CONCEPT HIERARCHIES FOR SENSOR DATA FUSION IN THE COGNITIVE IoT

Franco Cicirelli and Giandomenico Spezzano
Institute for High Performance Computing and Networking (ICAR)
CNR - National Research Council of Italy
87036 Rende(CS) - Italy
Email: cicirelli@icar.cnr.it, spezzano@icar.cnr.it

KEYWORDS

Sensor data fusion; Internet of Things; Multi-agent systems; Statecharts; MAB museum.

ABSTRACT

Sensor data fusion refers to technological solutions aiming at collecting, classifying and complementing data coming from multiple sensors. It has the potential of enabling context awareness which, on the other hand, represents a huge potential to be exploited in the field of IoT applications. Sensor fusion and IoT have to deal with multi-faced issues like heterogeneity, sensor/actuator management, data accuracy and reliability. This paper proposes a multi-tier approach dealing with sensor fusion and IoT aspects in a modular way. The approach relies on the use of the agent metaphor, statecharts and on the Rainbow multi-agent platform. Agents can be dynamically added and removed from an application thus promoting system openness and scalability. Heterogeneity and distribution issues are transparently managed by Rainbow which hides the physical layer on top of which the applications are built. As a significant case study, the approach was exploited for the implementation of a working prototype devoted to improve security of some artworks (statues) of the MAB museum located in the city of Cosenza, Italy.

INTRODUCTION

Nowadays, sensors are exploited in a huge variety of applications ranging from healthcare, transportation and logistic, surveillance, environmental monitoring, smart city and so forth. The sensing capabilities of the infrastructures and devices surrounding our daily lives are constantly improving and becoming more affordable. The widespread diffusion of sensors was also favored by the ever-increasing attention which is given to the field of the so called "Internet of Things" (IoT) (Miorandi et al., 2012). The basic idea of the IoT is a pervasive presence around us of a variety of things or objects such as RFID tags, sensors, actuators, mobile phones, etc. which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals (Atzori et al., 2010).

Important issues which are related to the development of significant sensor-based applications are data collection, classification and complementation. Sensor data fusion, or simply *sensor fusion* (SF) (Karimi, 2013), refers to technological solutions having the aim of addressing the above issues. The goal is to increase both the accuracy and reliability of sensed data as well as to enable context awareness (Bicocchi et al., 2014; Karimi, 2013; Schilit et al., 1994). Context awareness refers to the capability of disclosing information about the context (or situation) in which the data get acquired/generated. This allows to validate an event or an assumption made on sensed data, or to compensate the lack of complete information about the sensed environment thus permitting to take decisions and/or properly react to complex environmental stimuli. As an example, a person may not see flames under the hood of a car, but the smell of burning rubber and the heat coming from the dash would suggest that it is prudent to leave the car because the engine is on fire.

This paper proposes an approach for the development of distributed SF applications based on the IoT paradigm. The approach promotes separation of concerns through the use of a *multi-tier architecture* which allows to deal with SF and IoT issues in a modular and orthogonal way. The agent metaphor (Woolridge and Wooldridge, 2001) is used to structure the application logic, whereas agents' behavior is modeled by using statecharts (Booch et al., 2000; Cicirelli et al., 2011; Harel, 1987; Kielar et al., 2014). Statecharts are a state-based formalism which allows to specify complex and time-dependent behaviors by using a graphical notation. Complexity of a model is dealt with the use of hierarchical constructs.

Proposed approach permits the exploitation of *concept hierarchies* which allow the *classification* of *high-level situation* as well as the definition of *reaction-driven policy* which work on a multiplicity of low-level events. Concept hierarchies are useful to represent a system as a set of abstractions normally used by humans when reasoning about complex systems. The concepts which are introduced in different tiers, or within the same tier, can be related together (i.e., orchestrated) in order to specify complex reaction patterns to the stimuli coming from the external environment. The overall goal is to move from the IoT towards the Cognitive IoT (Tsai et al., 2014; Wu et al., 2014) where objects interact and operate by acquiring knowledge from the surrounding environment and by following a context-aware perception-action operational cycle.

Each tier, in the proposed architecture, groups agents on the basis of some predefined agent roles. For instance, the role *awareness* was defined, and the corresponding tier will contain agents devoted to managing the knowledge acquired on the whole system and to properly act on the system itself. A *low-level* tier, instead, contains objects (not necessarily agents)

Proceedings 30th European Conference on Modelling and Simulation ©ECMS Thorsten Claus, Frank Herrmann, Michael Manitz, Oliver Rose (Editors)
ISBN: 978-0-9932440-2-5 / ISBN: 978-0-9932440-3-2 (CD)

used to deal with the hardware/driver system entities.

The agent metaphor was chosen for its capability to naturally meet some needs of distributed SF applications like the autonomy in reacting to stimuli coming from the external environment, and its suitability to operate in a open and dynamic environment. Heterogeneity and distribution issues are transparently managed by the Rainbow (Giordano, Spezzano and Vinci, 2014) agent platform which has the capability of abstracting the physical layer on top of which the applications are built.

The proposed approach is novel because it is *comprehensive*, and explicitly considers aspects tied to sensors and actuators. Comprehensive means that it takes into account all the issues relevant to the development of SF applications starting from the awareness of the context down to the issues related to the management of physical layer. Flexibility and modularity is provided by the tiered-based architecture and by the use of the agent metaphor paired with statecharts.

As a significant case study, the approach was exploited for the implementation of a working prototype devoted to improve security of some artworks (statues) of the open-air MAB museum located in the city of Cosenza, Italy.

The paper is structured as follows. First, the proposed approach is described along with a characterization of the proposed multi-tier architecture. Then, an overview of the Rainbow agent-based platform is provided. After that, the case study is shown. Conclusions and indications about future research avenues are provided in the last section.

A MULTI-TIERS APPROACH FOR SF

The proposed multi-tiers approach allows the development of SF applications by promoting both separation of concerns and modularity. In each tier it is possible to focus on specific concerns having a different level of abstraction. The *low-level* tier, for instance, copes with software drivers managing physical devices. The *high-level* one deals with context/situation awareness. The terms *low* and *high* refer to the abstraction level exploited for developing an application or a working system. The use of tiers fosters both a *top-down* and *bottom-up* software development schema. A top-down approach can be used when an application is developed from scratch. A bottom-up approach, instead, is suggested to be used when an application is built on top of some preexisting components. The introduced tiers are reported in Figure 1. All the tiers except the lower one, are populated by means of agents. The *low-level* tier instead contains objects which are referred as *Virtual Objects* (VOs). Such name mirrors the fact that VOs are used to virtualize, i.e., to abstract, the physical devices used for realizing an application. A description of the introduced tiers follows.

- *Low-level*: it hosts the VOs that directly interact with physical devices. In the case of simple devices, such virtual objects directly manage information like distances, level of noise or temperature (both for sensing and actuation purposes). In other more complex cases, instead, virtual objects interact with the drivers of the physical devices and can abstract complex operations (e.g., those related to the management of a webcam). The goal of this tier is to decouple, from a sensing/actuation point of view, the functionalities from the equipment offering them. A change in the physical devices affects virtual objects but it should not affect the entities belonging the the other (upper) tiers. The challenge, here, is to decouple *functionalities* from *implementation*. A one-to-many mapping exists between VOs and the hardware equipments.

- *Sensing/Actuator*: the agents defined in this tier are boundary entities whose goal is turning the functionalities offered by virtual objects into location-independent services offered by agents. As a general guideline, it is suggested to develop a dedicated agent for all the functionalities offered by a single device abstracted through a VO. These dedicated agents must be co-located on the computational node hosting the virtual objects they have to interact. From a functional point of view, the Sensing/Actuator agents do not introduce new functionalities nor modify the existing ones. They can instead mediate the use of the wrapped functionalities by enforcing, for instance, *negotiation procedures* and/or *access policies* useful for guaranteeing an exclusive use, or a time-based use, of such functionalities.

- *CSensing/CActuator*: here the goal is to compose, or modify, functionalities offered by the agents belonging to the Sensing/Actuator tier. Composition of functionalities aims at obtaining aggregate operations starting from simpler ones, and at orchestrating independent functionalities in order to pursue a common task. Modify a functionality means to extend/improve it, e.g., for increasing the reliability of sensed data, or for allowing the transparent management of a group of redundant actuator devices viewed as a single device. As an example, if it is required to average the data coming from multiple temperature sensors, it is possible to consider an *averaging agent* which interacts with the agents related to the temperature sensors in order to get data and average it. In another case, the functionality offered by a temperature agent which provides only data in real-time, can be extended by an agent providing information about the maximum and minimum value of read data. Modify a functionality can mean also to hide it in order to meet specific application constraints. A many-to-many relationship can exist between agents in this tier and the lower one.

- *Concept*: this tier introduces a *significant increase* in the exploitable abstraction level. In fact, in the tiers previously described we have been dealing only with issues directly related to sensing and controlling the environment, e.g., reading a temperature value or closing a light. In this tier, instead, an actuation to be performed, and/or some environmental stimuli, are mapped onto more abstract concepts. Each concept is then modeled/implemented through an agent. The goal, here, is to allow reasoning by using abstract concepts directly related to the specific domain of the application being developed. For instance, in this tier, it is possible to introduce the concept of *climatic health* which depends on the value of the temperature

and humidity actually existing in a given room. In the same way, the concept of *office security* can be introduced in order to control a door office and makes it alarmed after a certain hour. A many-to-many relationship can exist between agents in this tier and the lower one.

- *CConcept*: this tier is equivalent to the CSensing/Cactuator one. In this case, though, composition and modification refer to concepts.
- *Awareness*: this is the tier permitting to operate, and to reason, at the highest level of abstraction. Agents introduced in this tier have a holistic vision of all the concepts which are related to the developed application and which govern the implemented system. Introduced concepts (or a subset of them) are orchestrated in order to pursue application goals. Information and data can be complemented in order to augment the knowledge about the context in which the application runs. For instance, if an application for managing *safety at home* is considered, information indicating that a person is found lying in the bathroom can be complemented in order to infer that the person could be afflicted by a malaise.

Figure 2 reports an example in which a certain number of agents and VOs populate the various tiers and interact among them. In the figure it is highlighted that communication among the introduced entities can be both unidirectional and bidirectional. Even if in the reported example all the tiers contain some entities, on the bases of application needs, a tier could also be empty. It is not allowed to consider intra-tier communication among the entities located in the tiers from T#3 to T#0 whereas it can occur, instead, in tiers T#4 and T#5. All of this favors to deal with the *true* application logic (e.g., the goal-oriented logic) only in the tiers with a higher level of abstraction.

Once the tiers are populated, and having established the communication patterns among the introduced entities, the next step is to define the behavior of each agent. In the approach here proposed, modeling the behavior of agents relies on the use of statecharts (Booch et al., 2000; Cicirelli et al., 2011; Harel, 1987; Kielar et al., 2014). As a consequence, the whole system can be seen as a network of interacting statecharts. Statecharts are well suited to model entities having a complex and time-dependent behavior. They enable both a hierarchical and modular modeling approach. All of this permits to face with the well-known state-explosion phenomenon which arises with large and complex models. The basic mechanism upon which statecharts rely, consists in the possibility of nesting a subautomaton within a (macro) state thus encouraging stepwise refinement of a complex behavior.

THE RAINBOW PLATFORM

Rainbow (Giordano, Spezzano and Vinci, 2014; Giordano, Spezzano, Vinci, Garofalo and Piro, 2014) is a distributed agent-based IoT platform composed of single-board computer nodes, such as the Raspberry PI 2, which is well suited to connect massive-scale networks of sensors and actuators to the Cloud. It was chosen as the reference platform exploitable for supporting the approach so far described.

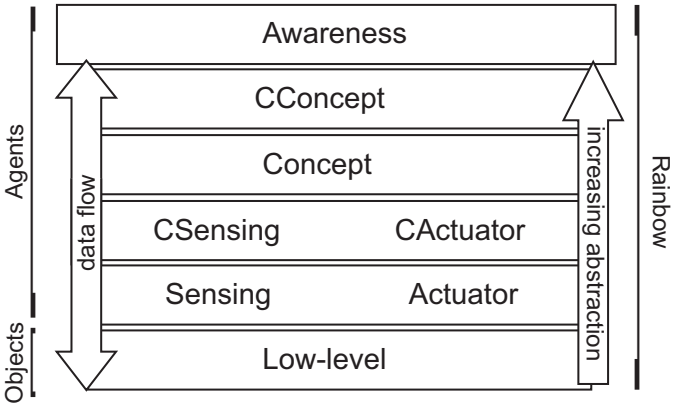

Fig. 1. The proposed multi-tier architecture

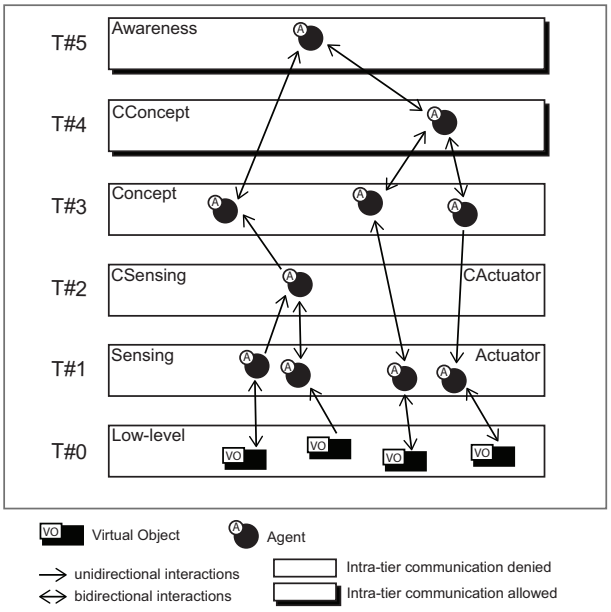

Fig. 2. The multi-tier architecture: an example of communication details

The physical part of the Rainbow architecture is constituted of sensors and actuators, together with their relative computational capabilities, which are directly immersed in a physical environment. Physical entities are usually spread across a large (even geographic) area. All of this implies that the controlling part of an application must be intrinsically distributed. Sensors and actuators are partitioned into groups, each of which is managed by a single computing node. A goal of Rainbow is to bring the computation (e.g., the controlling part) as close as possible to the physical part. Each computing node hosts multi-agent applications designed to monitor multiple conditions, or to operate activities within a specific environment. For this purpose, each node contains an *agent server* that permits agents to be executed properly. Agents can be intelligently assisted by cloud services, that support complex analytics, modeling, optimization and visualization tools, to make better operational decisions.

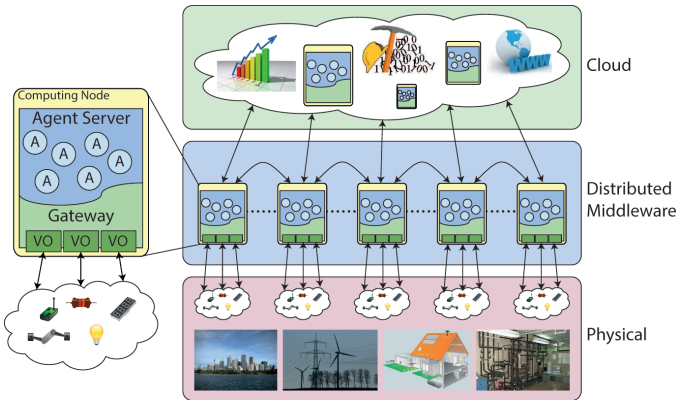

Fig. 3. The Rainbow architecture

The architecture of Rainbow (see Figure 3) is composed of the Cloud layer, the Intermediate layer and the Physical layer.

In the Intermediate layer, sensors and actuators of the Physical layer are represented as virtual objects (VOs). VOs offer to the agents a transparent and uniform access to the physical part through the use of a well-established interface. A VO allows agents to connect directly to devices without care about proprietary drivers and without addressing some kind of fine-grained technological issues. Each VO comprises *functionalities* directly provided by the physical part. Essentially, a VO exposes an abstract representation (i.e., a machine-readable description) of the features and capabilities of the abstracted physical objects spread in the environment. Functionalities exposed by different types of VOs can be combined in a more sophisticated way on the basis of event-driven rules which affect high-level applications and end-users. More in particular, all the devices are properly wrapped in VOs which, in turn, are enclosed in distributed *gateway containers*. The computational nodes that host the gateways and the agent server represent the middle layer of the Rainbow architecture.

Gateways and agent servers are co-located in the same computing nodes in order to guarantee that agents directly exploit the physical part through VO abstraction. Instead of transferring data to a central processing unit, we actually transfer the process (i.e., the fine-grain agent's execution) toward the data sources. As a consequence, less data needs to be transferred over a long distance (i.e., toward remote hosts) and local access and computation will be fostered in order to achieve good performance and scalability.

The upper layer of Rainbow architecture concerns the cloud part. This layer addresses all the activities that cannot be properly executed in the middle layer like, for instance, algorithms needing a complete knowledge of the whole system, tasks that require high computational resources, or when a historical data storage is mandatory.

SENTIENT STATUES IN THE MAB MUSEUM

The MAB is a particular open-air museum located on the main artery of the new part of the city of Cosenza (Calabria, southern Italy). The MAB was born thanks to the donation of the wealthy collector Carlo Bilotti, native to Cosenza but immigrant to America, who decided to donate his art collection to his city of birth after his death in 2006. The MAB houses some prestigious sculptures by artists like Salvator Dalí, Giorgio De Chirico and by some other artists of Calabria. Since its establishment, the museum has been the subject of some vandalisms and of some accidents that damaged some artworks.

The goal of the case study which is here described, is to use sensor fusion concepts in order to *furnish* to the statues of the MAB some *virtual senses* able to *make them aware* of what happens around them. The aim is therefore to make the statues *sentients*, i.e, able to discover and recognize a dangerous situation and to react to it with the aim of preventing damages and averting a possible hazard. The basic idea is to equip a sculpture with some suitable sensors and actuators enabling to sensing the environment and operate deterrence actions, e.g, asking for help. Obviously, in order to avoid causing damage to the artworks, both sensors and actuators should not be *worn* by the sculpture but deployed in their vicinity in some safe places.

A prototyped system was developed by using the approach described in this paper. The system was developed from scratch and, for this reason, a top-down development schema was exploited. More in particular, in a first phase, the layers of the multi-tiers architecture were populated starting from the *awareness* layer. Then, the behavior of each agent was modeled through statecharts and the code for the modeled agents and virtual objects was developed. Finally, the system was implemented. System implementation consisted in preparing the hardware equipment (i.e., the chosen sensors and actuators), connecting it to a Rainbow server and deploying the developed code of both agents and virtual objects over that server. The simplest devices were connected to the Rainbow server trough some Arduino devices.

In the following, a description of the developed prototype is reported. A description of the basic features of statecharts is provided too.

Populating the layers of the multi-tiers architecture

This is the most important phase because it defines the roles and the functionalities of all the entities which constitute the system. In Figure 4 are reported the agents and the VOs defined for the application.

In the tier of *awareness* we defined the agent which models the *mood* of a sculpture. On the base of the stimuli received from the environment, a statue can found itself in a status like quiet, worried or terrified. A detailed description of these status is provided in the next subsection. A statue perceives its surrounding environment through its *senses*. Three different senses were considered, namely the *touch*, the *sight* and the *hearing*. For each of the considered senses, a specific agent was defined in the *concept* layer.

On the base of the current status, some *safeguard* actions can be executed. For this purpose, a suitable agent is defined in the *cconcept* tier. This safeguard agent models a composite concept which is tied to the concepts of both *speaking* and making *deterrent actions* (see the Speech and Deter agents in the *concept* layer of Figure 4).

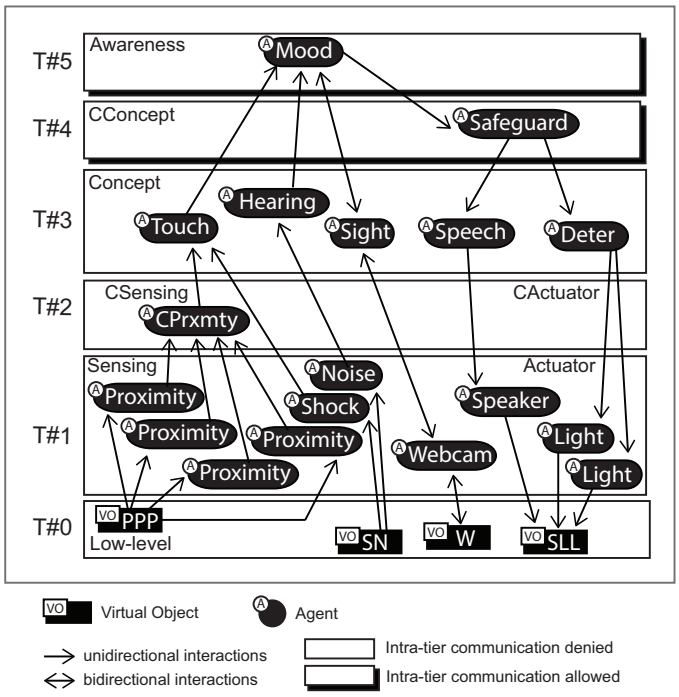

Fig. 4. Realizing sentient statues: agents' hierarchy and virtual objects

From the *sensing* tier it emerges that (i) the sense of touch is implemented by using both some proximity sensors and a shock sensor, (ii) the hearing is implemented by a noise sensor and (iii) the sight is realized by using a webcam. Proximity sensors are grouped in a single composite sensor (see the CPrxmty agent in the csensing layer of Figure 4). The latter composite entity was considered to highlight that, in this application, all the proximity sensors are indistinguishable and equivalents.

From the actuator point of view, the safeguard activities were implemented by considering some speech abilities realized through a speaker, and by some deterrent actions which are based on the flashing of warning lights.

Four VOs were considered: one for the management of proximity sensors, one for the shock and noise sensor, and another two respectively for the webcam and the warning lights.

Basic features of statecharts

A *state* of a statechart can recursively be decomposed into a set of *substates*, in which case such a state is said to be a *macro* state. A state that is not decomposed is said to be a *leaf* state. The root state of the decomposition tree is the only one having no parent and it is referred to as the *top* state. At a given point in time, a statechart finds itself simultaneously in a set of states that constitutes a path leading from one of the leaf states to the top state. Such a set of states is called a *configuration* (Harel, 1987). A configuration is uniquely characterized by the only leaf state which it contains. Each macro state specifies which of its substates must be considered its initial state. This substate is indicated by means of a curve originating from a small solid circle and ending on its border. This curve, although technically

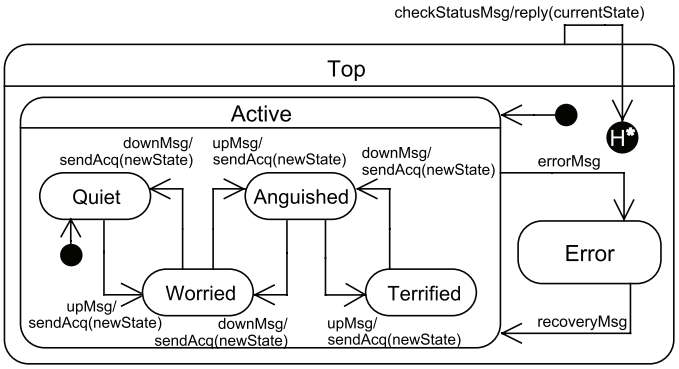

Fig. 5. Statechart of a Mood agent for a sentient statue

is not a transition, is referred to as the *default transition*. State transitions are represented by edges with arrows. Each transition is labelled by $ev[guard]/action$ where ev is the trigger (event or message causing the transition), $guard$ a logical condition which enables the transition when it evaluates to true, and action the action "á la Mealy" associated with the transition. When omitted, the guard is implicitly assumed to be true. Both source and destination of a transition can be states at any level of the hierarchy. Firing a transition leads the statechart to switch from one configuration to another. When a configuration is left, each of its macro states keeps memory of its direct *substate* that is also part of the configuration. This *substate* is referred to as the *history* of the macro state.

A transition always originates from the border of a state, but it can reach its destination state either on its border or ending on a particular element called *history connector* or H-connector. Such a connector is depicted as a small circle containing an H (*shallow history*) or an H* (*deep history*), and it is always inside the boundary of a compound state. Both shallow and deep history connectors allow to re-enter in a macro state by exploiting history information.

Modeling agents through statecharts

The behavior of each agent identified in Figure 4 was modeled through statecharts. For simplicity, here, only the statechart of the *mood* agent is described. The statechart is reported in Figure 5.

The mood agent can find itself in the *Active* or *Error* state. The above two status are contained in the *Top* macro-state having the responsibility or responding to *check* messages asking for the current status of the statue. Each time one of such a message is received, the statue replies with its actual state and then returns in the leaf-state owned just before receiving the check message. All of this is mirrored in the model by the use of the deep-history connector H^* which is attached to the state-transition edge departing from the *Top* state.

The *Active* state is also a macro-state. It contains the leaf-states related to the actual "morale" of a statue, namely *Quiet*, *Worried*, *Anguished* and *Terrified*. A mood agent moves from a status to another one by receiving *upMSG* and *downMSG* messages. These messages are generated by the senses of the statue. For instance, in the case the *touch* identifies the presence of someone in the nearness of the statue, an *upMSG*

```
[12/11/2015 12:15:13] Mood#24: received UpMsg, current state QUIET,
next state WORRIED. Send WORRIED to: Safeguard#12, Sigth#5
[12/11/2015 12:20:21] Mood#24: received DownMsg, current state WORRIED,
next state QUIET. Send QUIET to: Safeguard#12, Sigth#5
[12/11/2015 12:21:42] Mood#24: received UpMsg, current state QUIET,
next state WORRIED. Send WORRIED to: Safeguard#12, Sigth#5
[12/11/2015 12:22:12] Mood#24: received UpMsg, current state WORRIED,
next state ANGUISHED. Send ANGUISHED to: Safeguard#12, Sigth#5
[12/11/2015 12:23:19] Mood#24: received UpMsg, current state ANGUISHED,
next state TERRIFIED. Send TERRIFIED to: Safeguard#12, Sigth#5
[12/11/2015 12:26:29] Mood#24: received CheckMsg, current state
TERRIFIED. Send TERRIFIED to: OtherAgent#7
[12/11/2015 12:26:45] Mood#24: received DownMsg, current state TERRIFIED,
next state ANGUISHED. Send ANGUISHED to: Safeguard#12, Sigth#5
[12/11/2015 12:27:45] Mood#24: received UpMsg, current state ANGUISHED,
next state TERRIFIED. Send TERRIFIED to: Safeguard#12, Sigth#5
[12/11/2015 12:28:02] Mood#24: received DownMsg, current state TERRIFIED,
next state ANGUISHED. Send ANGUISHED to: Safeguard#12, Sigth#5
[12/11/2015 12:28:45] Mood#24: received DownMsg, current state ANGUISHED,
next state WORRIED. Send WORRIED to: Safeguard#12, Sigth#5
[12/11/2015 12:29:10] Mood#24: received CheckMsg, current state
WORRIED. Send WORRIED to: OtherAgent#7
[12/11/2015 12:29:25] Mood#24: received DownMsg, current state WORRIED,
next state QUIET. Send QUIET to: Safeguard#12, Sigth#5
```

Listing 1: An example of generated log

TABLE I. DESCRIPTION OF THE STATUE STATUS

Status	Description
Quiet	The data coming from the sensors gets elaborated. The webcam is turned on with a fixed shot on the statue. The lights and the speakers are switched off.
Worried	The data coming from the sensors get elaborated. The webcam is turned on with a movable frame around the statue. The lights and the speakers are switched off.
Anguished	The data coming from the sensors gets elaborated. The webcam is turned on with a movable frame around the statue. The lights are flashing and the speakers are switched off.
Terrified	The data coming from the sensors gets elaborated. The webcam is turned on with a movable frame around the statue. The lights are flashing and the speakers invited to move away from the statue
Error	The data coming from the sensors is discarded. The webcam, the lights and the speakers are switched off.

message is issued. On the contrary, when a previously sensed entity is no longer sensed, a *downMSG* message is sent to the mood agent. In such a way, a statue stays in the *Quiet* state when no senses reveal the presence of anyone in the nearness of the statue. On the contrary, when the senses get stimulated all together, the statue moves to the *Terrified* status. When a statue changes its status, a *newState* message is sent to the acquaintances of the statue. These messages carry out information about the new reached state. All the leaf-states of a statue, along with their description, are reported in Table I.

Description of the realized prototype

Figure 6 portrays the realized prototype. It consists in a thin wooden support upon which we arranged the lights, the speakers and a plastic case hosting: the Rainbow server running over a Raspberry device, the Arduino used to manage both the sensors and the actuators, the noise sensor and the shock sensor. The proximity sensors were instead fixed under the wooden support. A placeholder of the statue was placed over a transparent container hosting the lights. The webcam was instead attached to a metal support placed in the nearness of the equipment so far described.

The arrangement of the whole equipment takes into account how the system can be really placed in the museum. The statues, in fact, are placed over a large plexiglass base. The prototyped system was conceived to be embedded in such base with the lights, when flashing, visible from the outside of the base. The webcam was instead expected to be installed on the nearest building with respect to a statue.

In Listing 1 is reported an excerpt of the log file produced by a Mood agent during system execution. From the log it is possible to see how the agent reacts to the event coming from its senses and how it change its status accordingly.

CONCLUSIONS

This paper has presented a multi-tier approach for developing SF applications in the domain of the IoT. The approach is based on the use of the agent metaphor, statecharts and on the Rainbow multi-agent platform. Agents naturally allow the development of distributed applications in a open and dynamic environment. Statecharts are well suited to model entities having a complex and time-dependent behavior. Model complexity is dealt with modular and hierarchical constructs. The Rainbow platform has the capability of hiding the physical layer on top of which the applications are built.

Different levels of abstractions can be exploited whilst designing an application. All of this fosters modularity and separation of concerns. Both a top-down and a bottom-up development schema can be adopted. The paper furnishes some guidelines for using the provided abstract levels. The goal is to replace the management of the low-level environmental stimuli and actuations to be performed with some high-level concepts which are close to the considered application domain.

As a significant case study, the approach was exploited for implementing a prototype aiming at improving security of some artworks (statues) which are in the MAB museum located in the city of Cosenza, Italy.

Prosecution of the work is devoted to:

- making the realized prototype really working in the museum
- allowing the statues to cooperate with each other in order to discover and prevent dangerous situations
- extending the functionalities of the provided prototype by offering services for increasing usability of the museum (e.g., for making virtual tour in the museum or accessing to a virtual bulletin board) and by allowing the use of the senses of the statue on behalf of tourist needs (e.g., by using the sight for taking a selfie)
- making available a tool for the automatic generation of agent-code starting from the modeled statecharts
- enhancing the approach so as to exploit not only modeling capabilities but also making possible the analysis of the realized models (e.g., through distributed simulation (Cicirelli and Nigro, 2013)) before their final implementation.

ACKNOWLEDGMENT

This work has been partially supported by RES-NOVAE - ”Buildings, roads, networks, new virtuous targets for the Environment and Energy” project, funded by the Italian Government (PON 04a2_E).

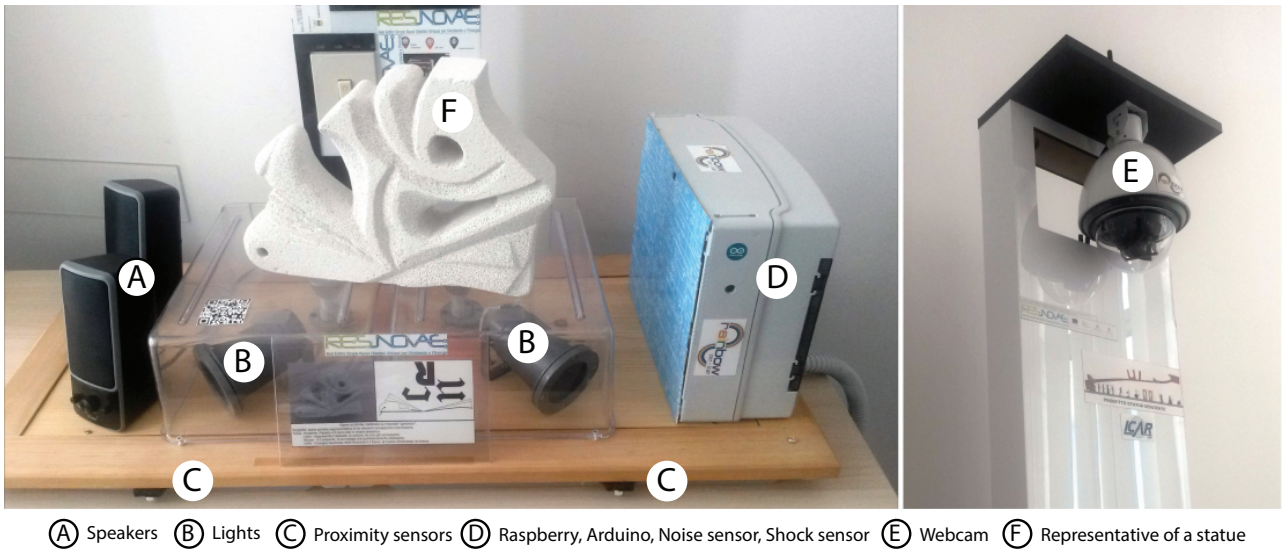

Fig. 6. Picture of the realized prototype

The authors wish to thank Christian Nigro, Alessandro Mercuri, Elisa Coscarella and Emilio Greco for their valuable collaboration in the development of the work here proposed.

REFERENCES

Atzori, L., Iera, A. and Morabito, G. (2010), The internet of things: A survey, *Computer networks* 54(15), 2787–2805.

Bicocchi, N., Fontana, D. and Zambonelli, F. (2014), A self-aware, reconfigurable architecture for context awareness, IEEE Symposium on Computers and Communications, ISCC 2014, Funchal, Madeira, Portugal, June 23-26, 2014, pp. 1–7. http://dx.doi.org/10.1109/ISCC.2014.6912485

Booch, G., Rumbaugh, J. and Jacobson, I. (2000), *The Unified Modeling Language User Guide*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Cicirelli, F., Furfaro, A. and Nigro, L. (2011), Modelling and simulation of complex manufacturing systems using statechart-based actors, *Simulation Modelling Practice and Theory* 19(2), 685–703. http://dx.doi.org/10.1016/j.simpat.2010.10.010

Cicirelli, F. and Nigro, L. (2013), *Communications in Computer and Information Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter An Agent Framework for High Performance Simulations over Multi-core Clusters, pp. 49–60.

Giordano, A., Spezzano, G. and Vinci, A. (2014), Rainbow: an intelligent platform for large-scale networked cyber-physical systems, Proceedings of the 5th International Workshop on Networks of Cooperating Objects for Smart Cities (UBICITEC 2014) co-located with CPSWeek 2014, Berlin, Germany, Apr 14, 2014., pp. 70–85.

Giordano, A., Spezzano, G., Vinci, A., Garofalo, G. and Piro, P. (2014), A cyber-physical system for distributed real-time control of urban drainage networks in smart cities, Internet and Distributed Computing Systems, Springer, pp. 87–98.

Harel, D. (1987), Statecharts: A visual formalism for complex systems, *Sci. Comput. Program.* 8(3), 231–274.

Karimi, K. (2013), The role of sensor fusion and remote emotive computing (rec) in the internet of things, Freescale Semiconductor. http://cache.freescale.com/files/32bit/doc/white_paper/senfeiotlfwp.pdf

Kielar, P. M., Handel, O., Biedermann, D. H. and Borrmann, A. (2014), Concurrent hierarchical finite state machines for modeling pedestrian behavioral tendencies, *Transportation Research Procedia* 2, 576 – 584. http://www.sciencedirect.com/science/article/pii/S2352146514001343

Miorandi, D., Sicari, S., De Pellegrini, F. and Chlamtac, I. (2012), Internet of Things, *Ad Hoc Netw.* 10(7), 1497–1516. http://dx.doi.org/10.1016/j.adhoc.2012.02.016

Schilit, B. N., Adams, N. and Want, R. (1994), Context-aware computing applications, Workshop on Mobile Computing System and Applications, IEEE Computer Society, pp. 85–90.

Tsai, C.-W., Lai, C.-F. and Vasilakos, A. V. (2014), Future internet of things: Open issues and challenges, *Wirel. Netw.* 20(8), 2201–2217.

Woolridge, M. and Wooldridge, M. J. (2001), *Introduction to Multi-agent Systems*, John Wiley & Sons, Inc., New York, NY, USA.

Wu, Q., Ding, G., Xu, Y., Feng, S., Du, Z., Wang, J. and Long, K. (2014), Cognitive internet of things: A new paradigm beyond connection., *IEEE Internet of Things Journal* 1(2), 129–143.