Context-aware Reactive Systems based on Runtime Semantic Models

Ester Giallonardo
Dept. of Engineering, University of Sannio
estergiallonardo@gmail.com

Francesco Poggi
Dept. of Computer Science and Engineering, University of Bologna
francesco.poggi5@unibo.it

Davide Rossi
Dept. of Computer Science and Engineering, University of Bologna
daviderossi@unibo.it

Eugenio Zimeo
Dept. of Engineering, University of Sannio
eugenio.zimeo@unisannio.it

***Abstract*— IoT, smart cities, cyber-physical systems and sensor networks are context-aware, highly dynamic and reactive systems. Their implementation should take into account the heterogeneity of their components and make easy the management of events unplanned at design time. According to these requirements, in this paper we propose an ontology-based approach to provide runtime models of the physical entities characterizing context-aware reactive systems. We extend SSN, a W3C standard ontology, to support complex reactive behaviors through the modeling of Logical Sensors and Actuators (LSA ontology); we also present a software architecture in which a knowledge base, structured coherently with this semantic model, is bound to real world entities by grounding (via web services) semantic elements to physical sensors and actuators. To validate the approach we discuss a case study related to smart buildings for cultural heritage preservation.**

***Index Terms*—Context modeling, Context-awareness, Semantic modeling, Semantic Sensor Networks, Ontologies**

I. Introduction

Internet of Things (IoT), smart cities and cyber-physical systems propose several scenarios characterized by a high level of dynamism and heterogeneity. Applications supporting these scenarios should be context-aware since this property has been widely acknowledged as an enabler for software adaptation to dynamic changes [1]. According to [2], context is the state that a system is able to access to or modify. This state is the set of variables that are possibly shared with other systems: they can be read or modified by users, devices or applications other than the one the state is referred to.

Various recent research works take the idea of using models as central artifacts to cope with dynamic aspects of ever-changing software and its environment at runtime. Szvetits et al. [3] comprehensively survey these approaches for adaptive context-aware systems highlighting the common idea of establishing semantic relationships between executed applications and runtime models.

In this paper, we focus on the design and implementation of reactive context-aware systems taking into account the heterogeneity of the physical devices they consist of and the ability to infer high-level context properties from directly measurable ones. Such systems need both (i) a way to describe the systems and their environments (i.e. to express architectural and state information), and (ii) a mechanism to define the application logic that drives their behaviors.

To meet the first requirement, we exploited the Semantic Sensor Network (SSN) ontology, a recent W3C recommendation [4] that has been designed to describe systems composed of a densely interconnected graph of sensors and actuators along with observations and actuations they produce. To satisfy the second requirement, we propose an extension of SSN called Logical Sensors and Actuators (LSA) ontology, since SSN does not provide mechanisms helping programmers (or reasoners) to close the gap between observations and actuations for programming context-aware reactive systems.

The LSA ontology introduces two main concepts: (software) *logical sensors* and *logical actuators*. A logical sensor (resp. actuator) is a sensor (resp. actuator) that generates observations (resp. actuations) as result of software procedures executions that use other observations as inputs. Logical sensors and actuators are entities that live only in the virtual space (e.g. knowledge base) and are connected to the external world through SSN simple sensors and actuators.

The proposed semantic runtime model is supported by a software architecture centered on a knowledge base which is bound to real world entities by grounding (mainly via web services) semantic elements to physical sensors and actuators. The behavior of the system can be specified by using sensing or actuating procedures tied to logical devices provided by the semantic model. These procedures can act upon the knowledge base by generating new facts or by redefining the structural aspects of the model thanks to the declarative approach adopted.

To clarify our approach, the overall architecture and the proposed ontology, we present a detailed scenario related to a case study in the domain of smart buildings hosting cultural heritage. The example has been implemented and tested with a prototype implementation of the proposed architecture based on Jena, OWL, and SPARQL, for the knowledge base, and RESTful services, for the interaction with the physical world, currently virtualized through an emulator.

The remainder of this paper is organized as follows. Sec-

DOI reference number: 10.18293/SEKE2019-169

tion II presents the related work from both research and standardization points of view. Section III introduces the SSN ontology, identifies its limitations with reference to the definition of complex and runnable sensors/actuators behaviors and proposes an extension of SSN. Section IV sketches a general architecture for context-aware applications. Section V validates the proposed ontology extension with a prototype of the infrastructure used to run an application scenario from eCulture domain. Finally, Section VI concludes the paper and highlights future work.

II. RELATED WORK

Several papers have tried to propose approaches and technologies to easily model and handle dynamic context-aware applications especially for ubiquitous and pervasive computing. One of the first ontology-based approaches is SOUPA [5]. It is expressed in OWL and includes modular component vocabularies to represent intelligent agents, time, space, events, user profiles, actions, and policies for security and privacy. However, it does not focus on sensors/actuators and reactive systems but on smart meeting places. In [6], the authors discuss the requirements that context modelling and reasoning should meet, including the modelling of a variety of context information types and their relationships, of high-level context abstractions describing real world situations, and of uncertainty of context information, without defining an ontology.

Paper [7] surveys context awareness from an IoT perspective. IoT researchers are taking into consideration Web technologies (WoTs) to support context-driven system engineering. The goal of the WoT is to extend Web services to devices, allowing a Web client to access devices properties, to request the execution of actions or to subscribe to events representing state changes [8]. The related ontology describes how to model physical or virtual sensors and actuators with the main objective of easing the binding with devices reachable through web protocols (REST, CoAP, etc.).

A different objective is pursued by the Semantic Sensor Network (SSN) ontology [4], an Open Geospatial Consortium (OGC)/World Wide Web Consortium (W3C) standard. It is mainly focused on the SOSA (Sensor, Observation, Sample, Actuator) pattern [9] to model reactive systems. It aims at supporting the definition of simple reactive behaviors that link observations, coming from modeled sensors, with the related reactions, performed by actuators. In order to link observations to physical or virtual properties, the SOSA pattern is extended with some system-oriented features. However, SSN does not directly support complex processing inside the knowledge base than asserting facts due to external sensing activities.

The Semantic Smart Sensor Network (S3N) ontology [10] is an effort that tries to specialize SSN for supporting the modeling of smart sensors. To this end a new class, s3n:SmartSensor, has been introduced as a specialization of ssn:System. A smart sensor is composed of embedded sensors, microcontrollers and communicating systems. The behavior is expressed by the execution of an algorithm (selected among the existing ones on context basis) by the microcontroller, which can be thought as a specialization of the ssn:Actuator, being able to select algorithms from the current context and to change the state of the whole smart sensor. Therefore, the main purpose of S3N is to support smart sensors modeling and not to close the logical gap between sensors and actuators for fully programming reactive systems.

III. SEMANTIC MODELING OF LOGICAL SENSORS AND ACTUATORS

Semantic Sensor Network ontology: the SSN ontology was specifically designed for supporting interoperability between WoT entities taking into account performance and composition requirements. Web developers, in fact, have their concern about semantic approaches that do not assure near real time data processing. For this reason, its core module is constituted by the lightweight SOSA ontology that defines concepts and properties through schema.org annotations. The SSN main perspective is the system one.

Systems of sensors and/or actuators can be deployed on platforms for particular purposes. Actuators determine changes of the state of the world through the execution of procedures triggered by the observations of properties. SSN does not fix restrictions on the way to implement procedures, allowing to describe any information that is provided to a procedure for its use (ssn:Input), and any information that is reported from a procedure (ssn:Output). Finally, sensors detect stimuli that originated observations, i.e. events that assign results to observable properties. Stimuli can be proxies for observations of properties related to features of interest. For example, infrared sensors respond to thermal stimuli detected from the environment. The thermal stimulus is a proxy for a live presence in the sensor zone, which represents the observable property related to a feature of interest.

SSN does not allow the definition of software procedures that implement machine actionable system behaviors. To overcome this limitation, we extended SSN introducing software procedures that we mainly exploit with logical sensors and actuators, which are active, composable system components able to generate observations by processing one or more observations asserted into the knowledge base and to generate actuations in a similar way, i.e. by running actionable behaviors tied to software procedures.

Logical Sensors and Actuators ontology: Fig. 1 shows a Graffoo [11] diagram of the core elements of the Logical Sensors and Actuators (LSA) ontology[2], an extension of the SOSA core of the SSN ontology that allows to describe logical sensors and actuators with a specification of their behavior.

Logical sensors and actuators are modeled with the classes lsa:LogicalSensor and lsa:LogicalActuator, which are subclasses of sosa:Sensor and sosa:Actuator, respectively. The behaviors associated to logical sensors/actuators are represented by the lsa:SoftwareProcedure class, and the property ssn:implementedBy is used to connect software procedures to sensors/actuators and consequently to the logical ones.

[2]The Logical Sensor and Actuator ontology is available at https://sites.google.com/site/logicalsensorsactuators

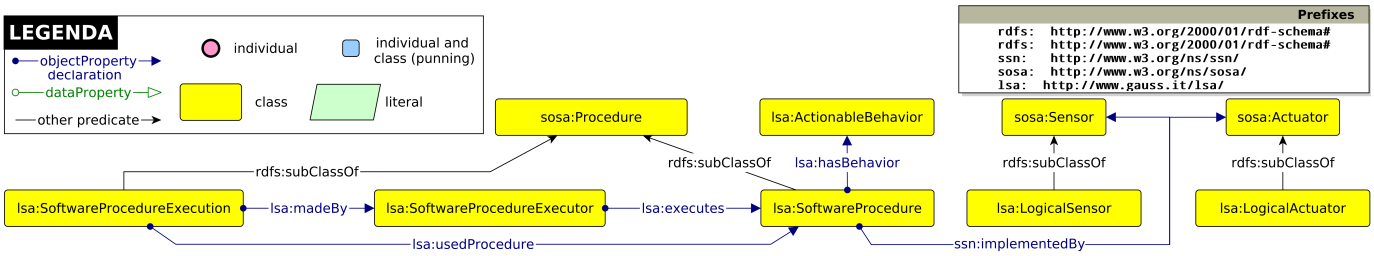

Fig. 1. Core classes of the Logical Sensors and Actuators (LSA) ontology.

A sosa:Procedure is defined in SSN as "a workflow, protocol, plan, algorithm, or computational method specifying how sensors make observations, or actuators make changes to the state of the world". A lsa:SoftwareProcedure is a specific kind of sosa:Procedure with an actionable behaviour. Software Procedures may be implemented by Sensors (Actuators) or by Logical Sensors (Actuators). Sensors (Actuators) exploit software procedures for exposing how clients may interact with physical Sensors (Actuators). Through Software Procedures, a System may expose/manage its internal states or trigger an internal process. A lsa:SoftwareProcedure behavior is described by executable code (lsa:hasBehavior property).

It is important to note that the LSA ontology does not impose constraints on how such behaviors should be represented. For example, they can be modeled as OWL-S [12] processes, BPMN processes described using the BPMN Ontology [13], etc. Another key point of the LSA ontology is that it allows to discern between:

- **procedures specifications**: the algorithm, workflow, protocol, etc. used by a sensor (actuator) to perform observations (actuations), along with a declaration of inputs and outputs. E.g. the algorithm used by a logical sensor that measures the perceived humidity (output) by aggregating a temperature and a humidity (input);
- **procedures executions**: the description of a specific execution of a procedure made by a sensor (actuator), which is carried out using a specific set of input values to produce a specific output. E.g. the perceived temperature X (output) of a room computed by using temperature Y and humidity Z as inputs.

In our pattern (which we aim at aligning with the ontology proposed in [14]) a procedure execution is modeled with the lsa:SoftwareProcedureExecution class. It is related (via the lsa:usedProcedure property) to a lsa:SoftwareProcedure and is perfomed (lsa:madeBy property) by lsa:SoftwareProcedureExecutor, a software agent able to execute (lsa:executes) a lsa:SoftwareProcedure that specifies the actionable behaviour (e.g. algorithm, workflow, protocol, etc.) manifested by the execution.

IV. AN ARCHITECTURE FOR SEMANTIC CONTEXT-AWARE REACTIVE SYSTEMS

We propose a reference architecture to design a framework able to host and exploit the semantic model described before for executing context-aware reactive applications. The main component of this architecture (see Fig. 2) is the Semantic Engine. It extends a knowledge base with the machinery needed to interact with sensors and actuators and execute their software procedures. The knowledge base contains a model of the physical world it interacts with that is enriched and modified with the data coming from the sensors, assuring consistency with the physical elements it represents. This alignment is usually referred to as *causal connection*. When a modification of the model causes the enactment of an actuator to materialize this modification in the physical world we say that the model is *bi-causally connected* [15], a feature that is supported by our architecture.

To exemplify these concepts just think about a simple reactive system immersed in an environment composed by a room with a light bulb, a bulb actuator and a light sensor, all these elements are represented in a virtualized form within the system. In a causally connected system the change of the state of the real-world light bulb (turned on/turned off) is reflected in the model element that represents the bulb within the system. In a bi-causally connected system, the modification of the state of a model element is reflected as a change of state of its real-world counterpart. Thus, if we set the state of the model element representing the light bulb to off while the real-world light bulb is turned on, this triggers an actuator to turn off the bulb.

The key ingredients to actualize a system of this type are: one or more models that describe real-world conceptual classes, a binding mechanism that maps sensor observations to knowledge base updates, logical causal connections that propagate updates throughout the knowledge base, and a binding mechanism that maps updates to actuators activation for preserving the model alignment with real-world situations. It is worth noting that causal connections need some kind of computational support. According to the organization above, our architecture presents: (i) a semantic model built using the previously introduced ontologies hosted by a knowledge base platform (a triple store); (ii) a linking mechanism to report sensor readings to the system, implemented using web services exposed by the system, which is responsible of converting readings into semantic triples to insert into the triple store; (iii) a programmed logic for generating new facts from observations; (iv) an actuation mechanism exploiting actuators Web services, consistently with the WoT approach.

Causal connections are supported by rules that correlate real

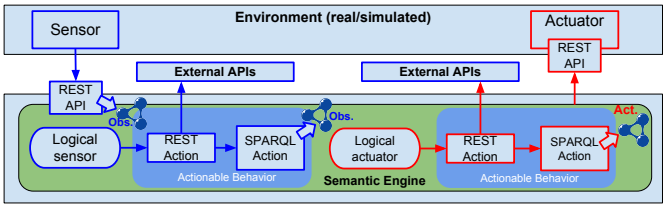

Fig. 2. Architecture outline.

world changes observed by sensors with knowledge base updates. We consistently represent these rules in the knowledge base itself: the activation part is modeled as software procedures associated to semantic sensors and actuators whereas the triggering logic is implemented by monitoring changes to the properties that are declared as inputs for these semantic sensors and actuators.

The engine (see Fig. 2) connects to the physical world by exposing a service API used to receive observations from external sensors (that can be real or simulated ones) and by invoking web service endpoints for activating external actuators or for invoking external services (on the right) to increment the capabilities of the software procedures associated to logical sensors and actuators. Whenever an external sensor notifies an observation invoking the engine's API, that observation is transformed in a semantic format and added to the knowledge base. If a logical sensor/actuator is interested in that observation (which means that it is modeled in such a way that its software procedure uses as one of its inputs the property reported by the observation) its related software procedure is executed (by running the actionable items that define the specific Actionable Behavior), producing new facts (observations or actuations) that could trigger external actuators. This approach allows for declarative definitions of reactive behaviors in a bi-causally connected system. In fact, both the model of the context and that of the system (in terms of logical sensors/actuators and their behaviors) is represented in a semantic format (e.g. by RDF triples). This allows to change the overall behavior of the system by manipulating the knowledge base: at runtime new logical sensors can be defined, the behavior of the existing ones can be modified, existing sensors/actuators can be deleted. A further advantage of this architecture is that self-adaptive behaviors can easily be implemented by simply allowing the software procedure of a sensor/actuator to work as described in [16], [17]. For example a software procedure can be activated by the detection of a failure in an external sensor to compose observations produced by other sensors in order to collect the expected events related to a feature of interest.

V. A CASE STUDY

We consider a running example derived from a larger system for Cultural Heritage preservation [18]. In a museum a new temporary exhibition is arranged. In a room of this exhibition a multimedia content has to be played. The organizers of the exhibition express the desire that the content starts playing when visitors enter the room, and stops when the room is empty. Museum rooms have no specific detectors for knowing the number of people inside them but are equipped with Bluetooth beacons (one per room) and Infrared (IR) sensors close to the doors (used as part of the anti-theft system). Beacons notify their presence to users' personal devices equipped with a specific App turning these devices into location detectors.

We can define a logical sensor for observing the number of people in a room with two different implementations: one based on beacons and personal devices and another one based on IR sensors. If the former is faulty, the declarative approach eases the selection of another (logical) sensor able to observe the same property. In the following, we assume the first is faulty and describe the IR-based logical sensor for clarifying the overall approach and the LSA ontology.

Multimedia playback control based on a logical presence sensor:

1. a tourist crosses the door of the museum, and the two physical infrared sensors on the door sides produce two observations about the presence of a person in their detection areas;
2. a logical sensor aggregating such observations produces another observation updating the number of persons present in the rooms;
3. if the tourist enters an empty room, an actuator starts to play a multimedia flow on the room monitor; if the tourist is the last person that leaves a room before the end of the playback, an actuator will stop the multimedia flow. In both cases, the information about the new actuation is inserted into the triple store.

1. Observations made by physical sensors: Fig. 3 shows the RDF statements that are added to the triplestore by the semantic engine when a person crosses a door. Whenever this occurs, the infrared sensors placed on the two sides of the door detects the presence of a person and invokes the engine REST API in sequence (providing their ids and the instants of time when the observations occurred as request parameters).

Two observations (i.e. gmus:observation/ir1/1 and gmus:observation/ir2/1) made by sensors gmus:ir1 and gmus:ir2 are produced, which relate to the same feature of interest (i.e. gmus:door1). Each observation concerns a distinct observable property (i.e. the presence in the detection area of each sensor: gmus:presence/room1/ir1/zoneDoorInside and gmus:presence/room2/ir2/zoneDoorOutside), and keeps track of the time in which the observations were performed.

In these examples we make use of punning[3], an OWL metamodeling capability that allows to treat model elements as classes and individual as the same time. Elements with this double nature are represented as light blue squares in the diagram. This has been used in Fig. 3, for instance, to model the concept of infrared sensor (gmus:IRSensor), which is at the same time a class (i.e. a specific subclass of sensors representing infrared sensors) and an individual (since it is connected with

[3]See https://www.w3.org/TR/owl2-new-features/#F12:_Punning

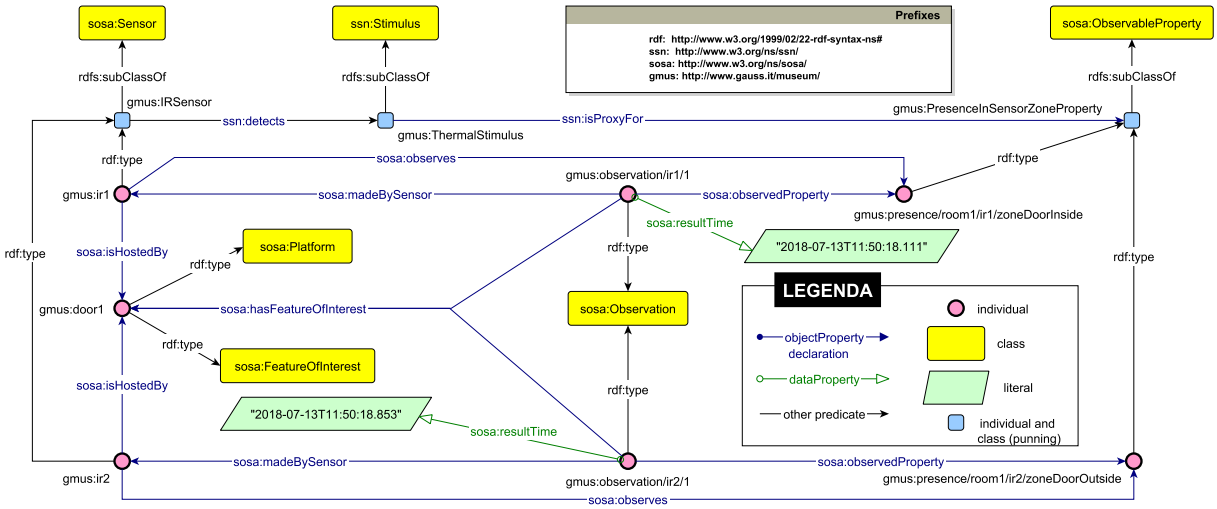

Fig. 3. Observations made by two infrared sensors.

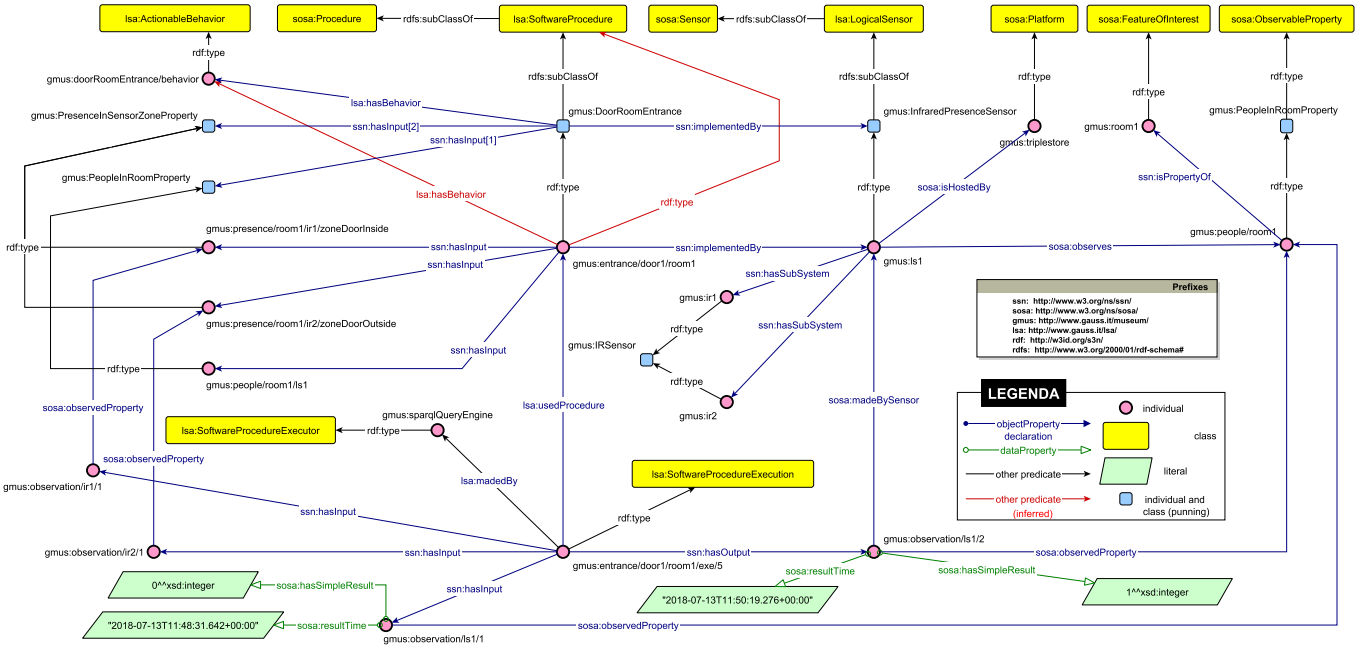

Fig. 4. Observations made by the logical presence sensor. Square brackets are used to specify property cardinality restrictions.

gmus:ThermalStimulus by the ssn:detects property). In the same way, gmus:PresenceInSensorZoneProperty is a type of observable property (i.e. subclass of sosa:ObservableProperty) and an individual (connected to gmus:ThermalStimulus by the ssn:isProxyFor property). This approach is also useful to model logical sensors behaviors, as described in the rest of this section.

2. Observations made by logical sensors: whenever a modification occurs in the triplestore (e.g. the insertion of a new observation), the semantic engine checks if one or more procedures specifying the behaviors of logical components (i.e. logical sensors and actuators) should be executed. To do so, the engine checks if the properties related to the new observations (e.g. gmus:presence/room1/ir1/zoneDoorInside and gmus:presence/room2/ir2/zoneDoorOutside in the previous example) are specified as inputs of one or more software procedures. Since these properties (see Fig. 4) are inputs of the gmus:entrance/door1/room1 procedure (as specified by ssn:hasInput), the semantic engine identifies the procedure, which is tied to the logical sensor gmus:ls1, a specific instance of gmus:infraredPresenceSensor (the class representing logical presence sensors) hosted by the triplestore (gmus:triplestore), and executes it by observing the presence of people in the specific room (gmus:people/room1).

A mechanism is adopted by the semantic engine to retrieve behavioral information (e.g. a sequence of activities to perform) pertaining logical sensors. Since behavioral information is shared by all logical sensors of a type, the engine identifies the related software procedure (gmus:DoorRoomEntrance in our case) and retrieves the behavioral specification (gmus:doorRoomEntrance/behavior) by navigating the lsa:hasBehavior property.

Such behavioral specification in this case is composed of two actions, i.e. two SPARQL CONSTRUCT queries checking the entrance/exit in/from the room, respectively. Each of these queries retrieve the new observations made by the two infrared sensors, and if they have been performed in a short time interval - e.g. one second - produces:

1) a new software procedure execution (gmus:entrance/door1/room1/exe/5), connected to the software procedure (gmus:entrance/door1/room1/) by the lsa:usedProcedure property, and to the observations used as input (those made by the two infrared sensors and those pertaining the number of persons in the rooms connected by the door[4]) and the software procedure executor (gmus:sparqlQueryEngine) by the ssn:hasInput and lsa:madeBy property, respectively;
2) two observations as output of the procedure execution represented using the ssn:hasOutput property. For instance, the number of people in the first room has been updated from zero (in gmus:observation/ls1/1) to one (in gmus:observation/ls1/2) since a person entered the room.

3. Actuations made by logical actuators: the newly added statements (i.e. those about the observations produced by the logical sensor gmus:ls1 and the relative procedure executions) trigger another control performed by the semantic engine to check logical sensors/actuators interested to those observations. In our example, the logical actuators controlling the video playback on the monitor in the room is activated, and the related software procedures is retrieved and executed, triggering the final actuation (REST invocation) of the physical device that starts the video playback on the monitor.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a proposal for an extension of the SSN ontology to support modeling of logical sensors and actuators, and their behaviors. The extension enables reactive behaviors of context-aware applications by defining the decision logic that exploits sensor observations to trigger actions. The ontology is accompanied by a an architecture that supports behaviors definition and the interaction with the real devices in the physical world. A prototype of the architecture has been implemented by using Jena, SPARQL and RESTful APIs for the interaction with the external environment, currently emulated with Freedomotic. We discussed and validated the proposed ontology extension and the supporting architecture with the help of a case study in the domain of smart buildings for cultural heritage. The ontology extension and the related architecture represent the first step towards the definition of a more complex platform for context-awareness able to take into account failures and adaptation policies.

[4]Because of space limitations in the diagram we depicted only the observations about a room (i.e. we omitted the observations about the number of people in gmus:room2)

ACKNOWLEDGMENTS

This paper has been supported by MIUR PRIN 2015 GAUSS Project and MIUR PON VASARI Project.

REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *International symposium on handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.
[2] A. Furno and E. Zimeo, "Context-aware composition of semantic web services," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 235–248, 2014.
[3] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Software & Systems Modeling*, vol. 15, no. 1, pp. 31–69, 2016.
[4] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic sensor network ontology," *W3C Recommendation, W3C*, 2017.
[5] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications," in *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* IEEE, 2004, pp. 258–267.
[6] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.
[7] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
[8] S. Kaebisch and T. Kamiya, "Web of things (wot) thing description," *First Public Working Draft, W3C*, 2017.
[9] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "Sosa: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, 2018.
[10] S. Sagar, M. Lefrançois, I. Rebai, M. Khemaja, S. Garlatti, J. Feki, and L. Médini, "Modeling smart sensors on top of sosa/ssn and wot td with the semantic smart sensor network (s3n) modular ontology."
[11] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali, "Modelling owl ontologies with graffoo," in *European Semantic Web Conference*. Springer, 2014, pp. 320–325.
[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne *et al.*, "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, no. 4, 2004.
[13] M. Rospocher, C. Ghidini, and L. Serafini, "An ontology for the business process modelling notation." in *FOIS*, 2014, pp. 133–146.
[14] M. Lefrançois, "Planned etsi saref extensions based on the w3c&ogc sosa/ssn-compatible seas ontology paaerns," in *Workshop on Semantic Interoperability and Standardization in the IoT, SIS-IoT*, 2017, p. 11p.
[15] M. Hölzl and T. Gabor, "Reasoning and learning for awareness and adaptation," in *Software Engineering for Collective Autonomic Systems*. Springer, 2015, pp. 249–290.
[16] F. Poggi, D. Rossi, P. Ciancarini, and L. Bompani, "Semantic runtime models for self-adaptive systems: a case study," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*. IEEE, 2016, pp. 50–55.
[17] F. Poggi, D. Rossi, and P. Ciancarini, "Integrating semantic run-time models for adaptive software systems," *To appear in Journal of Web Engineering*, 2019.
[18] E. Giallonardo, C. Sorrentino, and E. Zimeo, "Querying a complex web-based kb for cultural heritage preservation," in *Knowledge Engineering and Applications (ICKEA), 2017 2nd International Conference on*. IEEE, 2017, pp. 183–188.