2015 IEEE International Conference on Systems, Man, and Cybernetics

Towards a Development Methodology for Smart Object-Oriented IoT Systems: a Metamodel Approach

Giancarlo Fortino, Antonio Guerrieri, Wilma Russo, Claudio Savaglio
DIMES - University of Calabria
Via P. Bucci, cubo 41C, 87036 Rende (CS), Italy
Email: g.fortino@unical.it, aguerrieri@deis.unical.it, w.russo@unical.it, csavaglio@dimes.unical.it

***Abstract*—The Internet of Things (IoT) is a large-scale complex networked cyberphysical system in which the Smart Objects (SOs) will be the fundamental building blocks. Although, many research efforts in the IoT realm have been to date devoted to device, networking and application service perspectives, software engineering approaches for the development of IoT systems are still in their infancy. This paper introduces a novel software engineering approach aiming to support a systematic development of SOs-based systems. The proposed approach is based on metamodels that are defined at different levels of abstraction to support the development phases of analysis, design and implementation. The effectiveness of the proposed approach is demonstrated through a simple yet effective case study, showing the development of a smart office SO from the high-level design to its agent-based implementation.**

Keywords*—*Internet of Things; Smart Objects; Multi Agent Systems; Middleware; Agent Oriented Software Engineering.

I. Introduction

The Internet of Things (IoT) vision refers to a scenario in which daily real world objects participate to the Internet, being able to be globally networked, discovered and exploited, with consequent benefits both for individuals and businesses [1]. The concept of Smart Object (SO), which underlies a physical object strengthened in its functions by processing, communication, sensing and actuation units and augmented by an embedded smartness, is quite widespread and well-established. Due to these important features, SOs play an important role in the context of cyber-physical systems and they are the fundamental building blocks [2] of the Internet and/or Web of Things [3]. To full exploit the widely recognized SO's potential in analyzing, designing and implementing IoT complex eco-systems (in which human users and different devices interact), well-defined development methodologies are required.

To address this issue and specifically to model the IoT system properties at different levels of abstraction, the metamodel approach can be exploited [4]. To date, in literature several approaches for the development of IoT systems [5], [6] have been proposed. In [6], metamodels are exploited in the analysis phase to abstract the main entities of the IoT (namely the human/digital Users and the Smart Objects), their basic features and high-level interactions. The proposed approach in [5], instead, is focused on the rapid prototyping of smart spaces and proposes the Resource-Oriented and Ontology-Driven Development (ROOD) methodology, which involves metamodels related to the Smart Object Model (SOM) and the Environment Context Model (ECM) for describing high-level behaviors, interactions and context information of the entire smart space. Nevertheless, no well-formalized methodology able to support analysis, design and implementation phases of SO systems development is currently available. In fact, in literature it is possible to recognise operational [6]–[8] and non-operational [2], [9] SO metamodels which differently address such issue, although they do not support a systematic comprehensive approach. However, several multi-agent frameworks [10] exist to support the implementation of IoT systems [11]–[14].

In this paper, we propose a metamodel-based engineering approach for the systematic development of SOs, from analysis to implementation. In particular, (i) to support the analysis phase and therefore to model the relevant SO aspects, the High-Level Smart Object Metamodel has been introduced; (ii) to support the design phase and to model the functional components of the system, their relationships and interactions, the ELDA-based Smart Object Metamodel (where ELDA stands for Event-driven Lightweight Distilled statecharts-based Agent) and the ACOSO-based Smart Object Metamodel (where ACOSO stands for Agent-based COoperating Smart Objects), respectively based on the ELDAMeth meta-model [15] [16] and on the ACOSO framework [17] [18], have been exploited; (iii) to support the implementation phase, the ACOSO-based Smart Object Metamodel has been specialized with respect to the JADE platform, resulting in the JACOSO Metamodel. The rest of the paper is structured as follows. Section II presents in detail the four metamodels for IoT systems development. A concrete example of a SO modelled according to the introduced approach is shown in Section III. Finally conclusions are drawn.

II. A Metamodel-based approach

The proposed approach presents metamodels at different abstraction levels in order to support the SO development phases of analysis, design and implementation. As showed in Fig. 1, the four metamodels share a common derivation from the one placed at the analysis phase since it models the main basic SO features. Therefore, several abstractions

978-1-4799-8697-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SMC.2015.231

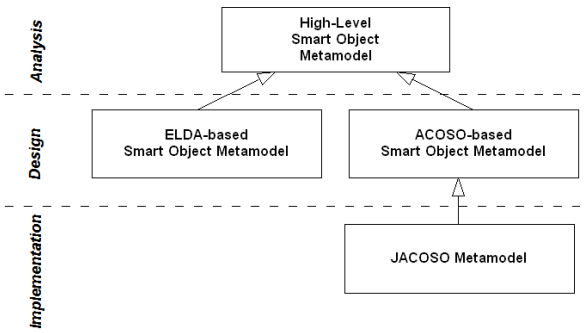

Fig. 1. Correlations between the proposed metamodels.

will be specialized and will evolve moving toward the design and implementation phases, revealing important similarities. In particular:

- at analysis phase, the *High-Level Smart Object Metamodel*, already introduced in [8], is exploited.
- at design phase, two Agent-oriented Metamodels (the *ELDA-based Smart Object Metamodel* [15] [16] and the *ACOSO-based Smart Object Metamodel*) are introduced, specializing the analysis-level metamodel. While on one hand they both adopt the Agent paradigm, on the other hand they have been conceived with different aims: in fact, an ELDA-based Smart Object can be simulated whereas an ACOSO-based Smart Object can be executed.
- at implementation phase, the *JACOSO (JADE-based ACOSO) Metamodel* [17] [18] specializes ACOSO-based Smart Object Metamodel that is specified with respect to a particular implementation metamodel based on the JADE [17] platform.

Every phase introduces new features and a higher degree of detail in the metamodels, maintaining at the same time strong relations with the higher levels metamodels.

A. High-Level Smart Object Metamodel

The metamodel portrayed in Fig. 2 is based on the one presented in [8]. It is a very high level metamodel, since its components may characterize an SO in a wide range of application domains (e.g. Smart Cities, Smart Factories, Smart Homes, Smart Grids, Smart Buildings, etc.). Specifically, it exposes both static (creator, physical properties) and dynamic (mainly related to the services provided) SOs features. Due to such reasons, the High-Level Smart Object Metamodel is highly suitable to support the SO analysis phase, modelling the main SO aspects. In particular, *SmartObject* consists of the following components:

- *Status*: represents the SO status.
- *FingerPrint*: contains the information of the SO such as its identifier, its creator, its type (e.g. smart pen, smart office, etc.) and its associated quality of service parameters (e.g. trustness, reliability, availability, etc.).
- *PhysicalProperty*: describes the physical properties of the original object without any hardware augmentation and embedded smartness.

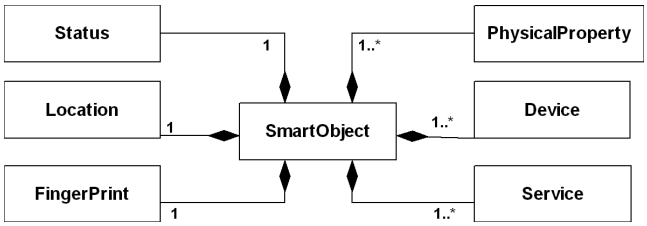

Fig. 2. High-Level Smart Object Metamodel.

- *Service*: describes a provided digital SO Service, by specifying its identifier, description, input/output parameters.
- *Device*: defines the hardware\software characteristics of a device (e.g. sensors, actuators, etc.) that allows to augment the physical object and makes it smart.
- *Location*: expresses the SO geophysical position.

B. ELDA-based Smart Object Metamodel

The metamodel portrayed in Fig. 3 refers to the ELDA-based Smart Object; it is based on the ELDA metamodel presented in [15] but new components (represented as grey blocks) have been introduced to model the SO basic features and services. In fact, the ELDA-based Smart Object metamodel specializes the High-Level Smart Object Metamodel, providing the functional components of the system, their relationships and interactions. Moreover a specific choice at the design phase has been performed by adopting the agent-based computing paradigm [10], [15]. Among the various agent models and frameworks based on lightweight architectures, asynchronous messages/events and state-based programming [17], which demonstrated great effectiveness for modelling agent-based distributed applications, the ELDA model has been choosen as it provides more effectiveness in designing complex and distributed agent-based systems. In particular, it also makes available several coordination abstractions and exploits high-level events that trigger reactive and proactive computation. Moreover a Multi Agent System (MAS) designed through the ELDA model can be seamlessly translated into platform-independent Java code by using the ELDAFramework, and then the obtained MAS implementation can be executed and validated by the ELDASim, a discrete-event simulator.

The ELDA-based Smart Object consists of the following components:

- *SmartObject*: is the ELDA agent that is composed by a single Behavior and interacts with SystemSpaces and CoordinationSpaces by a bi-directional channel. It also inherits from the SmartObject of the High-Level Smart Object Metamodel.
- *Behavior*: models the agent lifecycle and the reactions to different kinds of Events. In detail, the behavior is specified through a refined version of the FIPA template and relies on the DSC (Distilled StateChart) formalism [15], which is basically a hierarchical state machine with history pseudostates).
- *SystemSpace*: provides system services for the management of agent lifecycles, timers and resources (e.g.

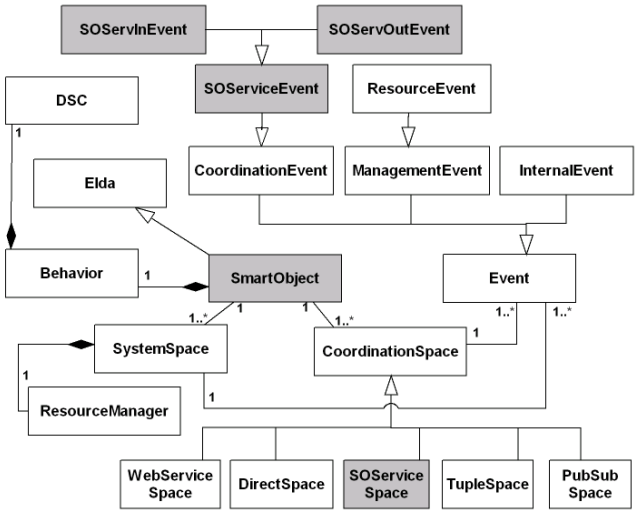

Fig. 3. ELDA-based Smart Object Metamodel.

consoles, databases, files, sensors), both internal and external to the agent. In particular, the *ResourceManager* allows the interaction with the SO Devices defined in the High-Level Smart Object Metamodel.

- *CoordinationSpace*: is a coordination structure which allows for local/remote inter-agent and agent/non-agent component interactions. In order to reach the maximum adaptability and efficiency in dynamic and heterogeneous environment, it is possible to choose among different communication models (*DirectSpace* for message passing, *PubSubSpace* for publish/subscribe, *TupleSpace* for tuple-based model, *WebServiceSpace* for webservices communication). In particular, the *SOServiceSpace* implements at the design phase the concept of Service defined in the High-Level Smart Object Metamodel and manages the interactions which contribute to realise it.

- *Event*: represents both self-triggering events (internal events) and requests to or notifications from the local agent server (management and coordination events). With reference to their target, events are divided into OUT-events (targeted towards an external agent) and IN-events (directed to the agent itself). In detail, events are classified into (i) *InternalEvents*, that proactively drive the agent's behavior; (ii) *ManagementEvents*, deputed to agent lifecycle, timers and resources management. The *ResourceManager* of the SystemSpace, exploits the *ResourceEvent*, a specific ManagementEvent; (iii) *CoordinationEvents*, that realise components interaction according to specific coordination models defined into the CoordinationSpace. An Event related to the SOServiceSpace is realised through a *SOServiceEvent*, and specifically through a *SOServInEvent* if the event models a service operation internal to to the agent itself or through a *SOServiceOutEvent* if it models a service interaction toward an external agent.

C. ACOSO-based Smart Object Metamodel

The metamodel presented in Fig. 4 refers to the ACOSO-based Smart Object. ACOSO [17] is a middleware specifically conceived for the full management of cooperating and agent-oriented smart objects. ACOSO-based Smart Object is composed by:

- *SmartObject*: specializes the SmartObject of the High-Level Smart Object Metamodel and it is composed by a Behavior and a set of subsystems that realise the SO functionalities.

- *Behavior*: models the SO actions and reactions, according to the agent-based paradigm. In particular, the behavior is defined as a set of tasks.

- *Task*: is an event-driven and state-based component which encapsulates specific objectives that need to be achieved. In particular, tasks can refer to internal operations (*SystemTask*), required for the agent lifecycle management, and to external operations (*UserDefinedTask*), defining the specific behaviors of the different SOs. Because of these reasons, the concept of task is central for the model.

- *CommunicationManagementSubsystem*: provides communication services between agents and external entities. In particular, the CommunicationManagementSubsystem incorporates a *CommunicationManager* which handles interactions by using appropriate *CommunicationAdapters*.

- *KBManagementSubsystem*: handles information pertaining the global current state of the SO, its inference rules and other useful data that can be shared among tasks. The *KBManager* exploits a local or remote knowledge base and coordinates different *KBAdapters*.

- *DeviceManagementSubsystem*: allows the management of and the interaction with sensors, actuators and devices embedded into SO. The *DeviceAdapter* allows the communication with heterogeneous devices by providing an homogeneous interface. Different DeviceAdapters are orchestrated by a *DeviceManager*.

- *Event*: is characterized by two properties: event type and event source type. An event can be (with reference to the source types): *InternalEvent*, the event source is an internal component; *ExternalEvent*, the event source is an external entity or component; *DeviceEvent*, the event source is an SO device. Event types are: *Inform*, events containing information; *Request*, events formalizing a request; *Log*, events for logging purposes; *Error*, events representing errors.

D. JACOSO Metamodel

The metamodel shown in Fig. 5 refers to the JADE-based implementation of ACOSO, namely JACOSO [17]. Considering the inheritance relationship from ACOSO, hereinafter it will be only presented the implementation components that characterize JACOSO, which are:

- *SmartObject*: is a *JADE Agent* and is composed by a set of tasks, a *CommunicationManager* and a *DeviceManager*.

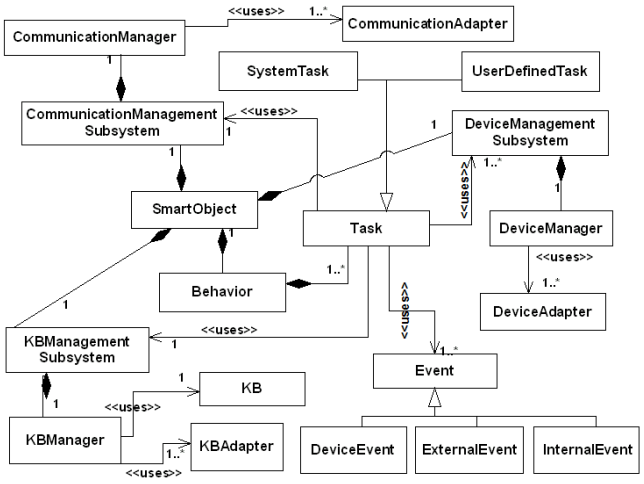

Fig. 4. ACOSO-based Smart Object Metamodel.

- *InferenceRuleTask*: implements mostly of the application logic, operating on the set of inference rules and knowledge exploited to drive the SO. It is modelled as a *JADE Behavior*.
- *ConfiguratorTask*: sets up the Tasks that have to run at the SO instantiation and the SO components (e.g. sensors, actuators or other SO devices). It is modelled as a *JADE Behavior*.
- *EventDispatcher*: manages all the SO internal events and forwards events to the Tasks that are specifically registered to them, playing a crucial role in the event-based architecture. It is modelled as a *JADE Behavior*.
- *ACLCommunicationAdapter*: allows a direct message passing between heterogeneous SOs by exploiting JADE ACLMessages.
- *TopicPSAdapter*: allows an asynchronous one-to-many message mechanism based on the pattern Publish Subscribe.
- *BMFAdapter*: provides JACOSO with the capability to control a Wireless Sensor and Actuator Network (WSAN) through the BMF (Building Management Framework) [19] framework. The BMF is a domain-specific framework, expressly conceived for the management of WSANs deployed in physical environments, which enables the control of spaces and devices/equipment.
- *SPINEAdapter*: provides JACOSO with the capability to control a Body Sensor Network (BSN) through SPINE (Signal Processing In-Node Environment) [20]. SPINE is a domain-specific framework designed for managing BSNs that aims at decreasing the development time of distributed BSN applications.

The metamodel shown in Fig. 5 also highlights the components that need to be redefined to create a new SO. In particular, the JACOSO architectural blocks do not need to be changed if a new SO is created. These blocks are the so called ”frozen spots” and, in Fig. 5, are represented in white. On the other hand, some specific components need to be re-implemented for new SOs to model SO functions and behaviors. Such components are indicated as ”hot spots” and are represented as grey blocks. In particular, the hot spots are (i) the ConfiguratorTask, since the Tasks setup phase is strongly variable; (ii) the UserDefinedTask, that can be extended by several Tasks; (iii) the CommunicationManager, because the communication patterns may be the existing ones (direct ACL message passing or Publish/Subscribe) or customized ones (e.g. Web Server, Socket, etc.); (iv) the DeviceManager, in order to make the SO able to interact with the desired devices.

E. Discussion

In the previous sections a set of operational metamodels, each of which is functional to a different development phase, has been presented. Here, the relations among the entity concepts in the different phases will be explained.

In details, from analysis to implementation, related metamodels provide abstractions that evolve, being refined from high-level concepts to implementable components, such as:

- *SmartObject*, in the High-Level Smart Object Metamodel is a very high level entity and does not contemplate any behavioral element. On the other hand, in the ELDA-based and in the ACOSO-based Smart Object metamodels, the SmartObject is agentified and its behavior defined. In particular, in the ELDA-based metamodel the behavior is defined through a DSC whereas in the ACOSO metamodel (and so in the JACOSO metamodel, which extends it) the behavior is modelled as a set of tasks.
- *Service*, in the High-Level Smart Object Metamodel no implementation details have been shown. In the ELDA-based metamodel services are exposed through the *SOServiceSpace* and the interactions with the SmartObject for the service provisioning/utilisation are realized through *SOServiceEvents*. In the ACOSO-based Smart Object metamodel (and so in the JACOSO one) services are mapped on *UserDefinedTask* and the related interactions are enabled by events.
- *Device*, the High-Level Smart Object Metamodel transparently connects devices to the SmartObject. In the ELDA-based metamodel devices (and other agent resources) are accessible through the *ResourceManager* within the *SystemSpace*. In the ACOSO-based Smart Object metamodel devices are managed by the *DeviceManagementSubsystem* which, through the *DeviceManager*, handles all the available *DeviceAdapters*. In the JACOSO metamodel devices are set up by the *ConfiguratorTask* so that all the tasks in the system can directly access the *DeviceAdapters* through the *DeviceManager*.
- *Basic Features*, while the High-Level Smart Object Metamodel elicits a set of features such as *Location*, *Status*, *PhisicalProperty* and *FingerPrint*, in the other metamodels these information are split on several components. In details, in the ELDA-based metamodel all the information is between the *(Agent) SmartObject* and the *Behavior*. In the ACOSO metamodel the

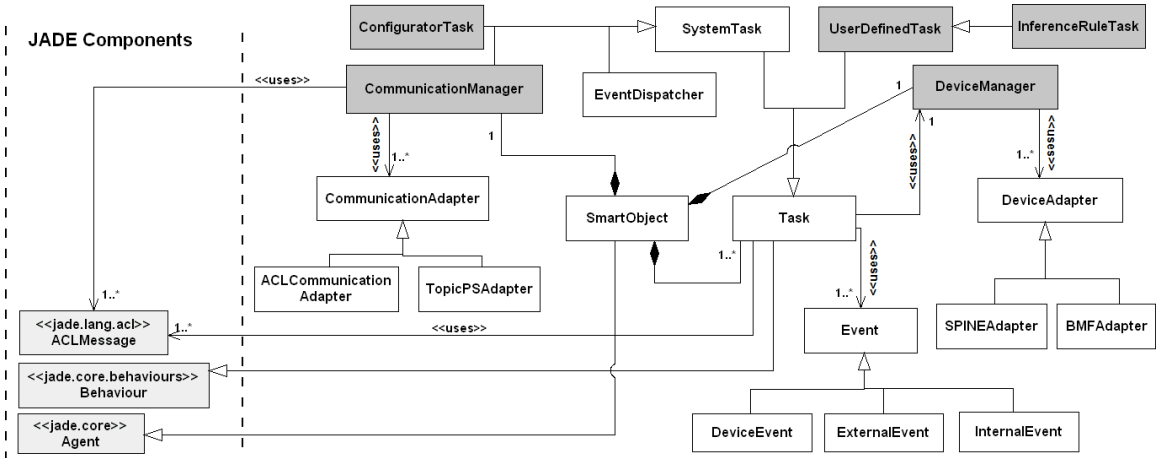

Fig. 5. JACOSO Metamodel.

information is spread between the *(Agent) SmartObject*, the *KBManagementSubsystem*, the *Behavior*, and the tasks. Finally, in the JACOSO Metamodel these features are split between the *(Agent) SmartObject* and in the tasks, especially in the *InferenceRuleTask*.

It is worth noting that in the High-Level Smart Object Metamodel the communication is not highlighted. Differently, in the ELDA-based metamodel the communication is demanded to the *CoordinationSpace* and based on events; in the ACOSO-based Smart Object (and JACOSO) metamodel the communication is managed through the *CommunicationManager* and a set of *CommunicationAdapters*.

III. An Example: Modelling a Smart Object

In order to show the application of the metamodel approach, this Section proposes the case study presented in [18], through the use of the metamodels defined in Section II. The case study is simple but functional, since it allows to show the application of the proposed metamodel-based approach. The SmartOffice provides smart services which perform sensing and actuation in the office environment and give suggestions to its users. Fig. 6 shows the SmartOffice High-Level Model, obtained as instantiation from the SmartObject High-Level Metamodel; in detail, basic SmartOffice static features (identifier, creator, location, associated quality parameter, etc.), dynamic information about its current status (temperature, user presence, light and humidity value), and the list of the available devices (one laptop and two different sensors) are represented. The information gathered by the sensors (light, presence, temperature and humidity) is processed with respect to a set of inference rules in order to support users in their daily working activities and to avoid energy wasting. In particular, the SmartOffice provides three smart services: (i) *PresenceService*, which detects the presence of people inside the office; (ii) *LightService*, which informs if the lights are switched on/off; (iii) *VisualizationService*, which shows notifications on the laptop monitor. By following the approach introduced in this paper, the SmartOffice High-Level Model has been specialized in the JACOSO Model of Fig. 7 through different steps which are the agentification of the SmartOffice, the modelling of its behavior and its services on proper tasks, and

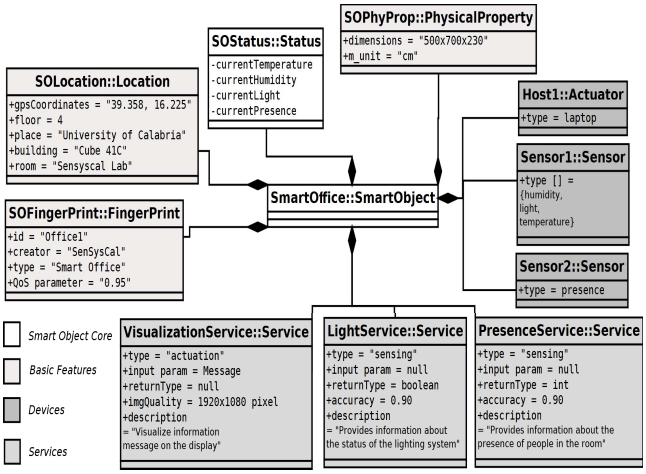

Fig. 6. SmartOffice High-Level Model.

the specialization of the required device and communication adapters. The specific elements of SmartOffice are represented as grey blocks. In detail, the hot spots components (ConfiguratorTask, DeviceAdapter and CommunicationAdapter) have been redefined in order to support the SmartOffice specific objectives and features. To realise the SmartOffice services specific UserDefinedTasks, namely the InferenceRuleTask, the VisualizationTask, the AmbientPresenceTask and the LightPresenceTask, have been implemented, as well as the BMFAdapter, which allows the management of the SmartOffice devices and specifically of the light and presence sensors, is used.

IV. Conclusion

This paper has introduced a metamodel-based approach for IoT systems development. In particular three levels, respectively linked to the analysis, design and implementation phases, of SO metamodels have been proposed, providing a seamless support among the different phases of SO development process. To this purpose, the ELDA meta-model and ACOSO framework have been used. Finally, the SmartOf-

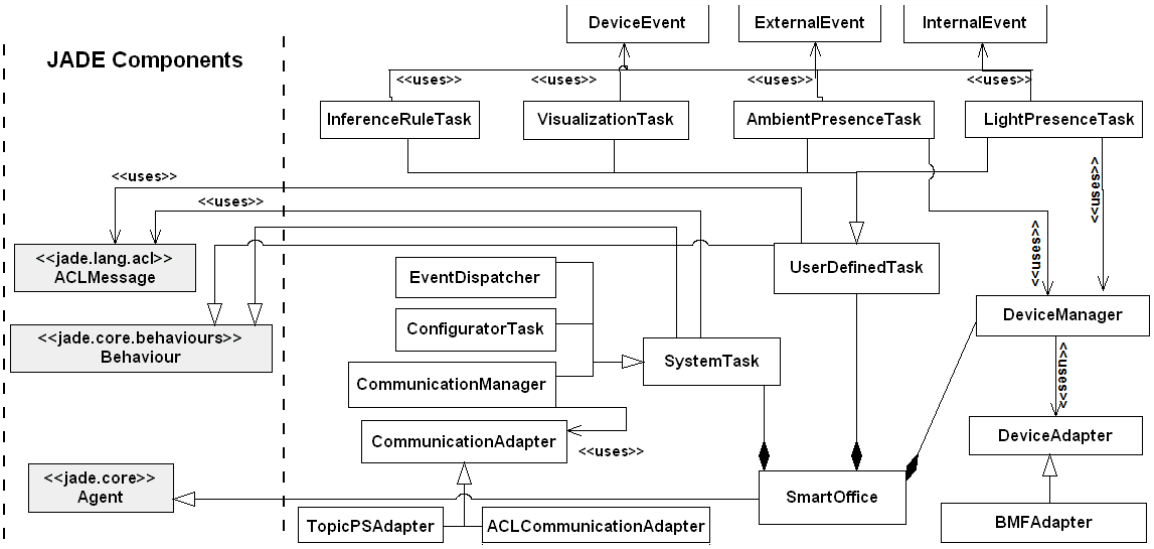

Fig. 7. SmartOffice JACOSO Model.

fice has been built as a case study following the proposed approach from the high-level analysis to the JACOSO-based implementation. Specific issues related to SO security and SO resources management are not tackled; however, the flexibility that characterizes the proposed approach allows to easily extend the work in these directions. The proposed approach is a first building block toward the definition of a full-fledged methodology for SO-based systems development, which is an on-going work.

ACKNOWLEDGMENT

This work has been partially supported by DICET INMOTO Organization of Cultural Heritage for Smart Tourism and REal Time Accessibility (OR.C.HE.S.T.R.A.), project funded by the Italian Government (PON04a2D), and by BodyCloud, project funded by the Calabria Region, Department 11, PO CALABRIA FSE 2007-2013 - ASSE IV.

REFERENCES

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, *Internet of things: Vision, applications and research challenges*, Ad Hoc Networks 10.7 (2012): 1497-1516.

[2] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, *Smart objects as building blocks for the internet of things*, IEEE Internet Computing, Vol. 14, n.1, pp. 44-51, 2010.

[3] F. Mattern, and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*, From active data management to event-based systems and more, Springer Berlin Heidelberg, pp. 242-259.

[4] S. Kent, *Model driven engineering*, Integrated formal methods. Springer Berlin Heidelberg, 2002.

[5] I. Corredor, A. M. Bernardos, J. Iglesias, and J.R. Casar, *Model-Driven Methodology for Rapid Deployment of Smart Spaces Based on Resource-Oriented Architectures*, Sensors, Vol. 12, pp. 9286-9335, 2012.

[6] A. Serbanati, C.M. Medaglia, and U. B. Ceipidor, *Building blocks of the internet of things: State of the art and beyond*, Deploying RFID-Challenges, Solutions, and Open Issues, C. Turcu (ed.), InTech, 2011.

[7] F. Kawsar, T. Nakajima, J.H. Park, and S.S. Yeo, *Design and implementation of a framework for building distributed smart object systems*, J. Supercomput, Vol 54, n.1, pp. 4-28, 2010.

[8] G. Fortino, M. Lackovic, W. Russo, and P. Trunfio, *A discovery service for smart objects over an agent-based middleware*, Lecture Notes in Computer Science, 8223 LNCS, pp. 281-293. Springer, Heidelberg, 2013.

[9] D. Uckelmann, M. Harrison, and F. Michahelles (eds.), *Architecting the Internet of Things*, Springer, 2011.

[10] A. Omicini, and S. Mariani, *Agents & multiagent systems: En route towards complex intelligent system*, Intelligenza Artificiale, Vol. 7, pp. 153-164, 2013.

[11] G. Fortino, A. Guerrieri, and W. Russo, *Agent-oriented smart objects development*, Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on, pp. 907-912, 2012.

[12] S. Clayman, and A. Galis, *INOX: A Managed Service Platform for Inter-Connected Smart Objects*, ACM IoTSP, 2011.

[13] J. Strassner, S. Kim, and J. Won-Ki Hong, *The Design of an Autonomic Communication Element to Manage Future Internet Services*, C.S. Hong et al. (Eds.), APNOMS 2009, LNCS 5787, pp. 122132, 2009.

[14] D. Kelaidonis, A. Somov, V. Foteinos, and G. Poulios, *Virtualization and Cognitive Management of Real World Objects in the Internet of Things*, 2012 IEEE International Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing, 2012.

[15] G. Fortino, A. Garro, S. Mascillaro, and W. Russo, *Using event-driven lightweight DSC-based agents for MAS modelling*, International Journal of Agent-Oriented Software Engineering, Vol. 4, n. 2, pp. 113-140, 2010.

[16] G. Fortino, and W. Russo, *ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems*, Information and Software Technology, Vol. 54, n. 6, pp. 608-624, 2012.

[17] F. Bellifemine, A. Poggi and G. Rimassa, *Developing multi agent systems with a FIPA-compliant agent framework*, Software Practice and Experience, Vol. 31, pp. 103-128, 2001.

[18] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, and W. Russo, *An agent-based middleware for cooperating smart objects*, Highlights on Practical Applications of Agents and Multi-Agent Systems, Springer Berlin Heidelberg, pp. 387-398, 2013.

[19] G. Fortino, A. Guerrieri, G.M.P. O'Hare, and A. Ruzzelli, *A flexible building management framework based on wireless sensor and actuator networks*, Journal of Network and Computer Applications, Vol. 35, Issue 6, pp. 1934-1952, ISSN 1084-8045, 2012.

[20] G. Fortino, R. Giannantonio, R. Gravina, P. Kuryloski, and R. Jafari, *Enabling Effective Programming and Flexible Management of Efficient Body Sensor Network Applications*, IEEE Transactions on Human-Machine Systems, Vol. 43, n.1, pp. 115-133, 2013.