Modeling and Simulation of Web-of-Things Systems as Multi-Agent Systems

Ion Mircea Diaconescu[1(✉)] and Gerd Wagner[1,2]

[1] Brandenburg University of Technology, Cottbus, Germany
{M.Diaconescu,G.Wagner}@b-tu.de
[2] Old Dominion University, Norfolk, USA

Abstract. In the Web of Things (WoT), special communication networks composed of sensor nodes, actuator nodes and service nodes form the basis for new types of web application systems, which are directly connected to the real world via sensors and actuators. We propose a conceptual framework for simulating WoT systems as multi-agent systems where both sensor nodes, actuator nodes and service nodes, as well as other systems in their environment interacting with them (such as other web applications, web services and human users), are modeled and simulated as agents. Our conceptual framework includes an ontology of WoT systems as sensor/actuator systems, and a meta-model for defining an agent-based WoT system simulation language.

Keywords: Web of things · Sensor-actuator systems · Agent-based modeling · Simulation

1 Introduction

In the Web of Things (WoT), special communication networks composed of sensor nodes, actuator nodes and service nodes form the basis for new types of web applications, which are directly connected to the real world via sensors and actuators, and can be private, such as smart home apps, personal robotics apps and factory control applications, or public, such as air pollution monitoring systems and city parking management apps. We propose a conceptual framework for simulating WoT systems as multi-agent systems where both sensor nodes, actuator nodes and service nodes, as well as other systems in their environment interacting with them (such as other web applications, web services and human users), are modeled and simulated as agents.

We consider systems of purposeful interacting systems (with some degree of autonomy) as *multi-agent systems*. Our conceptual framework includes an ontology of WoT systems (WoTS) as sensor/actuator systems, and a meta-model for defining an agent-based WoT system simulation language. Our approach supports "hardware in the loop", "software in the loop" and "humans in the loop", where "hardware" refers to sensor or actuator nodes, "software" refers to web applications and web services, and "humans" refers to human users, which can

J.P. Müller et al. (Eds.): MATES 2015, LNAI 9433, pp. 137–153, 2015.
DOI: 10.1007/978-3-319-27343-3_8

interact with a WoTS simulation via the user interface of a web application or via a user interface of a sensor or actuator node.

Our proposed simulation framework is not concerned with low-level networking issues in WoTS, e.g., resulting from specific network topologies. Rather, the goal is to develop a WoTS simulation approach including and above the application layer, with a focus on simulating the sensor, actuator and service nodes as parts of a WoTS, and the WoTS as a whole.

2 Related Work

In [4], an approach to agent-based simulation of a network of web-service-enabled devices is proposed. The authors argue that the *Service-Oriented Architecture* (SOA) paradigm can be used for achieving interoperability between the nodes of a WoT system and between such a system and modern enterprise networks. When devices expose their data and operations as web services, this provides an integration of devices with enterprise applications, allowing new innovative solutions to enterprise automation problems. The authors choose the open standard protocol *Devices Profile for Web Services* (DPWS) as the interaction protocol for web-service-enabled devices [5]. However, this protocol is based on a protocol stack that is too complex for constrained resource devices as needed, for instance, in battery-operated WoT networks, while it may work well in non-constrained office or factory environments. A Java-based multi-agent platform is used to create the agents that simulate DPWS devices. These agents are connected to the enterprise network in the same way as real DPWS devices. The resulting system can be used for evaluating the impact of a large number of networked devices on the running enterprise applications, without the need to set up a real device network, which would be quite expensive and more difficult.

In [6] is proposed a decentralized architecture where agents providing atomic system services run inside IoT devices. The authors argue that it is possible, by using their approach, to dynamically distribute system load and move the data processing tasks into the IoT devices in the edges of the network. HTTP and CoAP (for low resource devices) are used as communication protocols, and a proxy translates messages between these protocols. The assumption is that IoT devices exist in a physical form (they are not software simulated), and the discussion is rather generic, not making clear how such a system, composed of various IoT devices (software and hardware platforms) can be practically implemented.

A language and platform independent composition for mobile agents-based smart objects is proposed in [7]. Using this approach, communication and data sharing between agents is possible over disparate networks. The information infrastructure is realized with the IETF CoRE framework [8]. The REST principles are utilized in agent creation, migration, control, smart object communication and resources exposal to the internet. The authors discuss a system reference architecture as well as an application programming interface which allows basic HTTP and CoAP communication, content negotiation and authorization methods. Some of the principles presented in this work are also used in

the implementation of our JavaScript simulation framework, allowing agents to share information among heterogeneous networks and provide access to resources with the help of web interfaces.

3 IoT and WoT Systems

An *Internet-of-Things (IoT) system* is a communication network consisting of sensor nodes, actuator nodes and service nodes, such that at least one node is connected to the Internet. A sensor node consists of a controller to which one or more sensors and a communication unit are attached. An actuator node consists of a controller to which one or more actuators, zero or more sensors and a communication unit are attached.

A *WoT system (WoTS)* is an IoT system that is built with Web technologies. These technologies do not only include the classical Web technologies HTTP(S), HTML, CSS and JavaScript, but also the more recent Web technologies *Server-Sent Events*, *Web Sockets*, and the *Constrained Application Protocol (CoAP)* proposed in [9]. We distinguish between the following three cases:

1. WoT systems that do not have the limitations implied by constrained resource devices. These systems can use ordinary networking and web technologies such as IEEE 802.11 for wireless networking, HTTPS and SOAP for application-level messaging, and SOAP-based co-ordination and security techniques, as proposed in [4].
2. WoT systems based on constrained resource devices having unlimited power supply (not using batteries), such that power consumption is not a concern. These systems need an alternative software/technology stack that is adapted to the limited main memory, storage and processor speed of the constrained resource devices. Ethernet (or IEEE 802.11) can still be used for (wireless) networking, but only CoAP or an HTTP subset, and no HTTPS, can be used for application-level messaging.
3. WoT systems based on constrained resource devices that are battery-powered, requiring low-energy wireless networking technologies, such as *IEEE 802.15.4*, and small footprint software technologies, such as CoAP for application-level messaging. These systems often have higher packet error rates and a lower throughput (say, of only tens of kbit/s).

Unlike many other authors, such as [2,4], we consider the issue of using the new Internet Protocol (IP) version 6 (IPv6) instead of the established version 4 (IPv4) as orthogonal to the WoT. The main issue solved by IPv6, allowing a greater address space than IPv4, is not necessarily an issue for WoT systems, which can, in many cases, be built with either of them. Of course, the increasing use of IoT apps will contribute to the increasing demand for IP addresses. But since most IoT/WoT devices will not have to be reachable via an IP address, the expected explosive growth of the IoT/WoT will not imply a similar explosion of the IP address space.

The following are considered to be desirable features of a WoTS:

- self-configuration: the dynamic composition of WoTS by nodes joining and leaving the network at any time
- self-diagnosis: automatic discovery of failures and faults
- self-optimization of constrained energy (battery-based) WoTS: automatic monitoring and on/off-time control of resources

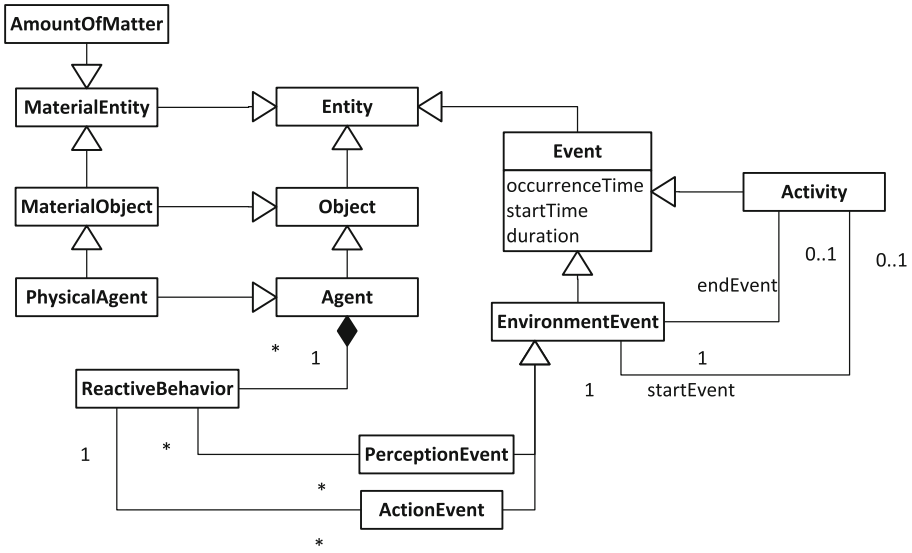

Fig. 1. A fragment of the AOES ontology.

Further, we discuss an approach for modeling sensor and actuator nodes as agents. The AOES ontology fragment shown in Fig. 1 is the basis of our proposal. `PhysicalAgent` represents the core of a WoTS model, and it consists of a set of built-in properties representing the physical characteristics of the agent (e.g., spatial position) and zero or more (built-in or custom defined) reaction rules defining its behavior. When needed, additional custom properties can be defined for specific agent types derived from `PhysicalAgent`.

3.1 Sensor Nodes

As described in the UML class diagram shown in Fig. 2, a sensor node represents a `PhysicalAgent`. It consists of a controller, a communication unit and one or more sensors. A sensor consists of one or more detectors, which represents simple (non-composite) sensors. A sensor is a measurement device that is attached to a controller within a sensor node. As shown in Fig. 3, we distinguish between three types of detectors:

1. quality detectors with an analog interface,
2. quality detectors with a digital interface,
3. event detectors.

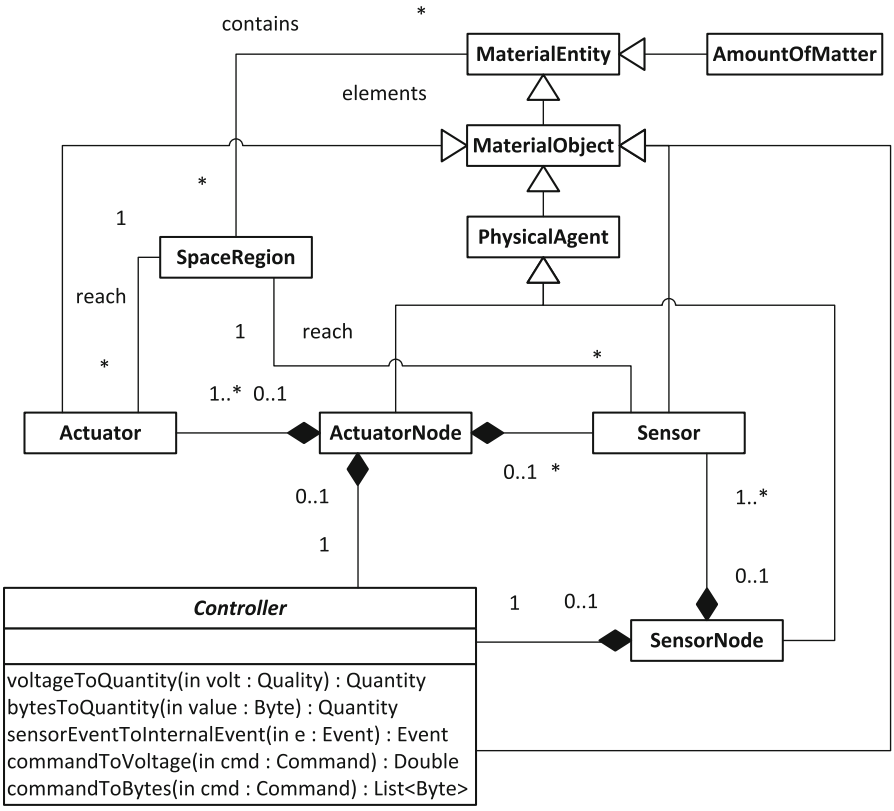

Fig. 2. An overview of the sensor and actuator nodes model.

Quality detectors and event detectors are simple (non-composite) sensors. A *quality detector* is a device that allows measuring a physical quality of its environment (or *reach*). An *event detector* allows detecting events occurring in its reach. Notice that the concept of physical qualities has been defined in the philosophical discipline of *ontology* (or *metaphysics*). A quality is an entity, and not a data value, but it can be approximately represented by a data value (namely the value of an attribute that captures the type of quality). For instance, the voltage level of a wire of a particular detector at some moment in time is a quality, which can be approximately represented by the value of an attribute `outputVoltage` used for expressing statements about, and measurements of, the detector.

Figure 3 shows how the sensing operation of an analog quality detector can be conceptualized as a transformation, which converts a quality to be measured

in the detector's reach to an internal quality of the detector device (typically, to a voltage level) that can be read and transformed to a measurement quantity by the controller. In our WoTS sensor ontology, we call the first transformation function *quality-to-voltage*, and the second one *voltage-to-quantity*.

The sensing operation of a digital quality detector can be conceptualized as a transformation, which converts a quality to be measured in the detector's reach to a sequence of bytes that can be read and transformed to a measurement quantity by the controller (see Fig. 3). In our WoTS sensor ontology, we call the first transformation function *quality-to-bytes*, and the second one *bytes-to-quantity*.

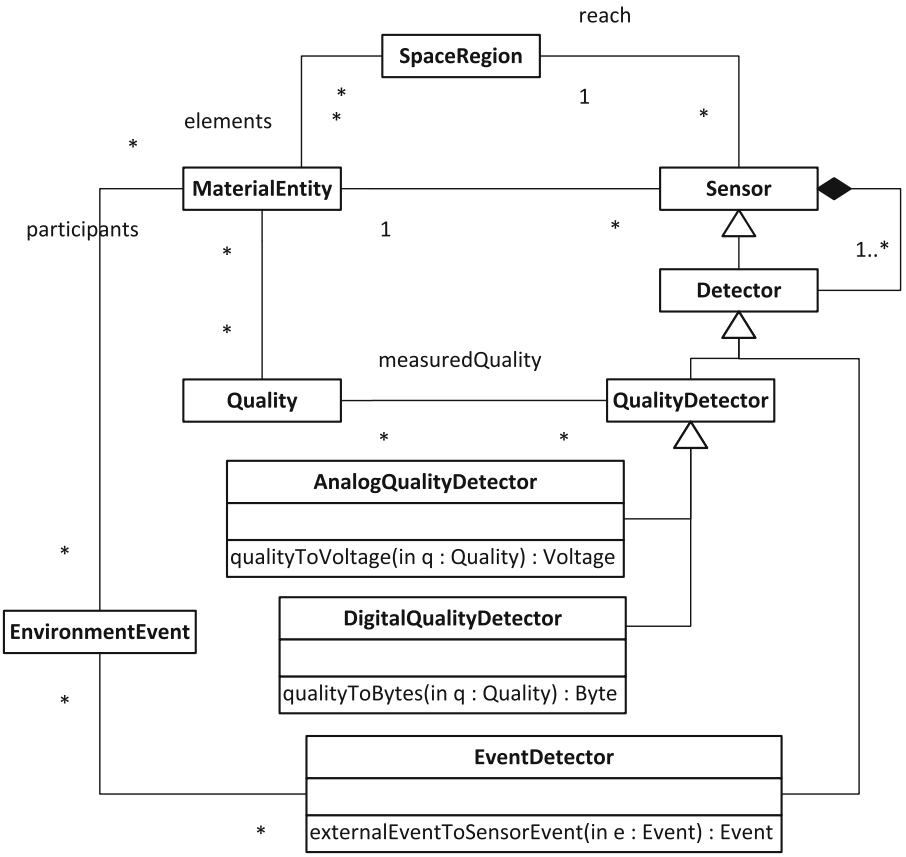

Fig. 3. An overview of the sensor model.

The sensing operation of an event sensor can be conceptualized as a transformation, which turns the occurrence of an event of a certain type in its reach to an internal event of the sensor device itself (typically corresponding to digital voltage signals) that can be detected and transformed to a control event by the controller. In our WoTS sensor ontology, we call the first transformation *externalEvent-to-sensorEvent*, and the second one *sensorEvent-to-internalEvent*.

3.2 Actuator Nodes

As shown in the UML class diagram in Fig. 2, an actuator node represents a `PhysicalAgent`. It consists of a controller, a communication unit, one or more actuators and zero or more sensors.

As a component of an actuator node, an actuator is an enactment device that is attached to a controller within an actuator node. Common types of actuators are electro-mechanical devices that are controlled with the help of a (voltage or digital interface) signal. Examples are motors, water pumps and relays.

Based on the interaction with the environment, we distinguish between two types of actuators: *event actuators* and *activity actuators*.

As shown in Fig. 4, the effect of an *event actuator* is an event which directly or indirectly affects qualities of its reach. A direct effect is when the resulting event produces immediate state changes in the environment (reach) quality. For example, a relay actuator which turns on a light, produces an immediate change in the light intensity value of the actuator reach. An indirect effect is when the resulting event produces a set of actions which later can result in reach state changes.

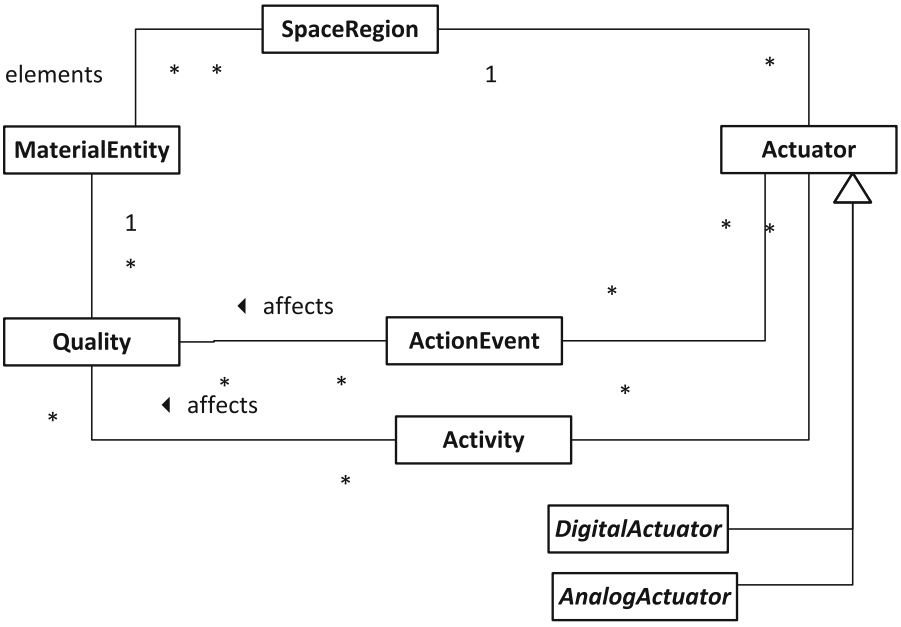

Fig. 4. An overview of the actuator model.

An *activity actuator* produces environment (reach) state changes over a time period. It has a start and an end triggering event. For example, a watering activity is started when the soil moisture is below a threshold and is ended when the soil moisture reaches the required level. During the watering activity time, the soil moisture increases as a result of the water flow, and a soil moisture sensor reads the changes of this quality.

We distinguish between two main types of actuators (see Fig. 4), with respect to their communication interface: digital and analog. Digital actuators are either controlled with a simple high/low signal or by using a specific digital communication protocol, such as I2C, SPI or UART.

Analog actuators are controlled by voltage levels, and their effect is represented by one or more functions, sometimes provided in the actuator datasheet. While for a vaste majority of actuators this information is not available, a linear, exponential or logarithmic function can be used to simulate the output of an actuator. Controlling an analog actuator requires to use a DAC (digital to analog convertor) capable controller, such as Arduino DUE [1].

Some *digital actuators* are controlled with a simple high/low signal. They can be in one of the two states: open or closed. Examples of such actuators are relays and electro-valves. Other digital actuators accept as input a set of bits which describes the command to be executed. For example, some PWM actuators expect a 64 bits encoded duty cycle and frequency values. In general, standard digital communication protocols (e.g. I2C, SPI or UART) are used to communicate with digital actuators, but for some, the datasheet may also describe a custom protocol.

3.3 The Environment

The environment (reach) of the sensors and actuators of a WoTS consists of amounts of matter (such as soil and air) and of discrete material objects (such as cars and animals). Amounts of matter and material objects bear certain physical qualities (such as color or temperature) that can be measured by quality detectors, and may participate in certain events that can be detected by event detectors. Physical qualities of the objects or amounts of matter in the environment can be affected by actuators. For example, turning on a heater affects the temperature of that specific heater actuator reach.

3.4 Examples of Sensors and Actuators

In the above sections we discuss about sensors and actuators as components of a node. A large variety of sensor and actuators, such as the ones shown in Fig. 5, exists for being used in WoT projects. For instance, an actuator node may consist of a *Grove* soil moisture analog quality detector and a *Pump* digital actuator attached to an *Arduino UNO* [1] micro-controller, plus an *ESP8266*[1] WiFi communication module. Notice that a `Pump` represents an actuator type, the instances of which are individual *Pump* actuators. In the same way, a sensor node may consist of a *DHT22*[2] (itself consisting of a temperature and humidity detectors) sensor, an *Arduino UNO* micro-controller, and an *ESP8266* WiFi communication module.

[1] ESP8266 WiFi module - http://www.esp8266.com/.

[2] DHT22 Sensor - https://www.adafruit.com/datasheets/DigitalhumidityandtemperaturesensorAM2302.pdf.

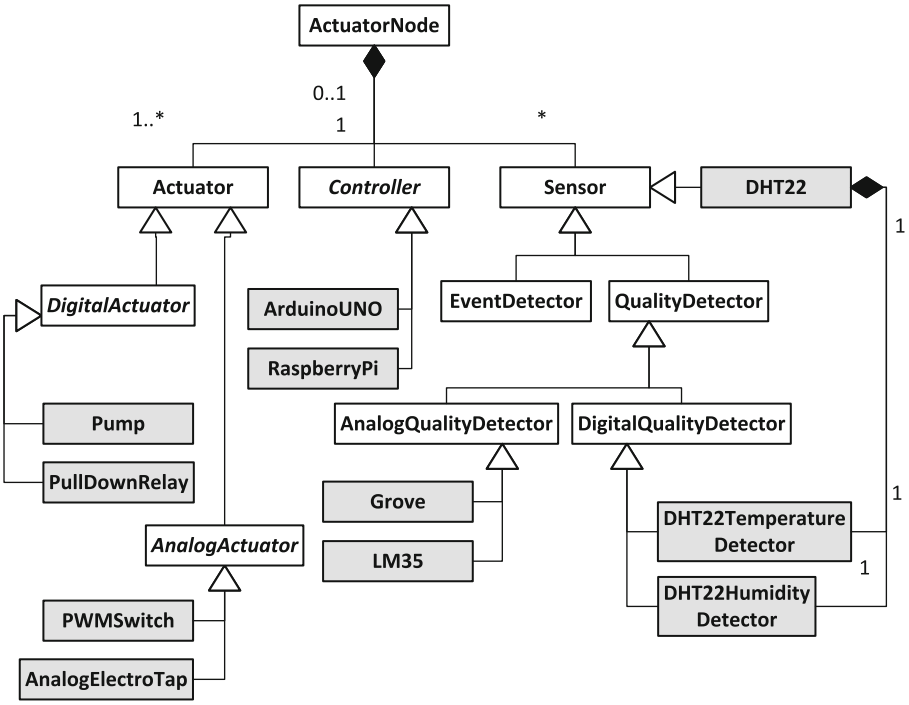

Fig. 5. Examples of sensor and actuator types.

4 The AOE Simulation Framework

Our AOES framework includes a simulation language and a simulator implementation, with support for various built-in sensor (e.g., LM35) and actuator types (e.g., PullDownRelay).

The simulation language is based on AORML [10] (Agent-Object-Relationship Modeling Language), and provides the language elements used to define WoTS simulations. In AORML, an entity is either an agent, an event, an action, a claim, a commitment, or an ordinary object. Agent and object form, respectively, the active and passive entities, while actions and events are the dynamic entities of the system model. Commitments and claims establish a special type of relationship between agents. These concepts are fundamental components of social interaction processes and can explicitly help to achieve coherent behavior when these processes are semi or fully automated. Only agents can communicate, perceive, act, make commitments and satisfy claims. Ordinary objects are passive entities with no such capabilities.

We extend the AOE Simulation Language with support for the new types required in a WoT simulation. The basic elements used to define a sensor node are shown in Fig. 6. Interfaces are used to describe operations (functionality),

such as the *attribute-to-voltage* mapping function needed for the analog quality detectors. Concrete WoT component types, such as `ArduinoUNO` (a microcontroller type) and `LM35` (an analog quality detector type) are instances of `MaterialObject`, from which physical properties are inherited (e.g., spatial position and sizes). A set of parameters are used when a new component type is created, one of the most important being `supertype` which allows to classify the new component (e.g., as a `DigitalActuator` or `AnalogQualityDetector`). This simulation language is used by our JavaScript simulator implementation prototype. For instance, a new analog detector type, such as `LM35`, is defined as follows:

```
var LM35 = new MaterialObject( {
  typename: "LM35",
  supertype: "AnalogQualityDetector",
  mappingFunction: {
    method: "linear",
    initialData: { q0: 0, v0: 0, q1: 100, v1: 1},
  },
  measurementRange: { min: 0, max: 100},
  accuracy: 0.5,
  precision: 1,
  resolution: 0.1,
  supplyVoltage: 5,
  properties: {
    "outputVoltage": { range: "Decimal"}
  }
});
```

Notice that in the class diagram of Fig. 6, the `ArduinoUNO` class represents, like a product type, a controller type, the instances of which are individual *ArduinoUNO* micro-controllers. Likewise, the `LM35` class represents a detector type, the instances of which are individual *LM35* detectors.

5 Modeling and Simulation of WoT Systems

A *WoTS simulation* consists of one or more simulated WoTS nodes, an environment simulator, and zero or more real WoTS nodes, satisfying certain conditions as defined below. The environment simulator is in charge of managing the state of the simulated environment, including the reaches of all simulated sensor and actuator nodes, and of simulating environment events, which may change the state of the simulated environment, e.g., by changing certain property values in certain simulated sensor reaches, and may affect event detectors if a simulated environment event of the right type occurs in the reach of a simulated event detector.

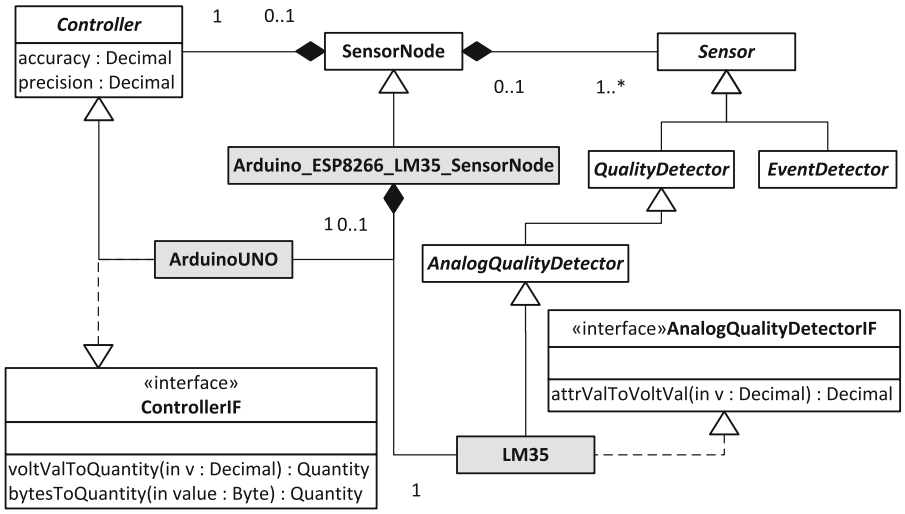

Fig. 6. AOES sensor node example.

5.1 Simulation of Sensor Nodes

A simulated sensor node consists of a controller to which one or more simulated sensors are attached. Since our simulation framework is not concerned with low-level communication issues, we do not include the communication unit as an explicit component of a simulated sensor node. A detailed discussion about how a simulated sensor reads a quality of the environment and produces a result representing the measured quantity was presented in [3]. We provide a summary of the mapping functions used to simulate the sensor output.

The Attribute-Value-to-Voltage-Value Function. In a measurement simulation, the value of a reach attribute is read by the simulated analog quality detector and transformed to a voltage value. The sensor resolution is used to compute the "detected" input value, then, using a mapping function, a voltage output value is obtained. The mapping function is either provided in the sensor datasheet, or approximated by a linear, logarithmic or exponential function, as shown in [3]. Considering the sensor characteristics, such as accuracy and precision, the final voltage output of the sensor is computed.

The Voltage-Value-to-Quantity Mapping Function. This function is used to provide a quantity for the measured quality when analog detectors are used. Using ADC (analog to digital converted) enabled controllers (such as Arduino UNO), the sensor analog output is mapped to a value representing the measured quantity of the environment quality. The ADC unit has accuracy and precision, like a sensor, since it is a voltage sensor with a digital output, which are reflected in the ADC output.

Attribute-Value-to-Quantity Mapping Function. This applies to sensors with digital output, no matter which communication protocol they use, since the final result is a decimal number. When the sensor datasheet specifies a particular mapping function, then it can be used in the sensor simulation. However, in many cases, this information is not available, the strategy being to use the same mapping functions as in the case of the attribute-value-to-voltage-value function, with the difference that the output is not a voltage value but a value which is the quantity of the measured quality. The sensors with digital output have also accuracy, precision and resolution factors, which are handled as detailed in [3].

5.2 Simulation of Actuator Nodes

A simulated actuator node consists of a controller to which one or more simulated actuators and zero or more simulated sensors are attached. As for the case of simulated sensor nodes, we do not include the communication unit as an explicit component of a simulated actuator node.

As shown in Fig. 4, an actuator affects the reach either by triggering events with direct or indirect effect, as discussed earlier in this paper, or by producing an activity. For the simulation of the environment state changes produced by actuators, *uniform* and *radial* mapping functions are used. Whenever needed, custom mapping functions may also be implemented, e.g., when a simulation scenario has custom requirements.

The Unform Mapping Function. This mapping function is used whenever a quality of the reach is affected in a uniform manner in the environment (or reach), no matter the physical coordinates. For example, a relay which turns on a set of lights may produce uniform light (considering an ideal environment) over the reach.

The Radial Mapping Function. Using this mapping type, the effects of the actuator are reduced (or amplified) with the increase of the distance between the actuator spatial position and the coordinates within the affected reach. For example, a water pump actuator produces a radial effect with respect to the soil moisture environment quality. In other words, the soil moisture level decreases along with the increase of the distance between the water pump, considered the origin of the water flow, and the spatial coordinates in the reach where the soil moisture quality measurement is performed. Concrete formulas used to compute the radial effects depends on the affected physical properties. One example is provided at the end of this paper, when a “Green House” simulation scenario is discussed.

5.3 Simulation of the Environment

The simulated environment (simulation world) consists of simulated actuator and sensor reaches, which overlap partially or totally. The physical space of

the simulated environment is divided in cells, which represent the atomic space units. A simulated reach is composed of a set of such neighbor cells. In a real world WoTS, a sensor or actuator can have a irregular physically shaped reach, i.e., a mesh shape. For simplicity reasons, in the case of simulated sensors and actuators, the corresponding reach spaces are described by using regular shapes, such as cuboids or approximated spheres. A simulated actuator actions have effects only on that specific actuator reach, while a simulated sensor is able to read (sense) only from its reach.

Actuator actions create environment events which activates environment reaction rules. As result, state changes of the physical qualities of the objects or matter within the simulated actuator reach may occur or activities are started or ended, as shown in Fig. 4. Simulated quality detectors (or sensors) detect quality state changes, while simulated event detectors (or sensors) receive perception events. For example, a simulated *LM35* analog quality temperature detector senses changes of the temperature quality within its reach, and a *PIR* (passive infrared sensor) detects the presence of an infrared emitting object within its reach.

For simulations with sensor hardware in the loop, it is important to notice that while simulated sensors have a simulated reach, real sensors are situated in a real-world environment. A simulated environment does not affect any real sensor node and a real-world environment does not affect any simulated sensor node, but the two can be part of the same sensor network, share the same gateway and provide data to the same services.

5.4 Modeling and Simulation of a WoT System as a Whole

A *WoTS simulation* consists of one or more simulated WoTS nodes, an environment simulator, and zero or more real WoTS nodes, such that

1. All simulated quality detectors (on simulated sensor and actuator nodes) can sense/read as their input the value of an attribute of their simulated reach corresponding to the quality to be measured. A simulated analog quality detector first maps the attribute value to a voltage value with the help of an attribute-value-to-voltage-value function, such that the simulated sensor node can then map it to the simulated measurement result with the help of a voltage-value-to-quantity function. A simulated digital quality detector directly maps the property value to the simulated measurement result with the help of a attribute-value -to-quantity function.
2. All simulated event detectors can detect simulated external perception events as inputs from the environment simulator
3. All simulated actuators on simulated actuator nodes create simulated action events as outputs to the environment simulator, which maps them to simulated physical signals as inputs to simulated sensors in the reach of the simulated actuator.
4. Simulated sensor nodes are not "coupled" with real actuator nodes: the reach of any real actuator node does not overlap with the reach of any simulated

sensor node. The reach of an actuator node is its local environment, in which real state changes can be caused by it. The reach of a simulated sensor node is the spatial region corresponding to its sensing radius in the simulated local environment of the simulated sensor node.

This definition of a WoTS simulation includes the special case where all nodes are simulated.

Notice that while there is a crisp boundary between real and simulated sensor and actuator nodes, the boundary between real and simulated service nodes is more fuzzy, since service nodes are not connected to the "real world", but only to digital network signals, which may represent real or simulated signals.

6 A Green House Test Case

Our Green House scenario considers a closed environment with three important parameters: soil moisture, air temperature and relative humidity. It has an area of 1250 m^2 ($50 \times 25m$) and a volume of $3750m^3$ ($50 \times 25 \times 3m$).

We consider a plant type, for which the optimal values corresponding to the three qualities and the quantity of water consumption per unit of time are known. The temperature variable (producing water vaporization) but also the water consumed by the plant affects both, the soil moisture and the relative air humidity.

6.1 Simulated Hardware Configuration

Two sensor types (three quality detectors) are used in our simulation, measuring the important environment qualities:

- DHT22/AM2302 digital temperature and humidity sensor with a custom 1-Wire digital interface (two quality detectors on the same physical package, with a common data interface). It allows measurements of air humidity in the range from [0, 99]% with a typical accuracy of 2 % and precision of 1 %, as well as temperature measurements in the range from [-40, 80]°C with a typical accuracy of 0.5°C and precision of 0.2°C.
- GROVE analog soil moisture detector, with a range of [0, 100]% and a typical accuracy of 10 %. The datasheet does not provide information about the sensor's precision, but our research on the web has shown that typically a value of 5 % is to be expected for this sensor type. This sensor has an analog quality detector.

Two actuator types are used to control the three monitored environment qualities, i.e., soil moisture, air temperature and relative humidity:

- Heaters, activated or deactivated by `PullDownRelay` digital actuators, are used to increase the temperature when needed. Each heater is able to increase the temperature with 5°C per hour, for $750m^3$ volume of air. Cooling is not considered in this scenario. The pull down relays are directly connected to heaters, and the composition of the two components represents the actuator.

– Ventilators, activated or deactivated by `PullDownRelay` digital actuators, are used to provide air flow to and from outside the green house space, thus providing some degree of control for this quality. The ventilators can rotate either forward or backward and can be on or off, without speed control. Each ventilator is able to provide a volume of $1250m^3$ air per hour. The pull down relays are directly connected to ventilators, and the composition of the two represents the actuator.
– A `Pump` digital actuator is used to increase the soil moisture when required. It can be on (provides water flow) or off (no water flows). The pump is able to provide $9\,dm^3$ water per minute.

A simulated sensor node consists of an Arduino Mini [1] controller (a cheap variation of Arduino UNO, much lower in size and with half the power consumption), an WiFi communication module and a DHT22 or a Grove sensor (or both). Because the temperature and humidity are considered uniform over the entire green house space, only one node contains a DHT22 sensor.

A simulated actuator node consist of an Arduino Mini controller, an WiFi communication module and a pull down relay or a water pump actuator (or both). Some of the actuator nodes contains also Grove sensors, depending on their spatial position.

In general, the goal is to allow using real devices in combination with simulated devices. Technically, this is possible by using a virtual router, a piece of software, which allows the simulated sensor and actuator nodes to behave in the network same as real hardware: connects to a Wi-Fi or Wired network, acquire IP from DHCP and use standard communication protocols, e.g., CoAP over UDP, for data transmission.

6.2 Environment Simulation

In this scenarion, the simulated sensor and actuator nodes exist in a simulated environment. The variation of the environment qualities, such as air temperature and humidity, is sensed by the simulated sensors and transformed to quantities by the controller, e.g., soil moisture level (in percent), for soil moisture quality. The environment physical space is discrete and it is composed of cubic cells with a size of $1 \times 1 \times 1m$. A cell represents the atomic space unit. A sensor or actuator reach (a region in the environment space) consists of a neighborhood set of such cells. Sensor reaches and actuator reaches share environment space (one or more cells), but a complete overlap is not required. The environment simulator takes into consideration the following qualities:

– *Temperature*: the variation from day to night is considered linear. The variation interval is set for 24 hours. The temperature starts to increase after sunrise until a specified daytime, e.g., 5:00 PM for summer time, then decreases until the next sunrise. Close to reality events, such as clouds, are simulated by introducing small random variations. In this scenario, we consider a uniform temperature distribution over the simulated environment (the temperature

value is the same on each space cell). The heaters are activated if the temperature goes below 22°C, and deactivated when the temperature reaches 25°C or more.

- *Air humidity*: the variation depends on the environment temperature and water dew point. Water dew point is the temperature at which the air can no longer hold all of the water vapor which is mixed with it, and some of the water vapor must condense into liquid water. In this scenario, air flow to and from outside allows to increase or decrease the air humidity. The minimum and maximum values for this quality are dependent on the values from the space outside the green house. The formula used to compute the relative air humidity is: $RH = 10^{m\left(\frac{T_d}{T_d+T_n} - \frac{T_a}{T_a+T_n}\right)} * 100\,\%$, where T_d is the water dew point temperature, T_a is the temperature in the environment, while m and T_n are constants which depend on temperature ranges and are provided in constant tables. In this scenario, we consider a uniform air humidity distribution over the simulated green house environment (the value of this quality is the same on each space cell).
- *Soil moisture*: the variation depends on both, the temperature which affects the water vaporization, and the quantity of water known to be consumed by the plant. When watering is required, because the soil moisture is below a threshold, the water pump is started. The water distribution in soil is computed by using the following formula: $\frac{d\theta}{dt} = \frac{d}{dr}\left(D\frac{d\theta}{dr}\right) - S$, where θ is the volumetric water content, t is the time, D is the soil water diffusivity, r is the radius, and S is the water uptake by the plant(s) root. In this scenario, for simplicty reasons, we do not consider the gravitational force when computing the water distribution in the soil, therefore the soil moisture is the same no matter the depth in the soil at which it is measured.

During the simulation runtime, various activities are possible. For example, when the measured soil moisture goes under a specified level, a *watering activity* is started. A *water pump* is then activated in the specific reach. A known quantity of water per unit of time starts to flow and the soil moisture in the actuator reach starts to increase (according with the above presented formula). The soil moisture sensor reads the value of this quality, and when it reaches an optimum level, the water pump is deactivated.

7 Conclusions

We have presented a proposal for modeling and simulating certain types of sensor-actuator systems and WoT systems consisting of simple sensors based on quality detectors and event detectors, such as LM35 analog temperature sensors and Proximity Infra-Red (PIR) sensors. Although our approach is more general than the approaches discussed in the section on related work, it does not provide a completely general model of sensors and actuators, since it does, for instance, not account for more advanced types of sensors such as LIDAR devices and video cameras. We work on a JavaScript implementation of the proposed simulation framework and expect to be able to present evaluation results, of the simulator and the presented test case, in a follow-up paper.

References

1. Arduino Foundation: Arduino Platform (2005). http://arduino.cc
2. Brambilla, G., Picone, M., Cirani, S., Amoretti, M., Zanichelli, F.: A simulation platform for large-scale internet of things scenarios in urban environments. In: Proceeding of the of the First International Conference on IoT in Urban Space (Urb-IoT 2014), Rome, Italy, pp. 50–55 (2014). Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (ICST)
3. Diaconescu, M., Wagner, G.: Modeling and simulation of web-of-things systems Part 1: sensor nodes. In: Winter Simulation Conference (WSC 2015), Huntington Beach, CA (To appear)
4. Karnouskos S., Tariq, M.M.J.: An agent-based simulation of SOA-ready devices. In: Proceedings of the Tenth International Conference on Computer Modeling and Simulation, pp. 330–335 (2008). IEEE Computer Society
5. Microsoft Corporation: Devices profile for web services. http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf (2006). Accessed 23 May 2015
6. Leppnen, T., Riekki, J.: A lightweight agent-based architecture for the Internet of Things. IEICE Technical Report
7. Leppnen, T., Riekki, J., Liu, M., Harjula, E., Ojala, T.: Mobile agents-based smart objects for the internet of things. In: Fortino, G., Trunfio, P. (eds.) Internet of Things Based on Smart Objects, pp. 29–48. Springer, Heidelberg (2014)
8. Shelby, Z.: Embedded web services. IEEE Wirel. Commun. **17**(6), 52–57 (2010)
9. Shelby, Z., Hartke, K., Bormann, C.: Constrained Application Protocol (CoAP) RFC 7252, June 2014. Internet Engineering Task Force (IETF)
10. Wagner, G.: The agent-object-relationship meta-model: towards a unified view of state and behavior. Inf. Syst. **28**(5), 475–504 (2003)