Agent-Oriented Cooperative Smart Objects: from IoT System Design to Implementation

Giancarlo Fortino, *Senior Member, IEEE*, Wilma Russo, Claudio Savaglio, *Student Member, IEEE*, Weiming Shen, *Fellow, IEEE*, Mengchu Zhou, *Fellow, IEEE*

***Abstract*—The future Internet of Things (IoT) is expected to enable a new and wide range of decentralized systems (from small-scale smart homes to large-scale smart cities) in which "things" are able to sense/actuate, compute and communicate, and thus play a central and crucial role. The growing importance of such novel networked cyber-physical context demands suitable and effective computing paradigms to fulfill the various requirements of IoT systems engineering. In this paper, we propose to explore an agent-based computing paradigm to support IoT systems analysis, design, and implementation. The synergic meeting of agents with IoT makes it possible to develop smart and dynamic IoT systems of diverse scales. Our agent-oriented approach is specifically based on the ACOSO (Agent-based COoperating Smart Objects) methodology and on the related ACOSO middleware: they provide effective agent design and programming models along with efficient tools for the actual construction of an IoT system in terms of a multi-agent system. A case study concerning the development of a complex IoT system, namely a *Smart University Campus*, is described to show the effectiveness and efficiency of the proposed approach.**

***Index Terms*— Internet of Things, Cooperative Smart Objects, Multi-Agent Systems, Agent-Oriented Software Engineering**

I. INTRODUCTION

THE Internet of Things (IoT) term refers to a loosely coupled, decentralized and dynamic system in which billions (even trillions) of everyday objects are globally interconnected and endowed with smartness, becoming active participants in business, logistics, information and social processes [1]. Such "things" can be commonly defined as smart objects (SOs) and, if supported by an "anywhere, anytime and anything connection" [2], they represent the fundamental building blocks for the IoT [3]. In fact, SOs are able to provide highly pervasive cyber-physical services to both humans and machines thanks to their communication, sensing, actuation, embedded processing, and even reasoning abilities.

Giancarlo Fortino, Wilma Russo and Claudio Savaglio are with the Department of Informatics, Modelling, Electronics and Systems Engineering (DIMES), University of Calabria, Via P. Bucci, cubo 41C, 87036 Rende (CS), Italy (g.fortino@unical.it, w.russo@unical.it, csavaglio@dimes.unical.it).

Weiming Shen is with the National Research Council Canada, Ottawa, Ontario, ON K1A 0R6, Canada (wshen@ieee.org).

Mengchu Zhou is with Department of Electrical and Computer Engineering, New Jersey Institute of Technology Newark, NJ 07102, USA (zhou@njit.edu).

The development of IoT systems, e.g., smart home [4], smart car, smart factory [5, 6], and smart city, their management as well as their integration in real applications, are complex and challenging, thereby requiring suitable models, methods/techniques and technologies. In this direction, several middleware solutions, tools and methodologies have been developed, facing notable challenges such as physical device virtualization [7], decentralized entity management [8], and guideline identification [9]. However, these solutions tend to tackle different specific issues, typically one at a time, without providing a full-fledged methodology to support the entire IoT system development process, from analysis to implementation. However, such partial approach results in poorly interoperable, poorly scalable or application-driven "Intranet of Things" systems, thus leading away from the original inclusive IoT vision. Therefore, by providing a full-fledged and application-neutral methodological approach for IoT systems development, we aim at concretely developing the IoT concept of a horizontal landscape of interoperable cyber, physical, and cyber-physical systems.

To deal with such challenges, this work proposes the exploitation of the *Agent-based Computing* (ABC) paradigm [10], which is centered on the concept of *Agent* as a well-defined software engineering and distributed computing abstraction, for designing, programming, deploying and managing IoT systems. The ABC paradigm allows modelling distributed software systems in terms of multi-agent systems (MASs), where agents are networked software entities that can autonomously perform specific tasks on behalf of a user by properly interacting with other agents and with their environment. Due to such reasons, agents have been effectively used in many application domains to develop robust and dynamic distributed systems/applications [11-13]. However, few research efforts are currently focused on defining methodologies and middleware to develop agent-oriented IoT systems. In our view, the main agent features (autonomy, social ability, responsiveness, proactiveness, and mobility) perfectly fit the generic and specific requirements of IoT systems [14, 15].

This work first elicits and discusses main IoT system development requirements at both system and things levels (namely, requirements related respectively to the whole system or to its individual components), then proposes a full-fledged approach to IoT system development based on our Agent-oriented COoperating Smart Objects Methodology,

called ACOSO-Meth for short, and its related middleware. ACOSO-Meth supports the SO development phases of analysis, design and implementation by means of metamodels featured by different levels of abstraction. Each metamodel is derived from the previous phase metamodel to allow a seamless transition from analysis to implementation phases as to enable an easy-to-do translation from analysis system models to implementation system models. The design and implementation phases are currently based on the ACOSO [17] middleware that provides an effective agent model and a JADE-based platform to program both basic SOs and more complex IoT systems. Finally, the application of ACOSO-Meth to the development of a complex IoT system, including small-, medium-, and large-scale SOs, highlights the effectiveness as well as the efficiency of the proposed approach.

This work contributes to the state-of-the-art in IoT system engineering with the following three main contributions:

- A comparison framework comprising IoT fundamental development requirements, raised from a thorough state-of-the-art analysis, has been designed. It inspired the ACOSO-Meth development but it can be reused to compare future work in the field.
- ACOSO-Meth is the first application domain-neutral, full-fledged agent-based approach able to support the main engineering phases of IoT systems and applications, thus fulfilling the fundamental system-level and things-level requirements (by referring to our previous work, [16] presented preliminary and less detailed models, not structured according to the SO main features, while [68] presented functional and data models specifically conceived for smart environments and designed for natively supported edge computing).
- ACOSO-Meth is applied to develop a complex *Smart University campus*, that have been tested according to our performance evaluation approach for IoT systems, specifically involving the concept of scale (small, medium, and large), the number of IoT devices and communication sub-networks (by referring to our previous work, [16] presented a case study but no running example, while [68] considered a single use case, with few devices and a single operation modality without any performance evaluation).

As highlighted also by the *Smart University Campus* case study, IoT system outcomes are strongly influenced by many cyber-physical factors such that any performance analysis or variable optimization could be narrowed only to the evaluated application and its specific configuration (in every real scenario, device deployments, adopted protocols, and infrastructure design are notably constrained to a physical environment, different functional and non-functional requirements, resource availability, etc.). Due to such reasons, it is out of the scope of this work to select specific requirements or optimize variables purposely defined for a specific application. The rest of this paper is organized as follows. Section II discusses generic and specific requirements that IoT system development poses. How such requirements have been so far tackled by means of different approaches is surveyed in Section III. Section IV describes the proposed ACOSO-Meth and related middleware. Section V presents a *Smart University Campus* case study to exemplify the ACOSO-based approach and the related performance evaluation to show its effectiveness. Finally, Section VI summarizes this work's research contributions, lessons learned and future work.

II. IoT System Development Requirements

IoT systems are composed of many distributed and interacting components that are usually heterogeneous in terms of hardware devices, communication protocols, software interfaces, data, and semantics. To effectively support their development, general and specific requirements need to be defined [18]. While the general requirements allow effective and flexible middleware for facilitating IoT system programming, the specific requirements are purposely defined for a target IoT system by considering its specific application domain. In the following, we focus on the former that are common to all IoT systems. In particular, we group such requirements in two categories: *System-level* (Table I), which includes requirements related to the whole distributed system and its development, and *Things-level* (Table II), which encompasses requirements particularly referring to the "things" such as Radio Frequency Identification (RFID) items, smart objects, mobile devices, and robots, in an IoT system.

Requirements listed in Tables I and II have been outlined after thoroughly analyzing the state-of-the-art of IoT middleware, architectures and platforms, focusing on their main features and extracting common keywords. Such requirements are not totally new, since they have been already studied in several fields of computer science and engineering. However, at both levels, they recur at the same time and with a substantial prominence within the IoT context and they allow accommodating all the most important features of IoT systems. Indeed, conventional computing devices and everyday things tend to converge in the IoT [1], requiring virtual networked alias (SLR_1), software interfaces (SLR_3) and communication/data abstractions (SLR_2-SLR_5) to synergistically cooperate, despite their heterogeneities (TLR_1). To cope with such cyber-physical (SLR4) and dynamic scenario rich in continuously evolving (TLR4) and augmented (TLR2) things, proper methodologies are needed (SLR6) to fully support the IoT system development. Furthermore, decentralized management (TLR3) mechanisms are essential for making things autonomous and effectively integrated in their application contexts. Finally, at both system- and things-levels, the cyber-physical nature of SOs introduces important novel elements in the characterization of SO-based systems, particularly with regard to the concept of "scale" (SLR_7 and TLR_5). In traditional distributed systems, the concept of scale is closely related to the number of involved computing nodes,

TABLE I
SYSTEM-LEVEL REQUIREMENTS (SLRS)

Requirement	Description
SLR_1: Hardware Devices (Virtualization)	IoT systems typically comprise heterogeneous devices; in order to facilitate their use, abstractions are needed to *virtualize* and let them be used, as they are homogeneous by following a kind of a "plug&play" paradigm [8].
SLR_2: Communication (Abstractions)	Software components and devices need to communicate with each other. Communication abstractions are needed to make them interact and cooperate, independently from the available low-level network protocols [20].
SLR_3: Software Interfaces	As software interfaces are usually heterogeneous, they need to be generic and standardized through higher level mechanisms such that their use is straightforward. Thus, software components based on such high-level interfaces can be seamlessly accessed [21].
SLR_4: Physicality (Self and Context Awareness)	Hardware and software components in IoT systems and entire IoT systems themselves are intrinsically situated. This implies that they have static or dynamic locations and refer to one or multiple contexts during their lifecycle. Abstractions are therefore needed to capture the concepts of location and context, as they are useful in the design and implementation of IoT systems [22].
SLR_5: Data (Abstraction)	Different hardware and software components, e.g., sensors, machines, smart objects, and mobile apps, usually produce data according to different modalities, formats and types. Thus, abstractions are needed to formalize data streams generated by such components. Continuous data streams, discrete data and sporadic events should be defined under a common framework. Moreover, the representation of data types needs to be standardized as it would allow interoperability in data exchange among heterogeneous components [7].
SLR_6: Development Process (Methodology)	To analyze, design and implement IoT systems, suitable software engineering methods and tools need to be defined. They should be able to effectively model IoT systems by using high-level modelling abstractions and fully support their design, implementation, deployment and management [23].
SLR_7: System's Scale Characterization	IoT systems can notably differ in terms of geographical extensions, network infrastructures and number of involved IoT devices. Hence, it is useful to define some criteria to facilitate the unambiguous characterization of their scale and possibly enable their comparison.

their geographical distribution and logical organization among different administrative domains. Such domains usually have different configurations, policies and privileges, thus emphasizing the need of interoperability and coordination mechanisms [19]. In traditional agent-based systems, the scale concept usually overlaps with agent population [24] and agents distribution among host devices, regardless of their actual geographic location. Note that, one of the peculiarities of agents is their mobility. Finally, in Wireless Sensor Networks (WSNs) the concept of scale refers both to the number of involved devices and to their spatial collocation, as the radio communications are strongly susceptible to interferences and mutual collisions [25, 26]. It is just in the WSN context, indeed, that the concept of density, intended as the number of sensors per unit area, appears [25]. In conclusion, depending on the application contexts, the "scale" term is differently defined as well as its characterizations (large, medium, and small scale) can notably vary (a large scale WSNs very likely will differ from a large scale computational grid in terms of geographical extension,

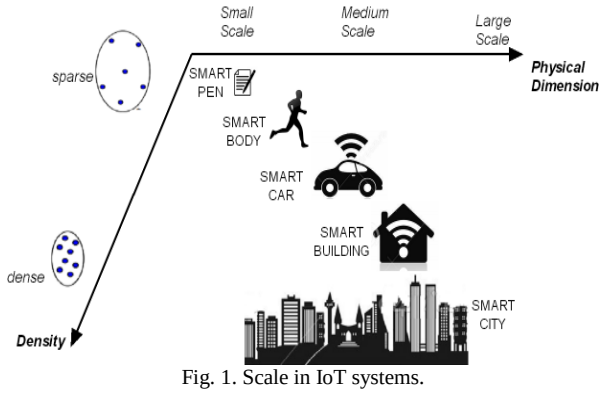


Fig. 1. Scale in IoT systems.

population and density). Within the SO-based IoT context, therefore:

i. It is handy to refer to well established concepts of "small-medium-large scale" taken from traditional distributed systems, as long as such definitions are not exclusively attributable to geographical factors. Moreover it is convenient to take into account the network infrastructure, in particular the number of subnets involved, in order to better evaluate system performance; and

ii. Since SOs are highly pervasive and mostly based on wireless interconnections, the density issue pointed out for WSNs strongly recurs. Although not only simple sensors but even other kinds of functionally heterogeneous devices are involved within SO-based IoT systems, the density remains a useful metric to characterize scenarios when the number of SOs changes.

On the basis of such considerations, and specifically for an unambiguous characterization of the case studies of Section V, hereinafter we classify IoT systems and SOs in small-medium-large scale on the basis of their physical dimension and density, as shown in Fig. 1. Similar criteria for scenario characterization are defined in [27, 28].

III. BACKGROUND

A. ABC paradigm

The ABC paradigm is centered around the concept of Agent, a sophisticated software abstraction that allows instilling smartness and autonomy within a single entity and consequently developing decentralized, cooperating and heterogeneous societies in terms of multi-agent systems (MASs) [31]. Indeed, this paradigm represents both a suitable design metaphor and an effective programming paradigm, providing specific methods, techniques and tools for effectively conceptualizing and developing dynamic, robust and distributed systems (TLR_2, TLR_4) within diverse cyber, physical, and cyber-physical application domains [10, 30]. It intrinsically contemplates high-level concepts (e.g., models, metaphors, and abstractions, SLR_1), concrete mechanisms (e.g., shared interfaces, protocols, and ontology, SLR_2, SLR_3, and SLR_5), and specific guidelines (SLR_6) to support, at both things and system levels, interoperability among agents, other computing systems, the surrounding environment and its resources (TLR_1, SLR_1, and SLR_4).

TABLE II
THINGS-LEVEL REQUIREMENTS (TLRS)

Requirement	Description
TLR_1: Heterogeneity and Interoperability	Applications that use "things" should be programmed independently from vendors-specific "things". For instance, if an application is based on a "smart chair", it should be able to use smart chairs built by different vendors. Moreover, applications should be able to exploit "things" to be built in the future. This implies to adopt a standardized approach or, if not applicable (standardization is a very long process), to exploit software layering-based dynamic adaptation techniques between application and the "things" levels [2].
TLR_2: Augmentation Variation	"Things" usually provide a set of devices and services that can vary in quantity and types both among different "things" and among similar "things". In particular, different "things" can provide same services whereas two similar "things" can provide different services. Thus, "things" cannot be crisply classified only by their type and may expose non-standard interfaces. Augmentation variation of "things" is an important requirement as it defines how "things" can modify their augmentation by providing diversified services that can change during their lifecycle. This implies to design not only methods to dynamically add/modify/remove "things" services and devices but also how they are actually furnished [29].
TLR_3: Decentralized Management	An effective management of "things" is crucial in IoT applications where tons of distributed "things" could potentially interact with each other and/or be used to fulfill a final goal. Applications and "things" should be therefore able to dynamically adapt as "things" could continuously change for different purposes (augmentation variation, mobility, failures, etc.). Thus, the matching among "things" services and application requirements should be often done at run-time. Discovery services are therefore strategic in such a dynamic context to find and retrieve "things" according to their static and dynamic properties [20].
TLR_4: Dynamic Evolution	Applications and "things" should be simply and rapidly prototyped and upgraded through proper programming abstractions. The evolution can be driven by programming, learning, or both. In particular, evolution by learning is usually based on smart self-evolving components (application-level components and smart "things") able to self-drive their evolution on the basis of some learning models [3].
TLR_5: Thing's Scale Characterization	"Things" can notably differ in terms of physical dimensions and number of aggregated devices. Hence, it is useful to define some criteria to facilitate the unambiguous characterization of their scale and possibly enable their comparison.

The fundamental characteristics of agents include: (i) *Autonomy*: agents should be able to perform the majority of their problem-solving tasks without the direct intervention of humans, and they should have a certain degree of control over their own actions and their own internal state (TLR_3); (ii) *Social ability*: agents should be able to interact, when they deem appropriate, with other software agents and/or humans in order to complete their own tasks (TLR_1); (iii) *Responsiveness*: agents should perceive the environment in which they are situated (SLR_4), a physical world, an agent container, and Internet, and respond in a timely fashion to changes that may occur (TLR_4); (iv) *Proactiveness*: agents should not simply act in response to their environment, but they should be able to exhibit opportunistic, goal-directed behavior and take the appropriate initiative; (v) *Mobility*: in order to fulfill distributed tasks, agents should be able to logically migrate from one machine to another (mobile software agents) and/or physically move in a targeted environment, like robots and drones. These features allow agents to be exploited in several crucial roles with various degrees of smartness [31]:

- *Decisional assignments*: Agents are provided with different degrees of intelligence that permit them to autonomously reach their goals and to model their plans on the basis of their sensed contexts (TLR_3, SLR_4). So, an agent paradigm is often exploited as an enabling cognitive technology that allows choosing the best algorithm to apply, the most proper coordination model in the context of collaborative tasks, the most effective realization of user characterization mechanisms and pattern recognition [32].
- *Operational assignments*: Agents are able to autonomously perform their tasks and, at the same time, to interact with their environment in an active manner (TLR_4). Among several operational tasks, in the IoT context, agents are often deputed to service discovery (a crucial task due to a large number and kinds of available services and providers, TLR_2), intra-extra system communication (acting as a middleware layer by exploiting common languages and interfaces, SLR_2, SLR_5), and interaction with the real world, e.g. through cyber-physical SOs [33].
- *Control assignments*: Agents typically monitor the fulfillment of predefined goals and the respect of some thresholds, thus determining the safety and trustworthiness of system/components. In order to realize these tasks, agents optimize the parameters of given algorithms or inspect other entities' behaviors. Particularly, considering that autonomous systems (whose spread is daily increasing in the IoT context, e.g. autonomous cars) contemplate several distinct decision makers, checking in real-time such agents' choices through external monitoring agents is obviously appreciated. Moreover, different from humans, agents react much more quickly and are suitable for a plethora of time-critical applications [65]. Finally, with regard to security concerns, agents are often used in order to increase safety, define trustworthiness metrics, and ensure information and resource flow in dynamic and malicious IoT environments [34].

Exploiting the aforementioned capabilities and features, the ABC paradigm is able to fulfill the system-level (SLR) and things-level (TLR) requirements presented in Section II: therefore, several agent-based middleware and architectures [7], [8], [35]-[37], [66] were presented in the past years. Beside such contributions, a few IoT methodologies [10], [24], [38]-[40], far from the ABC paradigm, have been proposed. Table III shows at which development phase the related work to be introduced next and ACOSO-Meth are placed, and how (totally, partially or not at all) they support the SLRs and TLRs in Tables I and II.

TABLE III
COMPARISON OF RELATED WORK (Y = totally supported, P = partially supported, Blank = not supported)

		Development phase (Analysis, Design, Implementation)			System-Level Requirements							Things-Level Requirements				
		A	**D**	**I**	**SLR1**	**SLR2**	**SLR3**	**SLR4**	**SLR5**	**SLR6**	**SLR7**	**TLR1**	**TLR2**	**TLR3**	**TLR4**	**TLR5**
Agent-oriented Middleware and Architectures	[7]	P	Y		Y	Y			Y	P		Y		Y	P	
	[8]	P	Y		Y	Y	Y	P		P		Y	Y	P		
	[35]		Y	P	Y	Y			Y	P		Y			P	
	[36]		Y	P	Y	Y		Y		P		Y		Y		
	[37]	P	Y		Y	Y	Y			P		Y	P			
	[66]		Y	Y	Y	Y		P		P		Y		P	P	
IoT Methodologies	[9]	Y	P		Y	Y			Y	P		Y	Y		Y	
	[23]	P	Y		Y		Y	Y	Y	P		Y	P	P		
	[38]	P		P			Y		Y	P		Y	Y			
	[39]	P		P			Y		Y	P		Y	Y			
	[40]	Y	P		Y	Y			Y	P		Y	Y		Y	
ACOSO-Meth		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	

In particular, it should be noted that ACOSO-Meth is the only contribution that, fully exploiting the ABC potential, fulfils all the listed requirements at the same time to our best knowledge.

B. Agent-oriented Middleware and Architectures

Agent technologies provide useful software abstractions, independent of a specific implementation (TLR_1), which encourage the development of:

- *Agent-based middleware*, in order to speed up system development and prototyping, as well as management and evolution (TLR_4);
- *Agent-based architectures*, for exploiting the ABC twofold roles: a technology integrator (agents interoperate at high-level, hiding the underlying implementation details, protocols, etc.) and modelling paradigm (SLR_6).

In [7], [8], [35]-[37], [66] IoT entities, even deeply heterogeneous with each other, are virtualized and homogenized by an "agent" definition (SLR_1). Each system component, e.g., everyday object, sensor, and robot, is represented by an agent: it communicates with other agents through well-known specifications (SLR_2), and it may be enhanced with machine learning techniques, pattern recognition mechanisms and semantic technologies. The latter are used both for descriptive specifications (service and resource descriptions) and for prescriptive specifications (component behavioral control and coordination) [35]: in such a way, an agent is able to reason over the data, overcoming the heterogeneities of standards and data formats, which typically represent significant obstacles for system interoperability (SLR_5). Such sort of "intelligence" makes the agent aware of its current status, abilities, goals, and environment. The capability of dynamically elaborating both explicit and implicit contextual information (SLR_4) coming from sensors and actuators makes an agent indispensable for the provision of context-aware dynamic services [8]. Moreover, in order to minimize the human intervention and realize a decentralized management (see TLR_3), an agent is often provided with autonomic capabilities [69] that allow it to self-protect, self-heal, self-configure, self-govern and self-optimize. In [7], an entire toolkit is conceived around the notion of an autonomic agent aiming to provide adaptive, composite and situated intensive services. In [36], agents are able to self-compose, promoting the integration of different applications with the dynamic re-use of system resources: such issue is obviously relevant for the IoT scenario, in which there may exist many resource-constrained components. SOs and related services composition can be done by agents even through high-level software interfaces (SLR_3), like RESTful web service API [37], thus avoiding the need for a specific agent-based middleware solution. In [66], every agent is provided with a generic plug-in for self-configuring its internal communication mechanisms and exploiting different communication protocols, according to the context and other agent technologies. To summarize, the briefly presented agent-oriented middleware and architectures fulfill (often just one at a time) all the SLRs of Table I but they only partially satisfy SLR_6, TLR_2 and TLR_4. In fact, they exploit the agent-based modelling but lack a comprehensive development methodology.

C. IoT Engineering Methodologies

Despite a variety of research efforts that tackle different specific issues within an IoT system development process, a full-fledged IoT methodology is missing. There are many studies which, instead of providing a proper methodology, collect domain-specific best practices, guidelines, checklists and templates. For example, Slama et al. [38] and Collins [39] build up a repository of technology-dependent solutions coming from the experience in the industrial/business world and specifically directed to the IoT makers and enterprises. In fact, they propose reference architectures and guidelines to make specific-purpose devices interoperable (TLR_1) through abstract data models (SLR_5) and high-level software interfaces (SLR_3). Differently, some researchers present general-purpose approaches. IoT-A [9] is a systematic collection of architectures, reference models, common definitions and guidelines that can be used to derive a concrete IoT

architecture. By means of different views, perspectives and metamodels, IoT-A aims to offer a unified approach to the development of IoT systems, in order to promote cross-domain interaction (SLR_2), to support interoperability (TLR_1) and to reduce fragmentation within an IoT context. Most of the indications provided by IoT-A have inspired the AIOTI (Alliance for the Internet of Things) [40] reference models, specifically the domain model. The latter describes IoT entities and their relationships, by eliciting all the TLRs in the SO analysis phase. Zambonelli [23] proposes a software engineering methodology centered on the main general-purpose concepts related to the analysis, design and implementation phases of IoT systems and applications. Such concepts are used to identify the key software engineering abstractions (SLR_1, and SLR_3-SLR_5) as well as a set of guidelines and activities that may drive the IoT systems development. The envisioned methodology, however, lacks the definition of models and tools to represent different conceptual and software artifacts. In brief, the existing methodologies neither completely support the TLRs and SLRs, nor cover the entire development process.

IV. ACOSO METHODOLOGY

As so far argued, SO development is a very complex and articulated process: in order to support the SO analysis, design and implementation phases, we present the ACOSO-Meth (Agent-based COoperating Smart Objects Methodology). It intends to integrate within a comprehensive methodology (SLR_6) the ABC paradigm (whose features and related benefits satisfy the SLRs as argued in Section III-A) and the agent-oriented modelling and programming techniques provided by the ACOSO middleware [17], with a special attention to the TLRs of IoT device, e.g., TLR_2 and TLR_4. Doing so, all the IoT development requirements presented in Section II, at both system- and things-levels, are fulfilled, as shown in Table III and better elicited in Section IV.D. In particular, ACOSO-Meth aims to systematically support an SO development process by means of metamodels placed at different abstraction levels and completely decoupled from any specific application context. Such choice provides generality to the presented methodology and, considering the plethora of ever-changing IoT scenarios, it is a remarkable merit. As matter of facts, a less detailed version of the metamodels presented in the following, and specifically exploited in Section V to model a SmartBridge providing structural health monitoring services, has been straightforwardly used in [70] to model a SmartOffice (aggregating in its turn a SmartDesk, a SmartProjector and a SmartWhiteboard) supporting an officer during a user's daily working activity. This demonstrates that the proposed metamodels are domain-neutral and suitable to be effectively exploited regardless the SO specific scale, purpose or application context.

As showed in Fig. 2 (diagrams are compliant with Object Management Group (OMG) Software Process Engineering

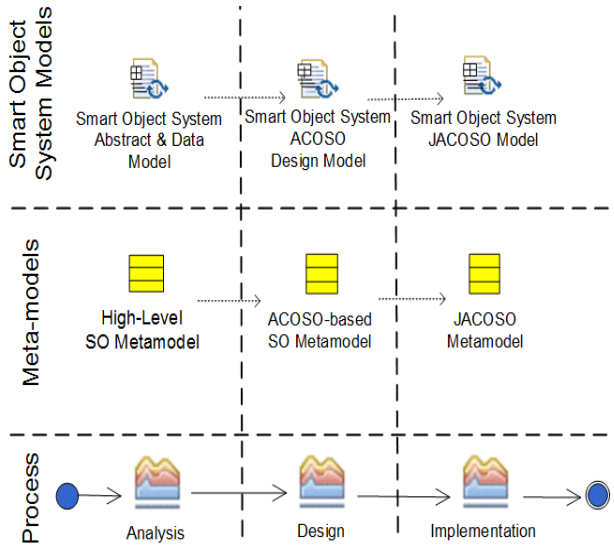


Fig. 2. Relationships among ACOSO-Meth metamodels at different phases

Modelling (SPEM) 2.0 [41]), ACOSO-Meth supports the analysis phase through a high-level model describing main basic SO features. Such model is specialized and better detailed, thus evolving at the design and implementation phases. In particular:

- at the *Analysis* phase, a High-Level SO Metamodel is exploited;
- at the *Design* phase, an ACOSO-based SO Metamodel specializes the analysis-level metamodel in order to model the functional components of the system, their relationships and interactions; and
- at the *Implementation* phase, a JACOSO (JADE-based ACOSO) Metamodel specializes the ACOSO-based SO Metamodel with respect to a particular implementation based on the JADE platform [42].

Every phase introduces new features and a higher degree of detail in the metamodels, maintaining at the same time strong relations with the higher-level metamodels. This allows the straightforward transition from the analysis to implementation phases, seamlessly supporting the translation of high-level system models into design-level agent-oriented platform-independent models that, in turn, may be refined into agent-oriented implementation platform-dependent system models.

A. System Analysis

The metamodel portrayed in Fig. 3 is a very high level metamodel, since its components may characterize an ecosystem of SOs [70] in any application domain, e.g., smart cities, smart factories, and smart homes. In fact, it models the main aspects of a generic SO/SO ecosystem in a very straightforward way, sharing similar characteristics with IEEE P2413 [43], AIOTI [40] and IoT-A [9] reference models. As matter of fact, main coarse-grained SO concepts (namely SO physical / virtual representation, SO user, SO service, and SO device) recur in all the aforementioned models, as well as in the High-Level SO Metamodel, as shown in Table IV.

TABLE IV
COMPARISON OF MAIN ENTITIES OF SO's METAMODELS OF ACOSO-Meth, IEEE P2413, AIOTI and IoT-A.

ACOSO-Meth High-Level SO MM	AIOTI SO MM	IoT-A SO MM	IEEE P2413 SO MM
SO	Virtual Entity	Virtual Entity	Virtual Entity
SO Physical Properties	Thing	Physical Entity	Physical Entity
SO Device	IoT Device	Device/Resource	IoT Device
SO Service	IoT Service	Service	N/A
SO User	User	User	User

To fully support the SO analysis phase, ACOSO-Meth High-Level Smart Object Metamodel exposes further features, reported by means of a UML class diagram in Fig. 3. These features describe both static (e.g., SO creator) and dynamic (mainly related to the services provided, e.g., quality-of-service indicators) SO characteristics.

They are categorized in five main groups:

- **SO BasicInfo** comprises basic SO information. In detail, the *Status* contains a list of variables, given as pairs <*name*, *value*>, that capture the SO state; *Location* represents its geophysical position (expressed in absolute terms by specifying latitude and longitude and/or in relative terms through the use of location tags); *PhysicalProperty* describes a physical property of the original object without any hardware augmentation and embedded smartness (it contributes to determining its scale); *FingerPrint* comprises immutable SO information like the SO identifier (or Id, which allows its unique identification within an IoT system), *SOCreator* that creates the SO for personal use, business or research purposes, *SOType* represents an SO type, e.g., a smart pen, smart building, and smart city, and *QoSParameters* defines one or more QoS parameters associated to the SO, e.g., precision, reliability, and availability.
- **SO Service** models a digital service provided by an SO. Each service is characterized by a name, description, type (e.g., sensing and actuation), input parameter type and return type. Each *Service* is implemented by one or more *Operations* and by zero or more *QoSIndicators* whose associated values are provided. In detail, an Operation, which defines an individual operation that may be invoked on a service, has a description, a set of input parameter types necessary for its invocation, and a return type related to its output value.
- **SO User** identifies an entity using the services provided by an SO. In particular, SO Users can be humans **(**representing the classical man-machine use relationship), SmartObjects **(**representing a less conventional use relationship, in which SOs take advantage of services exposed by other SOs and vice versa) or DigitalSystems (representing a generic digital entity, like a web server, software agent, robot or a more complex system).
- **Augmentation** defines the hardware and software characteristics of a device that allows augmenting the physical object and making it smart. A device can be specialized in one of the following three categories: (i) *Computer*, which represents the features of a processing unit of the SO, e.g., PC, smart-phone, and embedded computer; (ii) *Sensor*, which models the characteristics of a sensor node of the SO; and (iii) *Actuator*, which models the characteristics of an actuator node of the SO.
- **SO Aggregation** supports aggregation among SOs. In particular, a complex SO (e.g., a Smart City) may physically or logically aggregate other SOs to provide more advanced and integrated services.

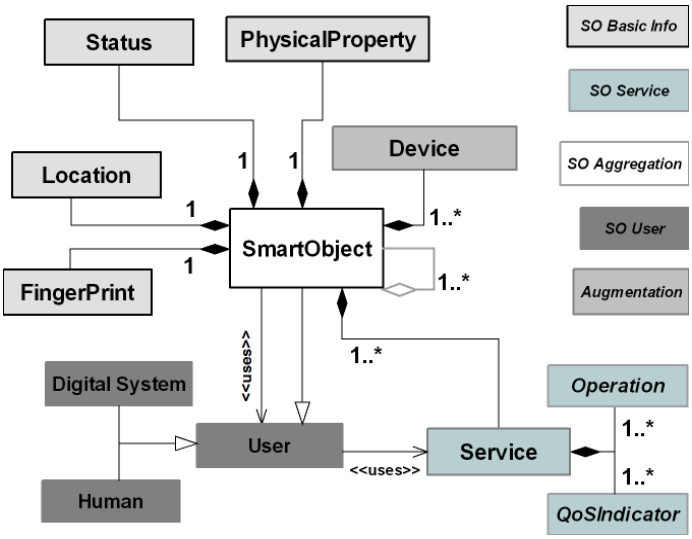


Fig. 3. Analysis Phase: High-Level SO Metamodel

B. System design

A High-Level SO Metamodel at the analysis level is refined to obtain an ACOSO-based SO Metamodel (Fig. 4), which allows, at the design level, agent-based modelling of the functional components of an IoT system, their relationships and interactions. An ACOSO-based SO Metamodel is suitable for modelling both basic IoT building blocks (e.g., basic devices as sensors and actuators, and smart objects) and more complex IoT components (e.g., WSNs and RFID systems) and represents the cornerstone of ACOSO [17], a middleware for the development, management and deployment of agent-oriented Cooperating Smart Objects (CSOs). ACOSO middleware provides an agent-oriented programming model for effectively realizing CSOs in any IoT application context requiring distributed computation, proactivity, knowledge management and interaction among SOs/sensors/actuators, thus fulfilling both system- and things-levels requirements identified before. According to the ACOSO-based SO Metamodel, an SO is modelled as an event-driven, lightweight and platform-neutral agent, whose lifecycle is specified in terms of *Behavior*. Behavior consists of one or more state machine-based components named *Tasks*. They can refer to internal system operations (*SystemTask*, e.g., SO shutdown/reboot/standby) required for the management of the agent lifecycle, or to user-defined operations (*UserDefinedTask*) defining specific SO-oriented and/or application-oriented functionalities of different SOs. SO Tasks are driven by *Events* according to the following model [17]: whenever the SO has to be notified (e.g., an incoming message

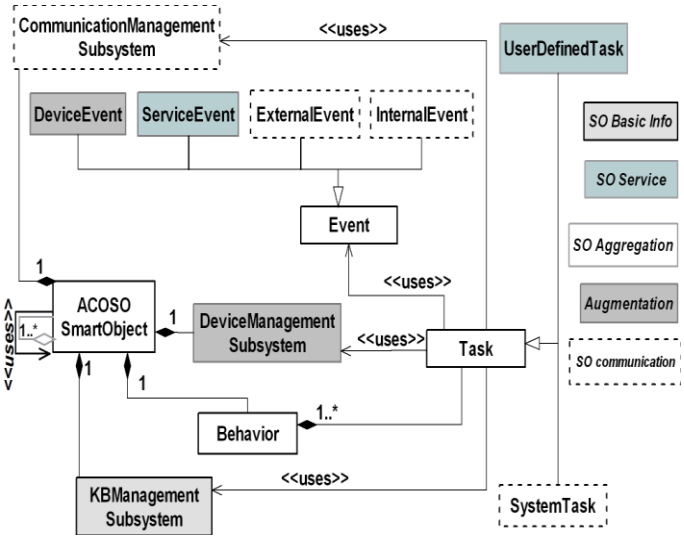
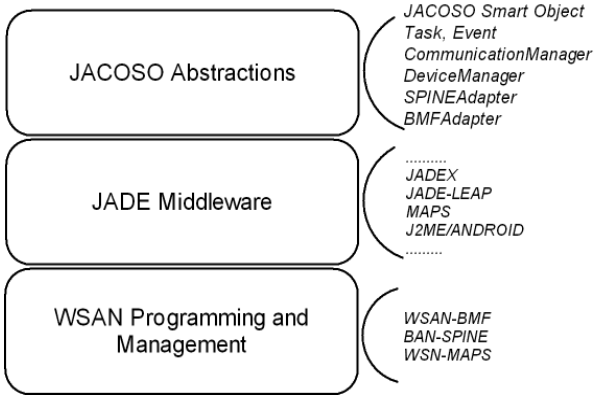


Fig. 4. Design Phase: ACOSO-based SO Metamodel

or a user request has arrived, an internal system operation is over), a specific event is created; hence, the event activates one or more Tasks according to its own event type and event source. Events are classified into: (i) *InternalEvent* (the event source is an SO internal component), raised to notify information/request/error messages coming from an internal SO module; (ii) *ExternalEvent* (the event source is an SO external entity), raised to notify information/request/error messages sent from entities external to the SO; (iii) *DeviceEvent* (the event source is an SO device), raised to notify information/error messages produced by the SO sensors, actuators, etc.; and (iv) *ServiceEvent*, raised from internal, external or device event sources, which specifically drives UserDefinedTasks to define application-oriented functionalities. The ACOSO-based SO metamodelling entities (Fig. 4) are categorized into four main groups:

- **SO Basic Info**: Basic information is spread between the SO itself and KBManagementSubsystem. The latter handles information pertaining its global current state, inference rules and other useful data that can be shared among tasks.
- **SO Service**: Services provided by SOs are encapsulated/implemented in specific application-level UserDefinedTasks. They are highly customizable, easily programmable, and interact with other SO components through ServiceEvents.
- **Augmentation**: the DeviceManagement Subsystem allows the management of sensors, actuators and devices embedded into SO. The interactions with such augmentation devices, regardless of their specific technology or protocol, are conducted through DeviceEvents.
- **SO Communication:** the CommunicationManagementSubsystem provides a common interface enabling communication toward the SO itself (through InternalEvents) or toward external entities (by means of ExternalEvents).



- Fig. 5. JACOSO three-layered architecture

C. System Implementation

In order to obtain the metamodel for supporting the implementation phase, we have implemented the ACOSO-based SO metamodel by using the JADE platform [42]. JADE is selected mainly for the following reasons: (i) it is an FIPA-compliant, well-known and Java-based agent middleware; (ii) it is open-source, has a spread community and, over the years, has evolved (e.g., JADEX [67], JADE-LEAP) to run atop novel and heterogeneous computing systems such as Java Micro Edition-enabled and Android-supported devices, as well as on sensor nodes constituting heterogeneous WSNs (Fig. 5); (iii) its middleware provides an effective agent-oriented management/communication infrastructure, that comprises an Agent Management System (AMS), ACL-based message transport system and Directory Facilitator (DF). In particular, DF supports agent service discovery, and has been extended with an agent-oriented interface [44] to allow SOs to register, index, and search on the basis of their specific functional and/or non functional features (e.g., Location, FingerPrint, and provided Services) introduced in the High-Level SO Metamodel of Section IV-A (these features are represented through metadata descriptions in a JSON format, which is lightweight, easy to read and to manually write, as well as to analyze and to automatically generate). Indeed, differently from general-purpose JADE agents, SOs have a strong "situatedness" and may seamlessly appear and disappear, but they may also evolve on the basis of some learning models or extemporary interactions with other SOs. An enhanced Directory Facilitator, providing a dedicated and dynamic SO discovery service, is thus fundamental. The metamodel shown in Fig. 6 refers to JADE-based implementation of the ACOSO-based SO Metamodel, named hereafter JACOSO.

Considering the inheritance relationship from the ACOSO-based SO model and the JADE components, hereinafter we present only the implementation components that characterize the JACOSO SO metamodel with reference to the related macro-components:

- **SO Basic Info:** JACOSO SO basic information is spread between the JADE-based agent itself and an internal knowledge base. The latter contains also the current values of the variables constituting the inference rules required for an SO decision-making process. Information, inference rule variables and configurations that need to be

provided at the SO instantiation (e.g. for the JACOSO SO devices), are set by *ConfiguratorTask*.

- **SO Service**: services are defined as UserDefinedTasks implemented as JADE-based behaviors. The application logic encapsulated in UserDefinedTasks can exploit the JACOSO SO inference rules required for the SO decision-making process by interacting with InferenceRuleTask.
- **Augmentation:** DeviceManager, by means of different DeviceAdapters, interfaces JACOSO SO with heterogeneous augmentation devices. In particular, BMFAdapter and SPINEAdapter allow the management of Wireless Sensor and Actuator Networks (WSANs) and Body Sensor Network (BSN) respectively through BMF (Building Management Framework) [45] and SPINE (Signal Processing In-Node Environment) [46] frameworks. BMF is a domain specific framework, expressly conceived for the management of WSAN in the context of environment monitoring and building automation; while SPINE is designed for efficient management of BSNs. Both SPINE and BMF comprise networks of heterogeneous devices (e.g., Shimmers, Telos-B and MICA2 sensor motes, Android-based devices, and conventional computers) based on typical IoT standards (e.g., IEEE 802.15.4, ZigBee, 6LowPan, Bluetooth) and they interact with JACOSO by means of the related deviceAdapters.
- **SO Communication:** CommunicationManager enables JACOSO SO to flexibly support different communication patterns by just implementing appropriate CommunicationAdapters. In particular, ACLCommunicationAdapter allows a direct message passing of ACLMessages [42] between a sender and receiver; TopicPSAdapter, instead, realizes an asynchronous one-to-may communication in which ACLMessages sent by a publisher are only received from those who have subscribed the related topic and operate in the same platform. JACOSO guarantees high versatility, allowing the implementation of SOs and IoT systems of different scales and within different application scenarios just by re-using and/or re-implementing some components. Such components that can model new SO functionalities are indicated as *hot spots* and are (i) *ConfiguratorTask* that sets up specific SO Basic Info, SO components and Tasks at the SO instantiation time; (ii) *UserDefinedTasks* that encapsulate a specific SO application logic; (iii) *CommunicationManager*, because it should be set up to handle new *CommunicationAdapters* realizing other communication patterns (e.g., web services and sockets) beside or instead of the existing ones (direct message passing and publish/subscribe); and (iv) *DeviceManager* if other *DeviceAdapters* are introduced to interface JACOSO SO with specific devices. On the opposite, JACOSO architectural blocks that do not need to be changed are the so-called *frozen spots*.

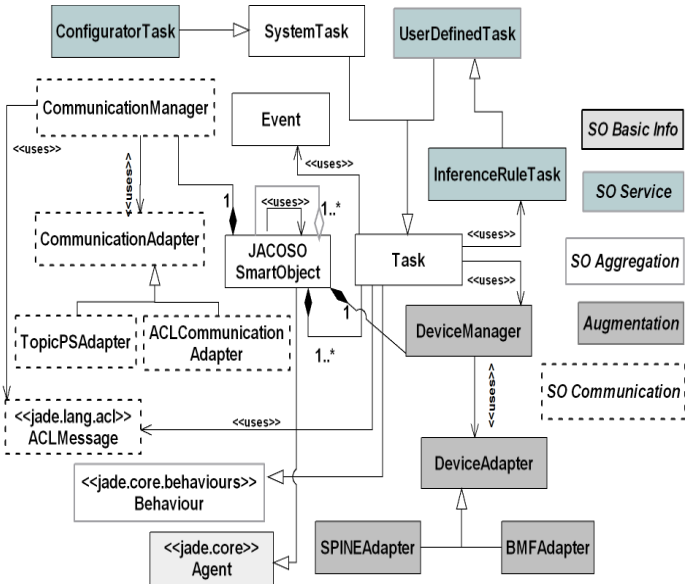


Fig. 6. Implementation phase: JACOSO SO Metamodel

D. Discussion

From analysis to implementation, SO-related concepts evolve, being refined from high-level abstractions to implementable software components. In Table V, such concepts are listed and hence their mapping to different development phases of analysis, design and implementation is presented. First, it should be noted that an SO at the analysis phase is described as a very abstract entity, becoming an agent only at the design phase: since the High-Level SO Metamodel is unbound from any paradigm, it may be used as a reference model, similar to AIOTI [38] and IoT-A [18] ones. Moreover, the abstract SO User introduced at the analysis phase is further replaced by an agentified user (ACOSO SO, if it aggregates and exploits other SOs) at the design phase and by a JADE agent at the implementation phase (JACOSO SO, if it aggregates and exploits other SOs). SO Basic Info, individually described in a High-Level SO Metamodel, is spread between the SO itself and its knowledge base in the design and implementation phases. Regarding the augmentation, SO devices are simply reported at the analysis phase while their management (DeviceManagementSubsystem and DeviceEvent) and actual interfacing (DeviceAdapters and DeviceManager) are elicited respectively in the ACOSO-based and JACOSO SO metamodels. SO communication features, not explicitly highlighted in a High-Level SO Metamodel, are introduced at the design and implementation phases. Indeed, an ACOSO-based SO Metamodel presents a CommunicationManagementSubsystem exploiting ExternalEvents and InternalEvents while a JACOSO-based SO Metamodel introduces the CommunicationManager, customizable CommunicationAdapters, and FIPA-compliant ACLMessages infrastructures. Finally, an SO Service concept is first abstractly presented in terms of operations and QoS indicators at the analysis phase and then refined as an application-level UserDefinedTask and ServiceEvents in the design and implementation phases.

TABLE V
EVOLUTION OF THE SO MAIN CONCEPTS FROM THE ANALYSIS TO THE IMPLEMENTATION PHASE

Concept	High-Level SO Metamodel (Analysis Phase)	ACOSO-based SO Metamodel (Design Phase)	JACOSO SO Metamodel (Implementation Phase)
SmartObject	High-level (conceptual) entity	ACOSO Agent, Behavior, Task	JADE Agent, JADE Behaviors, Task
User	SO, human, digital system	Agentified user, ACOSO SO	JADE Agent, JACOSO SO
SO Basic Info	FingerPrint, Location, Status, PhysicalProperty	(Agent) SmartObject, KBManagementSubsystem	(JADE Agent) SmartObject, InferenceRuleTask variables
Augmentation	Device	DeviceManagementSubsystem, DeviceEvent	BMFAdapter, SPINEAdapter, DeviceManager
SO Communication	Not explicitly highlighted	CommunicationManagementSubsystem, InternalEvent, ExternalEvent	TopicPSAdapters, ACLCommunicationAdapter, CommunicationManager, ACL Messages
SO Service	Service	UserDefinedTask, ServiceEvent	UserDefinedTask, InferenceRuleTask, ServiceEvent

With regard to the SLRs and TLRs fulfilment, ACOSO-Meth provides a systematic and full-fledged approach (SLR_6) to the SO development, exploiting (i) the agent abstraction to virtualize and homogenize the different SOs to be developed (SLR_1); (ii) a flexible and modular communication infrastructure (comprising at design phase the CommunicationManagementSubsystem and at implementation phase the Communication Manager with its CommunicationAdapters) to enable voluntary communication among different paradigms and data formats (SLR_2 and SLR_5); (iii) a customizable augmentation infrastructure (comprising the DeviceManagementSubsystem at design phase and DeviceManager with its DeviceAdapters) to enable interoperability among heterogeneous IoT devices (TLR_1); (iv) well-known FIPA-compliant interfaces and ontology in order to straightforwardly access SO functionality, historical and contextual information (leveraging at the design phase on the KBManagementSubsystem and at the implementation phase on SO internal knowledge bases, SLR_3 and SLR_4); (v) the ACOSO-middleware (in particular its domain-neutral metamodels and programming techniques) and the JADE facilities (e.g., AMS and DF) to speed up SO prototyping and evolution (TLR_4), and support their augmentation variation (TLR_2) and decentralized management (TLR_3); and (vi) a revised scale concept to unambiguously characterize SO-based IoT systems and possibly enable their comparison (SLR_7 and TLR_5).

V. Case study: Smart University campus

In this section, we present the application of ACOSO-Meth for engineering a complex Smart University Campus IoT ecosystem, specifically prototyped at the University of Calabria and named Smart UniCal. Several references to Smart University/Smart Campus scenarios are available in the literature [47-53] and, regardless of particular goals or implementations, they all present "comfortable and user-tailored environments, rich in innovative services".

Our Smart UniCal system (Fig. 13) is an aggregated SO composed by a Smart Bridge (dotted yellow bordered area) and Smart Departments (yellow bordered area), spanning multiple adjacent buildings, which contains smart rooms such as Smart Lab and Smart Office. Smart UniCal SOs have been characterized respectively in "L"arge (i.e., the SmartBridge), "M"edium (i.e., the SmartDIMES) and "S"mall scale (i.e., the SmartSenSysCalLab) SOs, according to the considerations reported in Section II. In particular, Table VI reports the list of services provided by Smart UniCal SOs:

- SmartBridge provides a cyber-physical service for a structural health monitoring [54] purpose;
- SmartDIMES (Department of Informatics, Modelling, Electronics and Systems Engineering), namely a Smart Department, provides a cyber-physical service to remotely control department spaces and facilities, e.g., HVAC and lights, aiming to save energy; and
- SmartSenSysCalLab, namely a Smart Lab, provides cyber-physical services to laboratory users who are supported in their daily activities.

It should be noted that the domain-neutrality of the ACOSO-Meth and ACOSO middleware allows supporting the development of the different kinds of Smart Unical's SOs and related services by keeping the same methodological approach and by exploiting the same metamodels and programming techniques. In the following, the descriptions of the SmartBridge in the analysis, design and implementation phases are provided according to the ACOSO-Meth. Finally, the *Smart UniCal* performance evaluation is presented.

A. Analysis

Smart Bridge (Fig. 14) is a large-scale SO physically based on the "Pietro Bucci" bridge, which crosses the Unical campus for 1.22 Km, linking together all the 14 university departments (spread among different building units called cubes). Its Administrator can query its status, specifically the currently recorded vibration, or use the smartVibration service for monitoring the bridge's structural health [54]. Service smartVibration allows the analysis of the vibrations generated by the transit of vehicles and pedestrians upon the bridge.

TABLE VI
SMART OBJECTS CONSTITUTING SMART UNICAL ALONG WITH THEIR PROVIDED SERVICES

Scale	Smart Object	Service	Description
S	SmartSenSysCalLab	smartWellness	Correct lifestyle suggestions
		smartComfort	Workplace conditions improvement
M	Smart DIMES	smartMonitoring	Indoor environmental monitoring
L	Smart Bridge	smartVibration	Bridge vibrations monitoring

Fig. 13. The Smart UniCal infrastructure: the SmartBridge part (dotted yellow bordered area) which crosses the SmartDIMES (yellow bordered area with building identification codes).

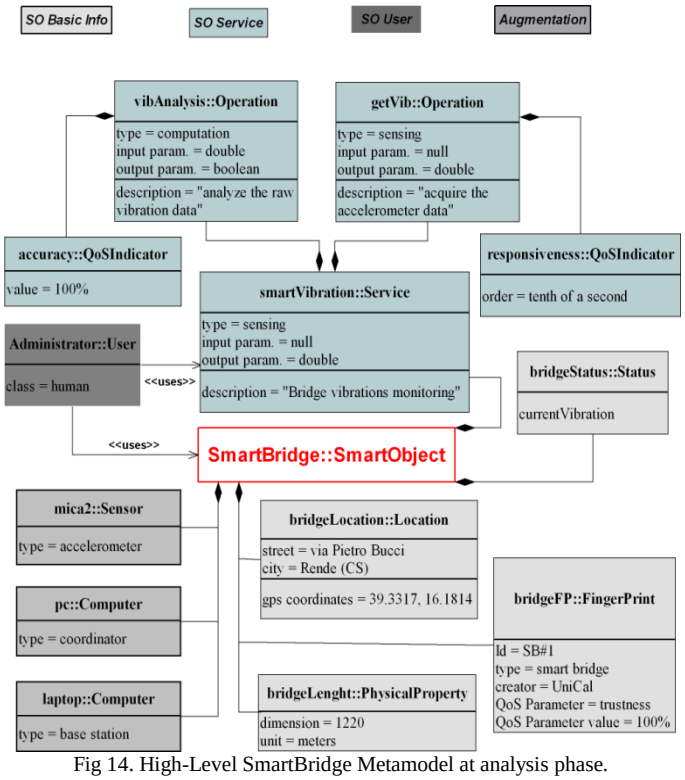


Fig 14. High-Level SmartBridge Metamodel at analysis phase.

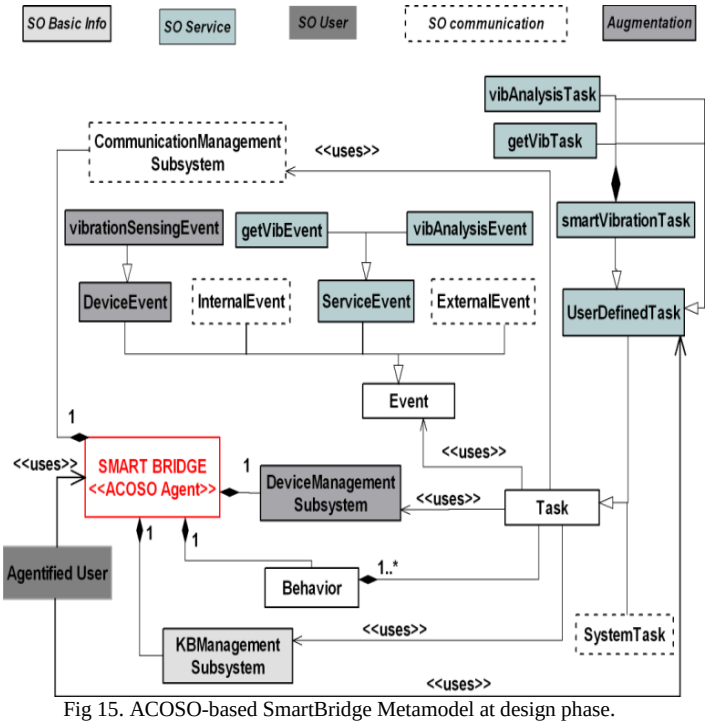


Fig 15. ACOSO-based SmartBridge Metamodel at design phase.

If the sensed vibrations reach warning thresholds, the service notifies such event to the Administrator. In order to provide such service, SmartBridge is augmented through different devices, including accelerometer sensor nodes and laptop base stations,coordinated by a PC acting as a main coordinator. In more details, smartVibration service has two basic operations: getVib that exploits accelerometer sensors on the bridge to accurately sense the vibrations, and vibAnalysis that exploits SmartBridge's computing devices to elaborate the raw vibration data acquired and to compare them with the defined thresholds.The getVib operation has a response time in the order of second while vibAnalysis detects all the vibrations exceeding the warning thresholds with 100% accuracy.

B. Design

SmartBridge's High-Level metamodel is refined at the design phase, resulting in Smart Bridge's ACOSO-based metamodel as shown in Fig. 15. In particular, SmartBridge is modeled as an ACOSO-based agent and SO Users as generic agents. The smartVibration service and the vibAnalysis and getVib related operations are modeled as UserDefinedTasks (smartVibrationTask, vibAnalysisTask and getVibTask respectively) driven by the corresponding ServiceEvents (getVibEvent and vibAnalysisEvent). The vibrationSensingEvent, instead, allows interfacing the accelerometer sensors with SmartBridge, e.g., providing the raw vibration sensed data.

C. Implementation

Smart Bridge's ACOSO-based metamodel is refined at the implementation phase, resulting in Smart Bridge's JACOSO-based metamodel (Fig. 16). In particular, in this phase the generic agentified SmartBridge is specialized into a JADE-based agent, as well as the agentified SO User. ACLCommunicationAdapter allows SmartBridge exploiting a

direct ACL-based messages exchange mechanism. SmartBridgeInferenceRuleTask contains both inference rules required for a SmartBridge decision-making process (Table VII) and current values of the variables constituting such inference rules. UserDefinedTasks (smartVibrationTask, vibAnalysisTask and getVibTask) implementing smartVibration and related events (vibAnalysisEvent and getVibEvent) are modeled as JADE Behaviour, while BMFAdapter interfaces SmartBridge with its devices. Interaction diagram of Fig. 17 illustrates the methods realizing the Smart Bridge's Smart Vibration service.

D.Technical implementation

In the following, some key technical implementation details of the *Smart UniCal* system, related to the used IoT devices and to the implemented JACOSO-based software components, are described. In particular, Fig. 18 (a-c) shows some technical deployment snapshot of the IoT devices of SmartBridge, SmartDIMES and SmartSenSysCalLab. Table VIII shows the characteristic of main hardware/software devices. Such heterogeneous IoT devices along with Android-based devices and conventional computers adopting different technologies (e.g., IEEE 802.15.4, Wifi, and Bluetooth) and managed by different frameworks (i.e., SPINE and BMF) have been made interoperable through the related deviceAdapters provided by the ACOSO middleware, independently from low-level network protocols or different communication paradigms through the exploitation of the related communicationAdapters. We focus on the implementation of the following services: (1) smartVibration service of SmartBridge, (2) smartMonitoring service of SmartDIMES, and (3) smartWellness and the smartComfort services of SmartSenSysCalLab.

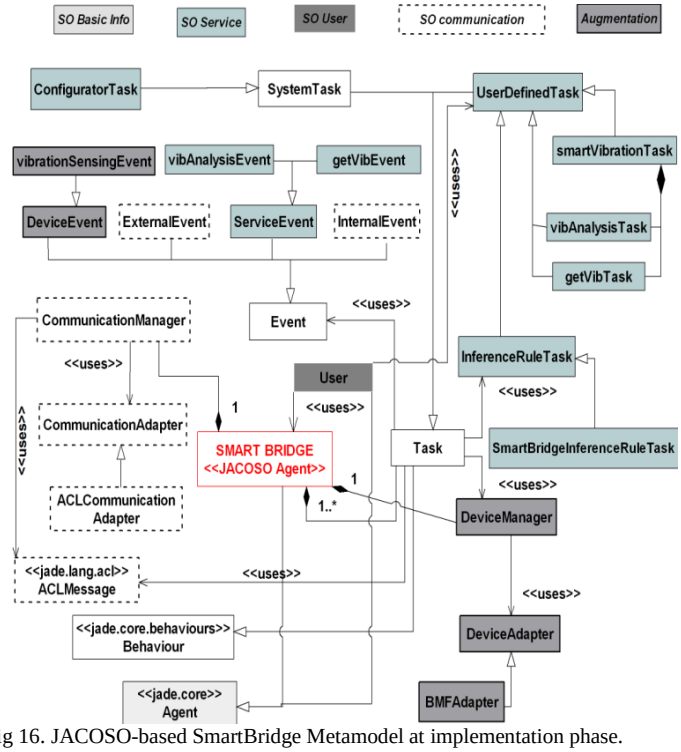


Fig 16. JACOSO-based SmartBridge Metamodel at implementation phase.

TABLE VII
EXAMPLE OF A RULE EMBEDDED IN THE SMARTBRIDGE INFERENCERULETASK

Rule#	Description
1	Alarm← currentVibration_transversalAxis>vibrationThreshold_transversalAxis v currentVibration_longitudinalAxis>vibrationThreshold_longitudinalAxis

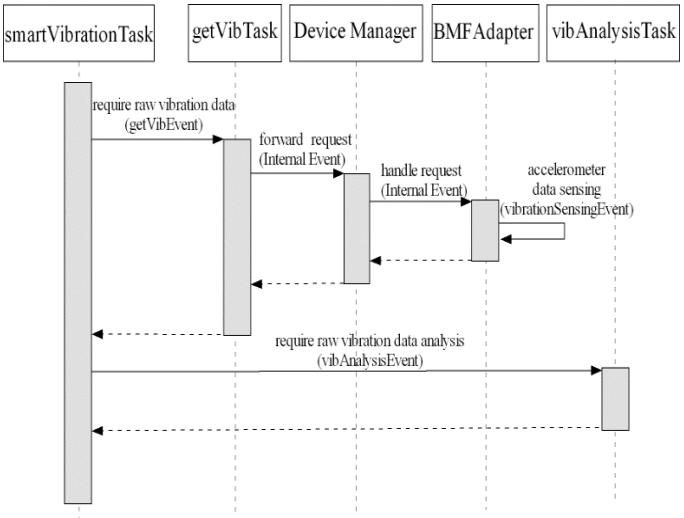


Fig 17. Interaction diagram of the Smart Bridge's Smart Vibration service

1) smartVibration is based on the data gathered by 90 Crossbow MICA2 devices. Every 27 meters, two of them are deployed facing each other and laying on a metallic beam that transversally passes through the axis of the bridge (Fig. 18(a)). Such network of Crossbow MICA2 devices is managed by 9 notebooks (placed in rooms in front of the bridge such that each notebook can manage data of its closest 10 motes), hosting the BMF application and collecting the data, while a central PC acts as a main coordinator and hosts the SmartBridge SO application. Each notebook works in a different subnetwork and all the notebooks interact with the main coordinator through an IP-based WiFi UniCal Intranet; Crossbow MICA2 devices, instead, are connected to their associated notebook through the 802.15.4 wireless protocol. Totally, 20 non-overlapping subnetworks have been used to realize this service.

2) smartMonitoring is based on 43 Telos-B-based indoor environmental sensors, i.e., humidity, temperature, light, and presence sensors, and on 20 Telos-B-based actuators (i.e., smart plugs) deployed within 18 DIMES rooms (Fig. 18(b)). In detail, at least two devices (one sensor and one actuator) have been deployed for each of the 18 monitored environments, located in different cubes. Each monitored environment is associated to a laptop (the environments located at the same floor of the same cube share the same one) hosting a base station and running the BMF application (totally, ten laptops have been used and interconnected through 10 overlapping subnetworks). Each laptop interoperates with the associated sensors/actuators through the BMFAdapter: it allows the collection of sensed data from the sensing devices to the base station, and the forwarding of

commands from a base station to actuating devices. The SmartDIMES application is hosted in a separate laptop. The SmartDIMES administrator, through such application, can transparently manage all the environments and send both request and configuration messages to the deployed Telos-B motes and smart plugs.

3) smartWellness provides customized and real-time hints to SenSysCal lab users by displaying notifications on their personal smartphones and/or laptop monitors. Data coming from 12 light/presence Telos-B sensors (one for each of the ten SenSysCal desktops, one at the entrance and one in the middle of the lab) and from 30 users wearable Shimmer sensors (three for each user, placed at user wrist, waist and leg) are forwarded by means of BMFAdapter (environment data) and of SPINEAdapter (wearable data), to a base station. The base station is a laptop running the SmartSenSysCalLab application that collects the overall data, elaborates them and sends back customized notifications to users (specifically, on their Twitter profile or on the computer screen placed on their desktop). The same base station, in the context of a *smartComfort* service and through BMFAdapter, periodically queries the light intensity value to every Telos-B sensor deployed atop one of the 12 user desktops. In case of poor lighting, the corresponding desktop lamp is switched on through its smart plug. The aforementioned devices that contribute to realize the SmartSenSysCalLab services (Fig. 18(c)) are connected to the local laboratory subnetwork.

E. Performance Evaluation of Smart UniCal

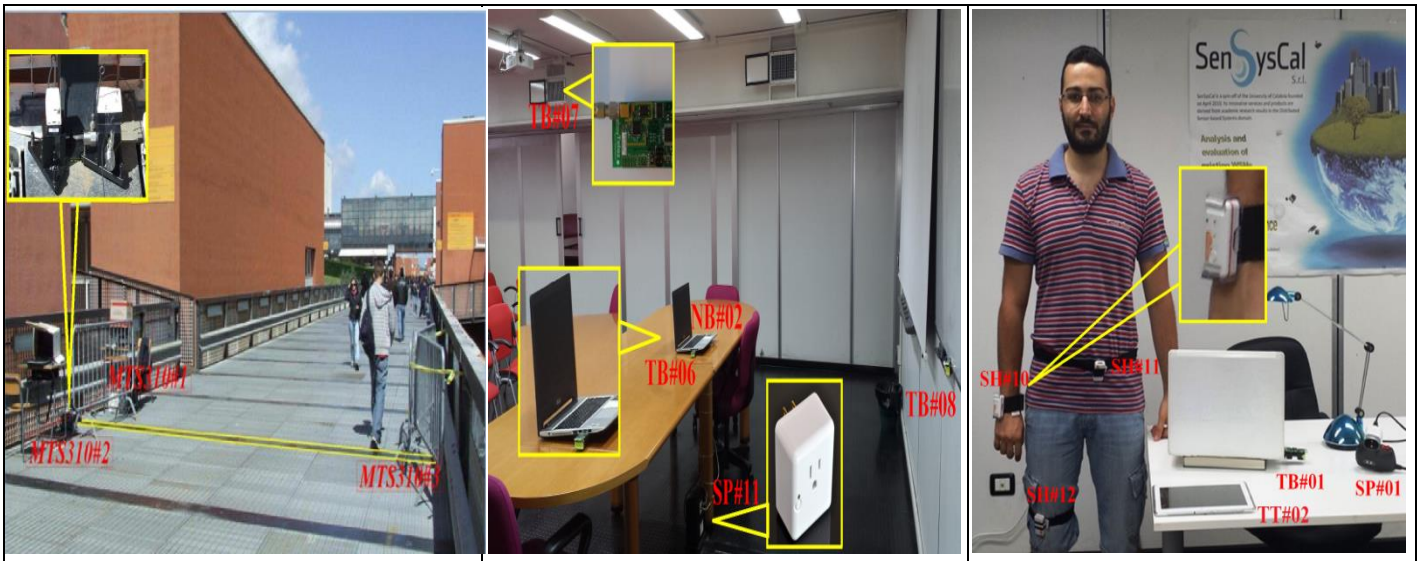
As presented in Section IV, ACOSO-Meth is based on a JACOSO SO Metamodel for system implementation, deployment and execution. In the following, the *Smart UniCal* performance evaluation is presented to assess the suitability of the ACOSO-Meth implementation phase in actually supporting efficient small-, medium- and large-scale IoT systems. Indeed, SmartBridge, SmartDIMES and SmartSenSysCalLab SOs and their aggregated IoT devices, are evaluated when providing their specific services (see below). However, in order to define the proper scenario size (number of SOs and their distribution in different subnetworks) that effectively enables the developed services, preliminary tests were conducted to evaluate SOs performance, thus analyzing possible bottlenecks (Fig. 19) in the information exchange (IE) phase. Please note that the deployment stage and performance evaluation require a significant effort, especially due to the number of SOs involved: fortunately, we can leverage on our previous experience in the fields of WSNs and cyber-physical systems [45, 46, 68] to speed up the identification of operation modalities and performance indices, as well as the SO monitoring and data gathering.

In particular, we considered SOs exchanging 2KB fixed length simple FIPA-compliant data messages by following either a Client/Server (C/S) or a Peer-to-Peer (P2P) paradigm. As some services are intrinsically centralized or distributed, they can be implemented following either a C/S or a P2P paradigm. Moreover, we considered IoT device data sources (or simply data sources) with either stochastic normal distribution (N, with mean = 0.5 msg/s and variance = 0.2 msg/s) or deterministic (D, 1 msg/s) message generation rate (MGR) models. Given the communication paradigms and MGR models, we focused on two fundamental network-oriented performance indices for distributed SOs when providing specific services collaborating with each other: (i) message delivery ratio (MDR); and (ii) round trip time (RTT). In Fig. 19, however, only the RTT values calculated in small- (SmartSenSysCalLab), medium- (SmartDIMES), and large-scale (SmartBridge) scenario are shown, as the MDR values are always 100%, being JADE communications based on TCP connections, thus fully reliable. Fig. 19(a) highlights how the increase of the involved SOs in the small-scale scenario adversely affects RTT, which rapidly grows due to the network congestion. In Fig. 19(b), differently from Fig. 19(a), where SOs are supported by just one network within a squared grid of side 10 m, SOs are now distributed in 10 different sub-networks within a squared grid of side 250 m. In such a deployment area, it happens that adjacent networks interfere with each other, since their coverage radii overlap. Nevertheless, a better SO distribution implies less congestion and lower RTT. For example, the RTT of 100 SOs distributed in 10 subnetworks is definitively lower than the RTT of the same number of SOs deployed in one network. Finally, in the large-scale scenario of Fig. 19(c), the SOs are distributed in 20 non-overlapping subnetworks within a squared grid of side 1000 m. Differently from the small- and medium-scale scenarios, in Fig 19(c), it can be noted that increasing the number of SOs has a little impact on RTT. Compared to the same configuration of a medium-scale scenario, RTT values are lower. In fact, in a large-scale scenario, networks are deployed in a wider area. Thus, they do not interfere with each other and consequently both congestion and RTT decrease.

TABLE VIII
MAIN HW/SW CHARACTERISTICS OF THE IOT DEVICES USED TO IMPLEMENT SMART UNICAL SOs AND THEIR SERVICES.

Device	Main characteristics	SO/Services
MICA2	*OS*: TinyOS. *CPU*: Atmel Atmega 128L (8 bit bus, 8MHz clock). *Memory*: 4K Ram 128K Flash 512K EEPROM. *Radio:* 802.15.4 compatible CC2420. Expansion board (2-axis accelerometers). *Battery:* 2X AA batteries (4000-5000 mAh in total, depending on the cell)	SmartBridge/ smartVibration
Telos-B	*OS*: TinyOS. *CPU*: TI MSP430F1611 (16-bit bus, 4-8MHz clock). *Memory*: 10K Ram 48K Flash 1M EEPROM. *Radio*: 802.15.4 compatible CC2420. On-board sensors (humidity, temperature light sensors). *Battery:* 650 mAh	SmartDIMES/ smartMonitoring SmartSenSysCalLab/ smartComfort
Shimmer	*OS*: TinyOS. *CPU*: TI MSP430F1611 (16-bit bus, 4-8MHz clock). *Memory*: 10K Ram 48K Flash. *Radio*: 802.15.4 compatible CC2420 On-board sensors (3-axis accelerometer) *Battery:* 650 mAh	SmartSenSysCalLab/ smartWellness

For example, given 50, 70 and 100 SOs, RTT values in the large-scale scenario significantly decrease by comparing those in the medium-scale scenario. The aforementioned SOs performance evaluation has provided important insights to define, for each SO and for each SO service, the best operation modalities in the Smart UniCal ecosystem. Such modalities are detailed in Table IX by specifying: the number of embedded devices (#EDev), number of involved subnetworks (#SubNets), evaluation time (EvTime), message length (ML) and deterministic message generation rate (D-MGR). Given the SOs and SO service operation modalities as shown in Table IX, the Smart UniCal performances have been evaluated in terms of MDR and RTT, IoT devices energy and memory consumptions (base stations are powerful and less constrained than motes and they are typically plugged to the mains electricity**,** such that they can be easily recharged), and the provided results reported in Table X. Such performance indices have been chosen to characterize SO performance both functionally and non-functionally: indeed, they provide useful insights about services responsiveness and reliability (according to the performance indices previously outlined to describe the IE phase), but also about the required resources (energy and memory, in particular). The latter is a relevant aspect considering that most of the IoT devices are resource-constrained.

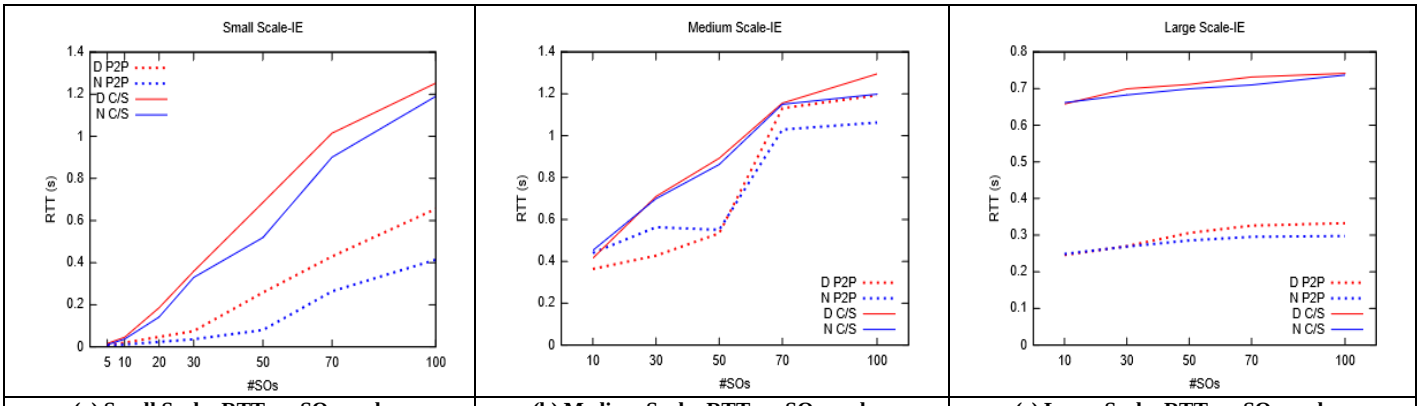


(a) Smart Bridge: arrangement of two MICA2 motes (reported in the picture with the codes MTS310#1 and MTS310#3) on the bridge axis, with the goal of monitoring the bridge vibrations generated by the transit of vehicles and pedestrians. Sensed data are locally collected by a notebook acting as a base station (MTS310#2).

(b) SmartDIMES: arrangement of some Telos-B motes (TB#06-TB#08) providing real-time information about the presence around them and the room lights' current status (on/off). Sensed data are locally collected by a notebook acting as a base station (NB#02). The smart plug (SP#11) can switch on/off the projector and the room lights in case of energy waste.

(c) SmartSenSysCalLab: three Shimmer motes (SH#10-SH#12) send their data to the BSN coordinator (TT#02) to recognize current user posture. The Telos-B mote (TB#01) placed atop the user desktop provides information about the current light intensity: if it is dim, the smart plug (SP#01) turns the lamp on. All the sensed data are collected by a base station.

Fig 18. Snapshot of the IoT devices of (a) SmartBridge, (b) SmartDIMES and (c) SmartSenSysCalLab



(a) Small Scale: RTT vs. SO number | **(b) Medium Scale: RTT vs. SO number** | **(c) Large Scale: RTT vs. SO number**

Fig 19. SmartSenSysCal, SmartDIMES, and SmartBridge performance evaluation considering different communication paradigms (C/S or P2P) and MGR models (D or N)

TABLE IX
THE OPERATION MODALITIES OF THE SMART UNICAL SMART OBJECTS

SO	Service	#EDev	#SubNets	Operation Modality	EvTime (h)	ML (KB)	D-MGR (msg/s)
SmartSenSysCalLab (small scale)	smartComfort	25	1	Periodically (every minute), each desktop light intensity is acquired and sent to the base station	8	2	1
	smartWellness	42		Periodically (every minute), environmental and body data are acquired and sent to the base station			
SmartDIMES (medium scale)	smartMonitoring	73	10	Periodically (every 30 seconds) environmental data acquired and sent to the base station	12	2	1
SmartBridge (large scale)	smartVibration	100	20	Both periodically (every minute) and occasionally data are acquired and sent to the base station	12	2	1

TABLE X
SMART UNICAL PERFORMANCE EVALUATION

Service	MDR	RTT (s)	Residual Energy	Residual RAM-ROM
smartComfort (SmartSenSysCalLab)	100%	0.046	85%	63%-28%
smartWellness (SmartSenSysCalLab)	100%	0.059	57%	56%-17%
smartMonitoring (SmartDIMES)	100%	0.513	48%	60%-26%
smartVibration (SmartBridge)	100%	0.507	87%	78%-12%

The results reported in Table X confirm the RTT trends shown in Fig. 19 and JADE message system's high reliability, being based on the TCP protocol. Then, the increase of #EDev adversely affects both RTT, which grows due to the network congestion, and energy consumption, especially if also the evaluation time increases (in the case of smartVibration services, the residual energy is only slightly nicked since MICA2 capacity is bigger than those of Shimmer and Telos-B ones). Moreover, EDev deployment on different SubNets affects RTT more than #SubNets. In particular, by comparing the RTT values of SmartSenSysCalLab and SmartDIMES, it should be noted that when #EDev scarcely doubles, RTT increases tenfold; however, if there is no overlapping among the SubNets, then the performances are quite stable, even if #EDev and #SubNets increase, as in the case of SmartBridge. SO lifetimes varies depending on the provided service, devices' batteries, operation modalities and scenario configurations as reported in Tables VIII and IX. In particular, we have defined the Residual Energy of an SO *X* providing a service *s* by exploiting (all or a set of) its different D_i devices as

$$RE\ (Xs)=min\ \{batteryD_1.....batteryD_n\}$$

where $batteryD_i$ is the amount of power currently left in a D_i's battery that enables its correct working [71] in the context of service provision. Given such definition, *RE(Xs)* can vary from 100% (all SO devices involved in the service provision are full of energy) to 0% (at least one SO device has an energy shortage preventing it from correct working). With regard to the Smart Unical and testing, for the sake of simplicity, each SO in providing only a single service, SO service provision varies from 18 hours (SmartSenSysCalLab providing only the SmartWellness service) to 92 hours (SmartBridge providing only the SmartVibration service). Finally, memory consumption results highlight that IoT devices have enough free memory to deploy other in-node services or customized extensions.

VI. Conclusions

Using an engineering methodology is widely recognized as a fundamental practice in any system development process, since the manual and non-systematic application of complex techniques, methods and frameworks would very likely reduce effectiveness, increase development time and tend to be error-prone. The need for a full-fledged development methodology is particularly crucial in the case of IoT systems development, which exposes specific requirements (both at system- and things-level) to enable the dynamic cooperation among cyber-physical SOs over heterogeneous networks and the provision of even complex cyber-physical services. In this paper, such requirements have been elicited and inserted in a comparison framework (which may be notably reused to compare future work in the field), showing that, according to the state-of-the-art, a comprehensive methodology that systematically fulfils the needs for the SO-based IoT systems development still lacks. In such a context, agent-based computing represents a very effective paradigm for the modelling of SO-based IoT systems as well as an efficient technology for their development.

Thus, in this paper, the agent-oriented ACOSO-Meth methodology is presented and applied for engineering SO-based IoT systems of different complexities and scales. At the analysis phase, ACOSO-Meth provides a High-Level SO Metamodel that shares main features with emerging IoT architectural standards such as IEEE P2430, AIOTI and IoT-A domain models. At the design phase, the agent-oriented ACOSO-based SO Metamodel derived from the High-Level SO Metamodel is used. The ACOSO-based SO Metamodel supports an agent-based modelling of functional system components, their relationships and interactions. Finally, at the implementation phase, JACOSO (the JADE-based version of the ACOSO-based SO Metamodel) is exploited to implement SO-based systems through the well-known JADE platform. In this paper, ACOSO-Meth has been used (from the high-level system design to the concrete JACOSO-based implementation) to develop the *Smart UniCal* IoT system. *Smart UniCal* is a complex case study as it comprises heterogeneous SOs of different scales, is deployed in a real scenario (the University of Calabria in Italy) and provides

cyber-physical services related to structural, indoor space and wellness monitoring. In particular, the systematic application of ACOSO-Meth has significantly facilitated and speeded up all the *Smart UniCal* development phases: (a) the High-Level SO Metamodel supports the abstract analysis of the main *Smart UniCal* SO features and functionalities, facilitating their identification; (b) the agent-oriented design provides the adequate flexibility and effectiveness to fulfil the fundamental requirements at both system- and things- levels of *Smart UniCal*; finally, (c) the JADE-based implementation allows a rapid and efficient prototyping of the *Smart UniCal* ecosystem; this just demands the only effort of programming (by extension) the application-specific JACOSO hotspots that represent only 25% of the overall lines of code constituting the *Smart UniCal* software. It follows that, because of such modular and extensible approach, most of the code does not need to be customized according to the particular application requirements, but it can be directly reused. In brief, the ACOSO-Meth and its related ACOSO middleware provide effective and domain-neutral design and programming models along with efficient tools for the actual and full-fledged engineering of the *Smart Unical* in terms of a multi-agent system, significantly facilitating and speeding up all the development phases. Such benefits are not affected by the hand-made transitions among the analysis, design and implementation phases: an automatic transition, whenever possible, may speed up the ACOSO-Meth application while keeping the same effectiveness. Moreover, we have carried out a thorough performance evaluation of the *Smart UniCal* system. The performance evaluation provides (i) general results, highlighting significant issues related to the number of SOs and to the distribution of SOs among the subnetworks of the whole system, and (ii) specific results, acknowledging the efficiency of JACOSO middleware for the development of SO-based IoT systems. Moreover, the result analysis indicates that the definition of SO operating modes (e.g., tuning of system parameters, and choice of communication paradigms) should take into account the target scenario, by following therefore an application-driven approach.

Our on-going work is devoted to the development of the following strategic *Smart UniCal* services: an RFID-based people counting/identification system (smartTrack service) for SmartBridge, an RFID-based inventory system for SmartDIMES valuable stuff (smartInventory service), and an NFC-based system to automatically record SenSysCal users' attendances and their daily timetable (smartAttendance service). Future work is already planned towards six main directions: (i) Extending the current ACOSO-based approach to support the BDI paradigm [67], the topic-based communication among different platforms [42] by means of dedicated "mediator" agents, and creation of a tool for the automatic models instantiation and code generation; (ii) Integrating Cloud/Edge computing with the ACOSO-based approach to enhance system scalability and enable more critical real-time system responses [55, 72, 74]; (iii) Defining a simulation-driven phase after the design phase to assess an ACOSO-based design (both from functional and non-functional perspectives) before its actual implementation and deployment [56, 57]; (iv) Incorporating formal methods, e.g., Petri nets [58]-[63], for systematic design and property verifications of IoT systems into our design phase; (v) Defining mechanisms to automatize, where possible, the transition among the different phases constituting the methodology, thus speeding up the ACOSO-Meth application while keeping the same effectiveness, and (vi) Using data validation models for IoT domains [64, 73].

ACKNOWLEDGMENT

This work has been carried out under the framework of INTER-IoT, Research and Innovation action - Horizon 2020 European Project, Grant Agreement n.687283, financed by the EU.

REFERENCES

[1] M. Zhou, G. Fortino, W. Shen, J. Mitsugi, J. Jobin, and R. Bhattacharyya, "Guest Editorial Special Section on Advances and Applications of Internet of Things for Smart Automated Systems," IEEE Trans on Automation Science and Engineering, vol. 13, no. 3, pp. 1225-1229, July 2016.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Computer Networks, vol. 54, no. 15, pp. 2787-2805, 2010

[3] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of things," IEEE Internet Computing, vol. 14, no. 1, pp. 44–51, Jan. 2010.

[4] M. Bal, W. Shen, Q. Hao, and H. Xue, "Collaborative Smart Home Technologies for Senior Independent Living: A Review," Proc. of 2011 15th Int'l Conf. on Computer Supported Cooperative Work in Design (CSCWD 2011), Lausanne, Switzerland, pp. 481-488, June 8-10, 2011.

[5] C. Yang, W. Shen, X. Wang, "Internet of Things in Manufacturing: An Overview," IEEE SMC Magazine, 2016.

[6] F. Zhang, F. Liu, M. Zhou, A. Shami, W. Shen, "An IoT Based Monitoring System Framework for Continuous Casting," IEEE Internet of Things Journal, 2016.

[7] A. Katasonov, et al., "Smart Semantic Middleware for the Internet of Things," ICINCO-ICSO 8, pp. 169-178, 2008.

[8] P. Vlacheas, et al., "Enabling smart cities through a cognitive management framework for the internet of things," Communications Magazine, IEEE, vol. 51, no.6, pp. 102–111, 2013.

[9] A. Bassi, et al., "Enabling things to talk," Springer, 2013.

[10] M. Luck, P. McBurney, C. Preist, "A manifesto for agent technology: Towards next generation computing," Autonomous Agents and Multi-Agent Systems, vol. 9, pp. 203–252, 2004.

[11] M. Pipattanasomporn, F. Hassan, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation", Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES. IEEE, 2009.

[12] E. Oliveira, K. Fischer, and O. Stepankova. "Multi-agent systems: which research for which applications," Robotics and Autonomous Systems vol. 27, no.1, pp. 91-106, 1999.

[13] F. Maturana, W. Shen, and D.H. Norrie, "MetaMorph: An Adaptive Agent-Based Architecture for Intelligent Manufacturing," Int'l Journal of Production Research, vol. 37, no.10, pp. 2159-2174, 1999.

[14] A. Rogers, D. Corkill, and N.R. Jennings, N. R. "Agent technologies for sensor networks," IEEE Intelligent Systems, vol. 24, pp. 13-17, 2009

[15] G. Fortino, A. Guerrieri, and W. Russo, "Agent-oriented smart objects development," in Proc. of IEEE 16th Int'l Conf. on Computer Supported Cooperative Work in Design (CSCWD), 2012, pp. 907–912.

[16] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, "Towards a Development Methodology for Smart Object-Oriented IoT Systems: A Metamodel Approach," Systems, Man, and Cybernetics (SMC), IEEE Int'l Conf. on. IEEE, pp. 1297-1302, 2015.

[17] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, and W. Russo, "An Agent-based Middleware for Cooperating Smart Objects", in Highlights on Practical Applications of Agents and Multi-Agent Systems,

Communications in Comp. and Inform. Science (CCIS), vol. 365, pp. 387-398, Springer, 2013.
[18] The IoT-A Unified Requirements list, http://www.iot-a.eu/public/requirements/copy_of_requirements
[19] B. C. Neuman, "Scale in distributed systems," in Casavant, T. L. and Singhal, M., editors, Readings in Distributed Computing Systems, IEEE CS Press, Los Alamitos, CA, USA, 1994, pages 463–489.
[20] D. Miorandi, et al., "Internet of things: Vision, applications and research challenges," Ad Hoc Networks, vol. 10, no.7, pp. 1497-1516, 2012.
[21] D. Bandyopadhyay, and J. Sen, "Internet of things: Applications and challenges in technology and standardization," Wireless Personal Communications, vol. 58, no. 1, pp. 49-69, 2011.
[22] C. Perera, et al., "Context aware computing for the internet of things: A survey," Communications Surveys & Tutorials, IEEE, vol. 16, no. 1, pp. 414-454, 2014.
[23] F. Zambonelli, "Towards a General Software Engineering Methodology for the Internet of Things," arXiv preprint arXiv:1601.05569, 2016
[24] E. Cortese, et al. "Scalability and performance of jade message transport system," AAMAS Workshop on AgentCities, Bologna, vol. 16, 2002, pp. 28.
[25] S. Galzarano, et al., "Gossiping-based AODV for Wireless Sensor Networks," Systems, Man, and Cybernetics (SMC), 2013 IEEE Int'l Conf. on. IEEE, 2013.
[26] M. Esseghir and N. Bouabdallah, "Node density control for maximizing wireless sensor network lifetime," Int'l Journal of Network Management, vol. 18, no. 2, p. 159170, 2008.
[27] Yoneki, Eiko. "Evolution of ubiquitous computing with sensor networks in urban environments." Ubiquitous computing conference, metapolis and urban life workshop Proc., 2005.
[28] J. Gubbi, et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no.7, pp. 1645-1660, 2013.
[29] F. Kawsar, et al., "Design and implementation of a framework for building distributed smart object systems," The Journal of Supercomputing, vol. 54, no.1, pp. 4-28, 2010.
[30] W. Shen, Q. Hao, H.J. Yoon, D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: An updated review," Advanced engineering INFORMATICS, vol. 20, no.4, pp. 415-431, 2006.
[31] M. Wooldridge, and N. R. Jennings, "Intelligent agents: Theory and practice," Knowledge engineering review, vol. 10, no.2, pp. 115-152, 1995.
[32] S. Russell, and P. Norvig, "Artificial Intelligence: a modern approach," Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.
[33] D. Uckelmann, M. Harrison, and F. Michahelles, "An architectural approach towards the future internet of things," Springer Berlin Heidelberg, 2011.
[34] M. Fisher, L. Dennis, and M. Webster, "Verifying autonomous systems," Communications of the ACM, vol. 56, no.9, pp. 84-93, 2013.
[35] S. Bandyopadhyay, et al., "Role of middleware for internet of things: A study," Int'l Journal of Computer Science and Engineering Survey, vol. 2, no.3, 2011, pp. 94-105.
[36] A. Manzalini, F. Zambonelli, "Towards autonomic and situation-aware communication services: the cascadas vision," in: Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. IEEE Workshop on, IEEE, pp. 383–388, 2006.
[37] T. Leppänen, et al, "Mobile agents-based smart objects for the internet of things," Internet of Things Based on Smart Objects. Springer International Publishing, pp. 29-48, 2014.
[38] D. Slama, F. Puhlmann, J. Morrish, R. Bhatnagar, "Enterprise Internet of Things," available at http://enterprise-Internet of Things.org/book/enterprise-Internet of Things/
[39] T. Collins, "A Methodology for Building the Internet of Things", http://www.iotmethodology.com/.
[40] AIOTI- Report on High-Level Architecture (HLA) http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=11812.
[41] Software & Systems Process Engineering Metamodel™ Specification (SPEM™), Version 2.0, Release Date: April 2008. Available at http://www.omg.org/spec/SPEM/2.0/
[42] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with JADE," Intelligent Agents VII Agent Theories Architectures and Languages. Springer Berlin Heidelberg, pp. 89-103, 2000.
[43] IEEE PROJECT P2413 - Standard for an Architectural Framework for the Internet of Things (IoT)-https://standards.ieee.org/develop/project/2413.html
[44] G. Fortino, et al., "A discovery service for smart objects over an agent-based middleware," Internet and Distributed Computing Systems. Springer Berlin Heidelberg, pp. 281-293, 2013.
[45] G. Fortino, et al., "A flexible building management framework based on wireless sensor and actuator networks," Journal of Network and Computer Applications, vol. 35, no. 6, pp. 1934-1952, 2012.
[46] G. Fortino, et al., "Enabling effective programming and flexible management of efficient body sensor network applications," Human-Machine Systems, IEEE Trans on, vol. 43, no.1, pp. 115-133, 2013.
[47] A. Rehman, A. Abu Zafar, and A. S. Zubair, "Building a smart university using RFID technology," Computer Science and Software Engineering, 2008 Int'l Conf. on. vol. 5, IEEE, pp. 641-644, 2008.
[48] H. I. Wang, "Constructing the Green Campus within the Internet of Things Architecture," Int'l Journal of Distributed Sensor Networks, 2014.
[49] L. Tan, and W. Neng, "Future internet: The internet of things," Advanced Computer Theory and Engineering (ICACTE), 2010 3rd Int'l Conf. on. vol. 5. IEEE, 2010, pp. V5-376.
[50] M. Cata, "Smart university, a new concept in the Internet of Things," RoEduNet Int'l Conf.-Networking in Education and Research (RoEduNet NER), 2015 14th. IEEE, pp. 195-197, 2015.
[51] T. G. Stavropoulos, et al. "System architecture for a smart university building," Artificial Neural Networks–ICANN 2010. Springer Berlin Heidelberg, pp. 477-482, 2010.
[52] J. Bohli, P. Langendorfer, and A. F. Skarmeta, "Security and privacy challenge in data aggregation for the iot in smart cities," Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, pp. 225-244, 2013.
[53] M. Nati, et al., "Smartcampus: A user-centric testbed for internet of things experimentation," Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on. IEEE, pp. 1-6, 2013.
[54] A. Sabato, M.Q. Feng, Y. Fukuda, D.L. Carni, G. Fortino, "A Novel Wireless Accelerometer Board for Measuring Low-Frequency and Low-Amplitude Structural Vibration," IEEE Sensors Journal, vol. 16, no. 9, pp. 2942-2949, 2016.
[55] G. Fortino, et al., "Integration of agent-based and cloud computing for the smart objects-oriented iot," Computer Supported Cooperative Work in Design (CSCWD), Proc. of the 2014 IEEE 18th Int'l Conf. on. IEEE, 2014.
[56] G. Fortino, and W. Russo, "ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems," Information and Software Technology, vol. 54, no.6, pp. 608-624, 2012.
[57] G. Fortino, W. Russo, and C. Savaglio, "Agent-oriented Modelling and Simulation of IoT Networks," Proc. of the 10th International Workshop on "Multi-Agent Systems and Simulation" (MAS&S'16), Gdansk, Poland, 11-14 September, 2016.
[58] L. Pan, Z. Ding and M. C. Zhou, "A Configurable State Class Method for Temporal Analysis of Time Petri Nets," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 44, no. 4, pp. 482-493, April 2014.
[59] W. Liu, Y. Du, M. C. Zhou, and C. Yan, "Transformation of Logical Workflow Nets," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 44, no. 10, pp. 1401-1412, Oct. 2014.
[60] J. Luo, H. Ni and M. C. Zhou "Control Program Design for Automated Guided Vehicle Systems via Petri Nets," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 45, no. 1, pp. 44-55, Jan. 2015.
[61] C. Liu, Q. Zeng, H. Duan, M. C. Zhou, F. Lu and J. Cheng, "E-Net Modelling and Analysis of Emergency Response Processes Constrained by Resources and Uncertain Durations," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 45, no. 1, pp. 84-96, Jan. 2015.
[62] Z. Ding, Y. Zhou, M. Jiang, and M. C. Zhou "A New Class of Petri Nets for Modelling and Property Verification of Switched Stochastic Systems," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 45, no. 7, pp. 1087–1100, July 2015.
[63] Z. Ding, Y. Zhou and M. C. Zhou, "Modelling Self-Adaptive Software Systems with Learning Petri Nets," IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 46, no. 4, pp. 483-498, April 2016.
[64] P. Yang, D. Stankevicius, V. Marozas, Z. Deng, E. Liu, A. Lukosevicius, F. Dong, L. Xu, and G. Min, "Lifelogging Data

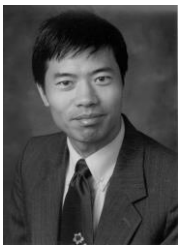
Validation Model for Internet of Things Enabled Personalized Healthcare," IEEE Trans on Systems, Man, and Cybernetics, vol. PP, issue 99, July 2016, pp. 1-15, July 2016.
[65] J. Vicente, and V. Botti, "Developing real-time multi-agent systems," Integrated Computer-Aided Engineering, vol. 11, no. 2, pp. 135-149, 2004.
[66] I. Ayala, Amor M., and L. Fuentes, "The Sol agent platform: Enabling group communication and interoperability of self-configuring agents in the Internet of Things," Journal of Ambient Intelligence and Smart Environments, vol. 7, no. 2, pp. 243-269, 2015.
[67] L. Braubach, A. Pokahr, and W. Lamersdorf, "Jadex: A short overview," Main Conference Net. ObjectDays, vol. 2004, pp. 195-207, 2004.
[68] F. Cicirelli, G. Fortino, A. Guerrieri, G. Spezzano, and A. Vinci, "Metamodelling of Smart Environments: from design to implementation," Advanced Engineering Informatics, 2016.
[69] C. Savaglio, and G. Fortino, "Autonomic and Cognitive Architectures for the Internet of Things," Int'l Conf. on Internet and Distributed Computing Systems. Springer, Cham, 2015.
[70] G. Fortino, A. Rovella, W. Russo, and C. Savaglio, "Towards Cyberphysical Digital Libraries: Integrating IoT Smart Objects into Digital Libraries,". in Management of Cyber Physical Objects in the Future Internet of Things, pp. 135-156, Springer International Publishing, 2016.
[71] T.S. López, et al., "Adding sense to the Internet of Things," Personal and Ubiquitous Computing, vol. 16, no .3, pp. 291-308, 2012.
[72] Cai, Hongming, et al., "IoT-based big data storage systems in cloud computing: Perspectives and challenges," IEEE Internet of Things Journal, vol. 4, no. 1, pp. 75-87, 2017.
[73] Cai, Hongming, et al., IoT-based configurable information service platform for product lifecycle management. IEEE Trans on Industrial Informatics, vol. 10, no. 2, pp. 1558-1567, 2014.
[74] Liu, Yang, et al., "Review on cyber-physical systems," IEEE/CAA Journal of Automatica Sinica, vol. 4, no. 1, pp. 27-40, 2017.

Giancarlo Fortino (M'01-SM'12) is a Professor of Computer Engineering at the Dept. of Informatics, Modeling, Electronics and Systems (DIMES) of the University of Calabria (Unical), Italy. He has a Ph. D. degree and Laurea (MSc+BSc) degree in Computer Engineering from Unical. He holds the Italian Scientific National Habilitation for Full Professorship and is High-end Foreign Expert of China, Adjunct Professor at the Wuhan University of Technology (China). His main research interests include agent-based computing, Internet of Things, body area networks, wireless sensor networks, pervasive and cloud computing. He is currently the deputy coordinator and STPM of the EU-funded H2020 INTER-IoT project. He authored about 350 publications in journals, conferences and books. He is the founding editor of the Springer Book Series on "Internet of Things", and currently serves (as associate editor) in the editorial board of IEEE T. on Affective Computing, IEEE T. on Human-Machine Systems, IEEE Sensors Journal, IEEE Access, JNCA, EAAI, Information Fusion. He is the recipient of the 2014 Andrew P. Sage SMC Trans Paper award. He is the Chair of the IEEE SMC Italian Chapter.

Wilma Russo received her Laurea (BSc+MSc) degree in Physics from University of Naples, Italy. She is a Full Professor of Computer Engineering at the Dept. of Informatics, Modeling, Electronics and Systems (DIMES) of the University of Calabria (Unical), Italy. Her research interests include distributed computing and systems, software agents, multimedia networks, and the Internet of Things. She authored over 100 papers in journals, conferences and books.

Claudio Savaglio (SM'16) received his B.S. and M.S. degrees in Computer Engineering in 2010 and 2013 from the University of Calabria, Italy, where he is currently pursuing the Ph.D. degree in ICT. In 2013 he was Visiting Researcher at University of Texas at Dallas (TX, U.S.A.), in 2016 at New Jersey Institute of Technology, (NJ, U.S.A.), and in 2017 at Universitat Politècnica de València (Valencia, Spain). His research interests include the Internet of Things, network simulation, and agent-oriented middleware and development methodologies.

Weiming Shen (F'13) is a Senior Research Scientist at the National Research Council Canada and an Adjunct Professor at Tongji University, China, and University of Western Ontario, Canada. He received his Bachelor and Master's degrees from Northern (Beijing) Jiaotong University, China and his PhD degree from the University of Technology of Compiegne, France. His recent research interest includes agent-based collaboration technology and applications, Internet of Things, and Big Data Analytics. He has published several books and over 450 papers in scientific journals and conferences in the related areas. His work has been cited over 10,000 times with an h-index of 47. He has been invited to provide over 80 invited lectures/seminars at different academic and research institutions over the world and keynote presentations/tutorials at various Int'l conferences.



MengChu Zhou (S'88-M'90-SM'93-F'03) received his B.S. degree in Control Engineering from Nanjing University of Science and Technology, Nanjing, China in 1983, M.S. degree in Automatic Control from Beijing Institute of Technology, Beijing, China in 1986, and Ph. D. degree in Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY in 1990. He joined New Jersey Institute of Technology (NJIT), Newark, NJ in 1990, and is now a Distinguished Professor of Electrical and Computer Engineering. His research interests are in Petri nets, intelligent automation, Internet of Things, big data, web services, and intelligent transportation. He has over 700 publications including 12 books, 390+ journal papers (over 280 in IEEE Trans), 11 patents and 28 book-chapters. He is the founding Editor of IEEE Press Book Series on Systems Science and Engineering. He is a recipient of the Norbert Wiener Award from IEEE Systems, Man and Cybernetics Society.