Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination,” *Comput. Netw.*, vol. 101, no. Supplement C, pp. 158–168, Jun. 2016. doi:10.1016/j.comnet.2015.12.017.

[44] M. Sanchez, J. Aguilar, and E. Exposito, “Industry 4.0 Survey and Challenges since an Integration Perspective,” Univeristé de Pau et des Pays de l’Adour, Anglet, France, Technical Report 186, Nov. 2017.