An Agent-Based Middleware for Cooperating Smart Objects

Giancarlo Fortino, Antonio Guerrieri, Michelangelo Lacopo, Matteo Lucia, and Wilma Russo

DEIS, University of Calabria

Abstract. This paper proposes an agent-oriented and event-based framework for the development of cooperating smart objects. Smart objects are objects of the real life augmented with computing, communication, sensing/actuation and storing functionalities. They are the building blocks of the future Internet of Things (IoT) towards the construction of complex smart environments. In the proposed framework, smart objects are modelled as agents that can cooperate as a multi-agent system to fulfill specific goals. The framework implementation relies on the JADE middleware that provides an effective agent management and communication infrastructure. In particular, cooperating smart objects can be implemented as JADE or Jadex agents and can cooperate through direct coordination based on ACL message passing and spatio-temporal decoupled coordination based on a topic-based publish/subscribe. A simple yet effective case study referring to a smart office environment constituted by two cooperating smart objects, is presented to elucidate the proposed approach.

Keywords: Internet of Things, Smart Objects, Multi-Agent Systems, Wireless Sensor and Actuator Networks, JADE.

1 Introduction

Recent progresses in electronics, telecommunications and computing are driving the vision of the Internet of Things (IoT), a world-wide network of interconnected heterogeneous physical objects (sensors, actuators, smart devices, smart objects, RFID, embedded computers, etc) uniquely addressable and based on standard communication protocols [1].

Among several approaches available for building the IoT [2], in this paper we focus on an IoT defined as a loosely coupled, decentralized system of cooperating smart objects (SOs). An SO is an autonomous, physical digital object augmented with sensing/actuating, processing, storing, and networking capabilities. It is able to sense/actuate, store, and interpret information created within itself and around the neighboring external world where it is situated, acts on its own, cooperates with other SOs, and exchanges information with other kinds of electronic devices and human users.

To date, research efforts have mainly focused on prototyping middleware infrastructures for the implementation of SO-based smart systems. Apart from

J.M. Corchado et al. (Eds.): PAAMS 2013 Workshops, CCIS 365, pp. 387–398, 2013.

many projects focused on smart environments but not specifically on smart objects, UbiComp [3], FeDNet [4] and Smart Products [5] promote SOs as central entities in developing the IoT infrastructure, even though they differ in many aspects (e.g. programming model, metadata, system architecture, SO architecture, communication model, proactivity, programming language, knowledge management).

To define an IoT infrastructure based on a well-defined distributed computing paradigm which effectively supports the definition of distributed computing entities, their architecture and their coordination, this paper proposes an agent-oriented and event-based framework for the development of cooperating smart objects. The framework relies on the agent paradigm, which is centered on the concept of agent, as it is a well-defined distributed computing paradigm for developing methods and middleware for SOs. Agents are defined as autonomus, proactive, social, and situated entities that can fulfill specific objectives [6]. Therefore, the characteristics of agents perfectly fit those of the SOs. The proposed framework is implemented in JADE [7] and allows to program cooperating smart objects as JADE or Jadex [8] agents that comply with a well-defined event-driven reference architecture. The exploitation of JADE will enable interoperability between SO applications and agent applications based on JADE. Moreover, the proposed smart objects are based on the BMF [9] and SPINE [10] frameworks, which manage their sensor and actuator networks that are based on IoT standards (e.g. IEEE 802.15.4, ZigBee, 6LowPan). The JADE-based framework is finally exemplified through a simple yet effective case study.

The remainder of the paper is organized as follows. Section 2 discusses related work. In Section 3 the event-driven reference architecture for cooperating smart objects is described. Section 4 presents the proposed agent-based framework whereas the case study is detailed in Section 5. Finally conclusions are drawn and future work is briefly discussed.

2 Related Work

Nowadays, the development of architectures and middlewares for SOs is still an emergent research activity. Available works can be roughly classified in three kinds: (i) ad-hoc middlewares fo smart environments that could be reused, after a proper enhancement, for smart objects (e.g. Smart-Its, 2WEAR, Ambient Agoras, Aura, Gaia, iRoom) [11]; (ii) infrastructures focused on an all-inclusive IoT vision, where each object, even a sensor or an RFID, belongs to the IoT [12], [13]; (iii) smart object middleware focused on the development of an SO-based IoT (e.g. FedNet [4], SmartProducts [5], and UbiComp/Gadgetware Architectural Style [3]). In particular, we discuss the latter works as they are specifically focused on SO middleware.

FedNet [4] uses XML metadata to describe the requirements of SO applications, but it does not consider the SO management. FedNet does not provide a SO architecture because it is a high level middleware providing an interface to different SO architectures. For this reason, the proactivity in FedNet is “out

of the SO" and applications are able to provide proactivity by orchestrating the SOs. The matching between FedNet application requirements and services provided by SOs together with the proactivity is managed by a (centralized) coordinator. FedNet supports 802.11x (TCP/IP) and bluetooth (RF-COMM) communication protocols.

SmartProducts [5] provides a metadata representation based on OWL and RDF languages. SmartProducts offers an SO architecture which allows the SOs to cooperate with each other in a P2P fashion through the communication middleware named MundoCore that supports several low-level communication protocols. SmartProducts supports proactivity in SOs, which can store knowledge in a proactive knowledge base (through specific APIs) which is associated with a reasoner to gather new knowledge.

UbiComp/GAS [3] uses XML data for the representation and communication of the SOs and their communication. It is based on a middleware named GAS-OS, which is installed on each SO and defines the SO architecture. In UbiComp/GAS, SOs are components of distributed applications collaborating in a P2P fashion. It is based on the plug/synapse model for interconnecting SOs and the proactivity is limited to the substitution of lost synapses. The knowledge management is based on Knowledge Bases (KBs) and on a Prolog inference engine that supports the gathering of new knowledge. The communication protocols provided are TCP/IP and eRDP.

The aim of our proposal is the design and implementation of an event-driven SO architecture suitable for every SO and a distributed high-level P2P framework for SOs based on the agent paradigm. Our framework supports several communication types (message passing and publish/subscribe) and provides proactivity based on inference rules and on local and remote KBs.

3 Event-Driven Architecture for Cooperating Smart Objects

A Smart Object (SO) is a common physical object augmented with sensing, actuation, processing, storing, and communication capabilities. To implement these capabilities, hardware and software components have to be added to its physical structure. In particular, the hardware components provide the objects with augmented capabilities and the software components implement the SO functionalities.

The hardware structure of a SO is typically composed by a computing device (such as a PC/notebook/tablet/smartphone or even an embedded computing node) and a set of wireless/wired sensors and/or actuators nodes. The device computing power is purposely defined depending on the functionalities the SO will provide and on the dimension and complexity of the SO.

The software infrastructure can be logically organized according to a master/slave model, in which the master (or coordinator) is the most powerful device of the SO and can control a set of software entities running on sensor/actuator nodes. The coordinator is the only component having the capability of communicating with other SOs and other external, personal and environmental devices.

The communication capability with other SOs provides the basis for cooperation among SOs to achieve common goals, e.g. data sharing, complex service provisioning, ambient intelligence management, etc.

This simple yet effective SO model is quite general as it can accommodate any size of SOs: small (e.g. pencil, desk, sofa, coffeemaker, door), medium (e.g. motorbike, tram, bus shelter) and large (e.g. building, tunnel, highway). Moreover, in case of large SOs, the coordinator can be hierarchically organized to optimize the SO management functionalities.

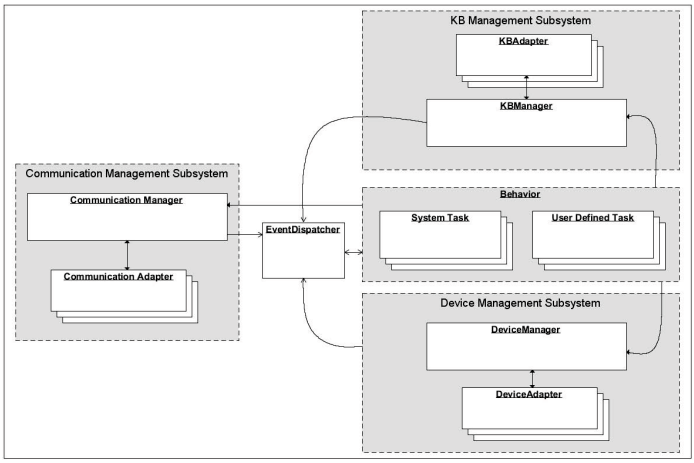

Fig. 1. SO Architecture

The defined cooperating smart objects (CSO) comply with the event-driven architecture (see Fig. 1) which is an instance of the high level architecture reported in [11]. The proposed architecture is composed of a *Behavior* that formalizes the object behavior, an *EventDispatcher* that manages all the internal events of the object, a *Communication Management Subsystem* that manages communications with other CSOs and external entities, a *Device Management Subsystem* that manages the sensor/actuator nodes of the object, and a *KB Management Subsystem* that manages the object knowledge base.

The Behavior component is composed by a set of Tasks. Tasks are software subcomponents programmed to reach specific objectives through a set of operations, involving computation, communication, sensing/actuation, and storage management. They can be either proactive or reactive. Proactive tasks are able to self-trigger to fulfill specific objectives whereas reactive tasks are only triggered by events sent by other internal or external entities. Tasks have also an internal state and can interact with the CSO subsystems and with other tasks.

According to the proposed architecture, as tasks are driven by events, external CSO communication, signals to/from the CSO devices, data to/from the KB

are always formalized as events and handled by the EventDispatcher that sends them to the interested tasks. In particular, when the EventDispatcher starts its execution, waits for events. When an event arrives, it is inserted into the event queue of the EventDispatcher, which fetches, filters and, if not discarded, forwards the event only to the interested tasks. More than a task can be target of the same event instance.

Depending on the realization of the architecture, tasks can be implemented following either run-to-completion or multi-threading paradigms.

Tasks can be divided in:

- **System Tasks**: they provide basic services common to all the CSOs. In particular, the system tasks are:
  - *Shutdown/Reboot/Standby* tasks, which respectively implement the shutdown/reboot/standby operations of the CSO.
  - *Discovery* task, which enables the CSOs discovery.
  - *Information Access* task, which provides the information related to the basic CSO functionalities.
  - *Parameter Setting* task, which allows setting the basic parameters of the CSO.
- **User Defined Tasks**: they are application-level tasks designed to define specific CSO behaviors. Examples of User Defined tasks are provided in Section 5.

Events are characterized by two properties: event type and event source type. The types of event can be:

- **Inform**: events containing information;
- **Request**: events formalizing a request;
- **Log**: events for logging purposes;
- **Error**: events representing occurring errors.

The event source types can be:

- **Internal**: the event source is an internal software component.
- **External**: the event source is an external entities or components.
- **Device**: the event source is a CSO device.

A priority among events is defined as follows. Error events have the highest priority whereas the Log events have the lowest one. Inform and Request events have the same priority, which can also be specifically customized.

Device Management Subsystem, Communication Management Subsystem, and KB Management Subsystem are designed to be generic and are respectively based on extensible DeviceAdapters, CommunicationAdapters, and KBAdapters to allow for interaction with different entities through different protocols.

The Device Management Subsystem manages interactions with sensing/actuation devices and is composed by:

- *DeviceManager*, which manages and coordinates different DeviceAdapters.
- *DeviceAdapter*, which allows to interact with sensors/actuators hiding low level-details. In particular, it interprets high-level requests from tasks which translates into the specific sensor/actuator protocol, and receives data from sensors/actuators which makes available to tasks.

The Communication Management Subsystem provides a common interface for different kinds of communication (local or remote) with other CSOs or different devices so as to allow the CSO to manage all the communication in the same way. This subsystem is composed by:

- *CommunicationManager*, which manages and coordinates different CommunicationAdapters.
- *CommunicationAdapter*, which manages all the active connections and monitors the channel to set new connections by hiding low-level mechanisms;

The KB Subsystem provides the CSO with a knowledge base and consists of:

- *KBManager*, which manages and coordinates different KBAdapters.
- *KBAdapter*, which manages a KB containing the knowledge of the CSO. The KB can be local or remote and store information that can be shared among tasks.

4 Agent-Based Implementation

The event-driven architecture for CSOs has been fully implemented and integrated in the JADE middleware (see Fig. 2). CSOs are thus JADE-based agents so exploiting all features of JADE middleware at the agent management and communication levels. In the following subsections we first provide some basic information about the JADE middleware and then describe the JADE-based implementation of CSOs.

4.1 The JADE Middleware

JADE [7] is a FIPA-compliant middleware for the development of distributed multi-agent systems. A JADE agent is defined as a set of behaviors, each of which represents one or more tasks to fulfill. JADE doesn't provide any high-level abstraction for the definition of intelligent behaviors. To this purpose, the Jadex framework [8] has been introduced. It allows to program agents according to the BDI (Belief Desire Intention) paradigm. Specifically, a Jadex agent is defined as a triple: *Goal* (the agent objectives), *Belief* (the agent beliefs), and *Plan* (the agent plan). Jadex provides an execution model based on events that trigger the execution of plans. It is possible to execute Jadex agents on the JADE platform by using the specific adapter.

The agent communication is managed by the JADE platform through ACL message passing according to the FIPA specifications. The agent interaction can

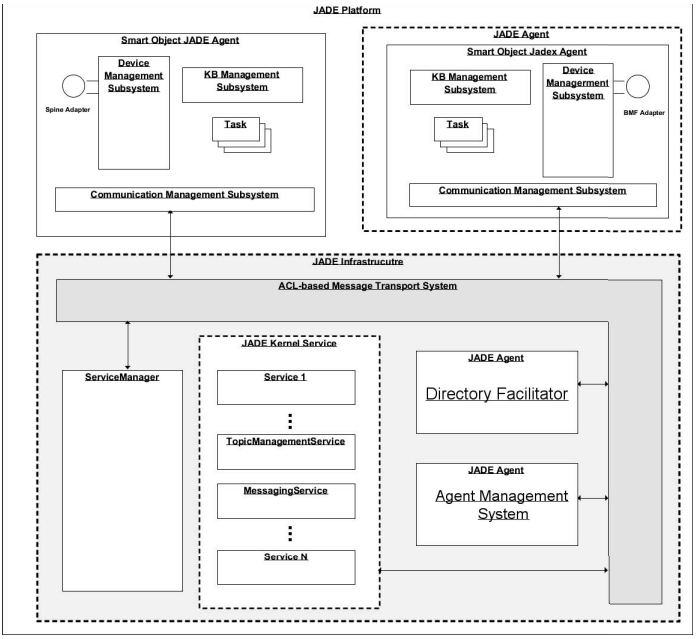

Fig. 2. Realization of the event-driven architecture for CSOs using the JADE middleware

also occur by means of the publish/subscribe pattern based on the topic mechanism. Such mechanism allows to send a given message to many agents without knowing the identity of the target agents. The topic-based communication is implemented in the JADE kernel service, named *TopicManagementService*, which manages the creation and the subscription to topics.

The exploitation of ACL messages and the topic-based coordination allows for interoperability among CSOs that will be able to request services and exchange information with each other and with other FIPA agents.

4.2 Agent-Oriented CSOs

The JADE-based CSO architecture is reported in Fig. 2. CSOs are agents of the JADE platform so they are managed by the AMS (Agent Management System) and can use the DF (Directory Facilitator) to look up other agents. The communication layer is based both on ACL messages and topic-based Publish/Subscribe. In the following we provide the most relevant JADE-based implementation details of the CSO architecture components (see Fig. 1).

– **Task**. Due to the affinity between the Task concept and the concepts of Behaviour (in JADE) and Plan (in Jadex), tasks are defined as JADE Behaviours or Jadex Plans. Thus, the task execution is based on the mechanisms provided by the specific framework.

- **EventDispatcher**. The EventDispatcher (ED) is modelled as an active component (Behaviour or Plan) and operates according to the execution mechanisms of the exploited platforms. For each event submitted to the ED, it adds such an event to its queue and self triggers the event dispatching through an ACL message in JADE or through an event in Jadex. In JADE, each task implemented as a Behaviour will wait for a specific ACL message that, in turn, contains the high-level event to be dispatched to the task. A task will use the *register* method provided by the ED to register to the events of interest with its ID and, by using the JADE message template, will intercept the ACL messages having a conversationID equal to its ID. Thus, for each registered event, the ED builds an ACL message, sets the apposite conversationID and sends the message into the internal system. Jadex is based on events so allowing to distinguish between internal events and input/output messages. As in Jadex a plan can be executed upon the occurrence of an event of interest, an event is uniquely associated to a task (the association could be based either on XML or Java classes). In particular, tasks register the high-level event of interest and the triggering Jadex event to the ED.
- **Communication Management Subsystem**. As shown in Fig. 2, JADE provides a set of services (TopicManagementService and MessagingService) and an ACL-based communication channel for the agent iteraction. To provide communication among CSOs, an active component, named CommunicationManagerMessageHandler has been introduced (as Behaviour in JADE and as Plan in Jadex), which captures the ACL messages targeting CSOs and translates them into external events (see Section 3). Moreover, two other handlers (TCPAdapter and UDPAdapter) have been defined to manage communication with external networked entities based on TCP and UDP.
- **Device Management Subsystem**. The management of wireless sensors/actuators is carried out through the DeviceManager that handles several DeviceAdapters. In particular, two DeviceAdapters have been realized: the BMFAdapter, which allows to manage wireless sensor and actuator networks (WSANs) based on the BMF framework [9], and the SPINEAdapter, which allows to manage Body Sensor Networks (BSNs) based on the SPINE framework [10]. BMF and SPINE are based on IoT standards protocols such as IEEE 802.15.4, ZigBee, and 6LowPan.
- **Knowledge Management Subsystem**. Currently, the KB is constituted by just one object containing the global state variables of the CSO; ocal variables can also be kept inside the CSO tasks.

5 Designing a Smart Environment through Cooperating Smart Objects

In this section we exploit the agent-oriented framework described in the previous section to develop a case study. In particular, the case study refers to a smart environment composed of two CSOs: a Smart Office (or SmartO) and a Smart Body (or SmartB). The two CSOs will gather information and cooperate to

support the working activity of the office user. As shown in Fig. 3: (i) the SmartO is physically composed of an office room with two desks, two PCs with screen, a whiteboard, a projector, and a chair, and is augmented with a set of wireless sensors, organized as a BMF-based WSAN; (ii) the SmartB corresponds to the office user that wears a BSN, which consists of two accelerometer-equipped sensor nodes and a mobile basestation, which is able to recognize the following user activities: standing still, sitting, walking, and laying down.

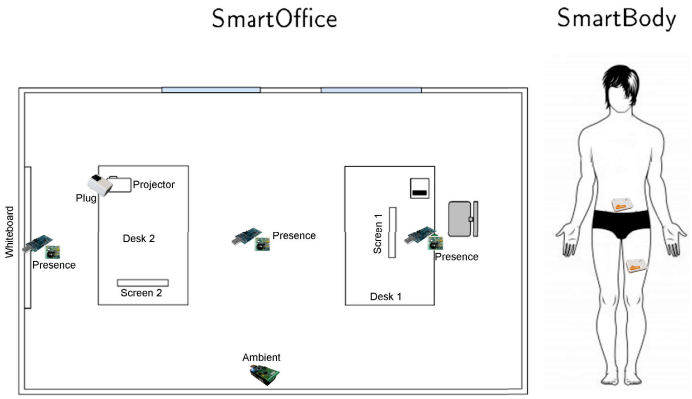

Fig. 3. Physical and hardware structure of the CSOs: Smart Office and Smart Body

5.1 Operating Scenarios

The operating scenarios of the case study refer to a usual working day: entry in the office, work at desk, work at whiteboard, meeting, etc. On the basis of the information gathered, the two CSOs will support the user during the working activity by suggesting to turn the lights and/or the projector off while not used, to take a break, and showing such information on the screen closest to the user. In Table 1 the defined scenarios are described in detail by reporting the correlated inference rules and actions performed by the smart environment.

5.2 Interaction between SmartO and SmartB

Each CSO publishes a set of topics and services that can be exploited by each other or by other entities to implement more complex services. In particular, in Table 2 the topics that the two CSOs publish and subscribe to are reported. Moreover, the provided services can be requested one-shot or according to specific state transitions of the CSO. In particular, the services offered by SmartB allow to query SmartB about the activity the user is currently performing, whereas SmartO, apart from the services for querying the room state (e.g. user presence, temperature, light, etc), provides actuation services that allow the exploitation of the screens.

Table 1. Operating scenarios of the case study

Scenario	Description	Inference	Action
1	User enters in the office	Uncertain position	Information shown on both the screens
2	User at desk 1	Work at desk	Information shown only on Screen 1
3	User is sitting for too long time	User should stand up	Alerting message displayed on Screen 1
4	User moves around the room	Uncertain position, walking	Information shown on both the screens
5	User uses the whiteboard	Work at whiteboard, standing	Information shown only on Screen 2
6	User starts a presentation	Presentation running	Switch screens off (do not disturb), notify information through Twitter
7	Presentation over, user forgets the projector on and sits to Desk 1	Energy wastage	Alerting message sent to the user on Screen 1
8	User leaves forgetting lights on	Energy wastage	Alerting message sent to the user through Twitter

Table 2. Published and subscribed topics

Topic	Publisher	Subscriber
Sitting	SmartB	SmartO
PresentationIsRunning	SmartO	SmartB
Work_at_Desk	SmartO	SmartB
Work_at_Whiteboard	SmartO	SmartB
MorePeopleInTheRoom	SmartO	SmartB

5.3 An Overview of SmartB and SmartO

SmartB has been developed by using the JADE version of the proposed agent framework that can work atop J2SE and J2ME (or Android). The management of the BSN (constituted by two Shimmer nodes, see Fig. 3) is based on SPINE which is integrated through the SPINEAdapter. To display feedback messages to the user, SmartB uses the screen service of SmartO. The KB of SmartB is distributed among its tasks. Some inference rules embedded in SmartB are as follows:

1. $TooLongSitting \Leftarrow SittingTime > 2h \wedge WorkAtDesk$
2. $DoingPresentation \Leftarrow Walking \wedge PresentationIsRunning$
3. $UseScreen1 \Leftarrow DeskUsed \wedge Sitting \wedge \neg MorePeopleInTheOffice$
4. $UseScreen2 \Leftarrow WhiteboardUsed \wedge \neg MorePeopleInTheOffice$

SmartO has been developed by using the Jadex version of the proposed agent framework that can work atop J2SE. The management of the WSAN (constituted by five sensor nodes: three TelosB equipped with presence sensors, one TelosB for ambient sensing, and an Epic smart plug node to measure the power consumption of the projector, see Fig. 3) is based on BMF which is integrated through the BMFAdapter. Moreover, SmartO directly controls the two screens for providing feedback to the user through a specific GUI. The KB consists of a Java class which maintains all the state variables and generates events when the values of such variables change. Some inference rules embedded in SmartO are as follows:

1. $isSomebodyInTheRoom \Leftarrow AmbientPresence \vee isDeskUsed \vee isWhiteboardUsed$
2. $isPresentationRunning \Leftarrow lowAmbientLight \wedge isSomebodyInTheRoom \wedge isProjectorOn$
3. $notDisturb \Leftarrow isPresentationRunning$
4. $morePeopleInTheOffice \Leftarrow isPresentationRunning \vee (isDeskUsed \wedge isWhiteBoardUsed)$
5. $uncertainPosition \Leftarrow morePeopleInTheOffice \vee (isSomebodyInTheRoom \wedge \neg isWhiteboardUsed \wedge \neg isDeskUsed)$
6. $waste \Leftarrow isAmbientLightHigh \wedge (isProjectorOn \vee \neg isSomebodyInTheRoom)$
7. $useScreenOne \Leftarrow isDeskUsed \wedge \neg notDisturb$
8. $useScreenTwo \Leftarrow isWhiteboardUsed \wedge \neg notDisturb$

6 Conclusion

This paper has proposed an agent-oriented framework for the development of cooperating smart objects as building blocks for the constitution of even complex smart environments towards the future IoT. An ecosystem of cooperating smart objects is modelled and implemented as a distributed MAS based on the widely used JADE middleware. Finally, a case study has shown the effectiveness of using the proposed approach in developing smart environments based on smart objects. Moreover, apart from the well-recognized benefits to exploit an agent-oriented approach, the exploitation of JADE can facilitate integration with other FIPA-compliant agent systems.

On-going work is being devoted to define tiny cooperating smart objects based on the MAPS (Mobile Agent Platform for SunSPOTs) framework [14], integrate them on the basis of the JADE-MAPS gateway [15], and extend the JADE DF with effective CSO discovery mechanisms. Future work will focus on the customization of the agent-oriented methodology ELDAMeth [16],[17] for the development of smart environments based on the proposed agent-oriented framework.

Acknowledgments. This work has been partially supported by TETRis - TETRA Innovative Open Source Services, funded by the Italian Government (PON 01-00451).

References

1. Vasseur, J.P., Dunkels, A.: Interconnecting Smart Objects with IP - The Next Internet. Morgan Kaufmann (2010)
2. Kortuem, G., Kawsar, F., Sundramoorthy, V., Fitton, D.: Smart Objects as Building Blocks for the Internet of Things. IEEE Internet Computing 14(1), 44–51 (2010)
3. Goumopoulos, C., Kameas, A.: Smart Objects as Components of UbiComp Applications. International Journal of Multimedia and Ubiquitous Engineering 4 (2009)
4. Kawsar, F., Nakajima, T.: A Document Centric Framework for Building Distributed Smart Object Systems. In: Proc. of the 2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2009, pp. 71–79. IEEE Computer Society (2009)
5. Miche, M., Schreiber, D., Hartmann, M.: Core Services for Smart Products. In: Smart Products: Building Blocks of Ambient Intelligence (AmI-Blocks 2009), collocated with AmI 2009 (2009)

6. Luck, M., McBurney, P., Preist, C.: A Manifesto for Agent Technology: Towards Next Generation Computing. Autonomous Agents and Multi-Agent Systems 9(3), 203–252 (2004)
7. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Softw. Pract. Exper. 31, 103–128 (2001)
8. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI Reasoning Engine. In: Multi-Agent Programming, pp. 149–174 (2005)
9. Fortino, G., Guerrieri, A., O'Hare, G., Ruzzelli, A.: A flexible building management framework based on wireless sensor and actuator networks. Journal of Network and Computer Applications 35(6), 1934–1952 (2012)
10. Bellifemine, F., Fortino, G., Giannantonio, R., Gravina, R., Guerrieri, A., Sgroi, M.: SPINE: A domain-specific framework for rapid prototyping of WBSN applications. Software - Practice and Experience 41(3), 237–265 (2011)
11. Fortino, G., Guerrieri, A., Russo, W.: Agent-oriented smart objects development. In: Proc. of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012, pp. 907–912 (2012)
12. de Souza, L.M.S., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., Savio, D.: SOCRADES: A web service based shop floor integration infrastructure. In: Floerkemeier, C., Langheinrich, M., Fleisch, E., Mattern, F., Sarma, S.E. (eds.) IOT 2008. LNCS, vol. 4952, pp. 50–67. Springer, Heidelberg (2008)
13. Floerkemeier, C., Lampe, M., Roduner, C.: Facilitating RFID Development with the Accada Prototyping Platform. In: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOMW 2007, pp. 495–500. IEEE Computer Society, Washington, DC (2007)
14. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A java-based agent platform for programming wireless sensor networks. Computer Journal 54(3), 439–454 (2011)
15. Mesjasz, M., Cimadoro, D., Galzarano, S., Ganzha, M., Fortino, G., Paprzycki, M.: Integrating JADE and MAPS for the development of agent-based WSN applications. In: Fortino, G., Badica, C., Malgeri, M., Unland, R. (eds.) Intelligent Distributed Computing VI. SCI, vol. 446, pp. 211–220. Springer, Heidelberg (2012)
16. Fortino, G., Garro, A., Russo, W.: A discrete-event simulation framework for the validation of agent-based and multi-agent systems. In: Proceedings of WOA 2005 - 6th AI*IA/TABOO Joint Workshop "From Objects to Agents": Simulation and Formal Analysis of Complex Systems, pp. 75–84 (2005)
17. Fortino, G., Russo, W.: ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems. Information and Software Technology 54(6), 608–624 (2012)