Where Do Your IoT Ingredients Come From?

Chiara Bodei, Pierpaolo Degano, Gian-Luigi Ferrari, Letterio Galletta

Where do your IoT ingredients come from? ⋆

Chiara Bodei, Pierpaolo Degano, Gian-Luigi Ferrari, and Letterio Galletta

Dipartimento di Informatica, Università di Pisa
{chiara,degano,giangi,galletta}@di.unipi.it

Abstract. The Internet of Things (IoT) is here: smart objects are pervading our everyday life. Smart devices automatically collect and exchange data of various kinds, directly gathered from sensors or generated by aggregations. Suitable coordination primitives and analysis mechanisms are in order to design and reason about IoT systems, and to intercept the implied technology shifts. We address these issues by defining IoT-LySA, a process calculus endowed with a static analysis that tracks the provenance and the route of IoT data, and detects how they affect the behaviour of smart objects.

1 Introduction

This is the era of the Internet of Things (IoT), where digitally connected devices are intruding into our everyday life. "Software is eating the world" is the vivid slogan referring to the *smartification* of the objects and devices around us. As buzzword, the IoT is indeed simple and accurate: a global network of things ranging from light bulbs to cars, equipped with suitable software allowing things to interact each other and coordinate their behaviour. For instance, our smart alarm clock can drive our coffeemaker to prepare us a cup of coffee in the morning, while our smart TV can suggest us some movies for the evening. Furthermore, smart devices can automatically exchange information of various kinds gathered from different sources (e.g. sensors) or generated by aggregating several data sets.

More connected smart devices and more applications available on the IoT mean more software bugs and vulnerabilities to identify and fix. For instance, a bug can cause you to wake up into a cold house in winter or an attacker can enter into your smart TV and break your bank account. This is not a big surprise: every advance in information technology has exposed software to new challenges.

Smart devices exhibit and require *open-endedness* to achieve full interactive and cooperative behaviour, and thus they generalise the so-called "embedded systems." These are essentially controllers of machines and are *closed* systems. Therefore, we cannot simply rely on standard techniques for supporting the design and development of IoT, and new software solutions have emerged, e.g. Amazon AWS for IoT and Google Brillo. We argue that the formal techniques and tools need to be adapted in order to support open-endedness of IoT applications and the new complex phenomena that arise in this hybrid scenario.

⋆ Partially supported by Università di Pisa PRA_2016_64 Project *Through the fog*.

Here, we contribute to this new line of research by introducing the kernel of a formal design framework for IoT, which will provide us with the foundations to develop verification techniques and tools for certifying properties of IoT applications.

Our starting point is the process calculus IoT-LySA, a dialect of LySA [4,6], within the process calculi approach to IoT [15,8]. It has primitive constructs to describe the activity of sensors and of actuators, and suitable primitives for managing the coordination and communication capabilities of smart objects. More precisely, our calculus is made up from:

1. systems of nodes, consisting of (a representation of) the physical components, i.e. sensors and actuators, and of software control processes for specifying the *logic* of the node, including the manipulation of data gathered from sensors and from other nodes. Intra-node generative communications are implemented through a shared store à la Linda [12,7]. The adoption of this coordination model supports a smooth implementation of the *cyber-physical control architecture*: physical data are made available to software entities that analyse them and trigger the relevant actuators to perform the desired behaviour.
2. an asynchronous multi-party communication among nodes, which can be easily tuned to take care of various constraints, mainly those concerning proximity;
3. functions to process and aggregate data.

A further contribution of this paper is the definition of an analysis for IoT-LySA to statically predict the run time behaviour of smart systems. We introduce a Control Flow Analysis (CFA) that safely approximates the abstract behaviour of a system of nodes. Essentially, it describes the interactions among nodes, tracks how data spread from sensors to the network, and how data are manipulated.

Technically, our CFA abstracts from the concrete values and only considers their provenance and how they are put together. In more detail, it returns for each node ℓ in the network:

- an abstract store $\hat{\Sigma}_\ell$ that records for sensors and variables a super-set of the abstract values that they may denote at run time;
- a set $\kappa(\ell)$ that over-approximates the set of the messages received by the node ℓ, and for each of them its sender;
- a set $\Theta(\ell)$ of possible abstract values computed and used by the node ℓ.

The result of the analysis can be exploited as the basis for checking and certifying various properties of IoT systems. As it is, the components κ and Θ track how data may flow in the network and how they influence the outcome of functions. An example of property that can be statically checked using the component κ is the detection of redundant communications, thus providing the means for refactoring the system to avoid message storms. Further, the analysis can be used to check whether the values produced by a certain classified sensor reaches an untrusted node. This helps evaluating the security level of the system and detecting potential vulnerabilities.

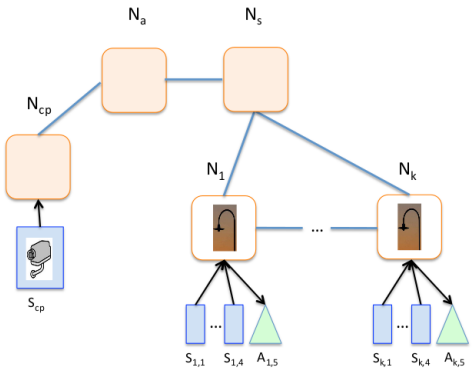

Fig. 1. The organisation of nodes in our street light control system.

The paper is organised as follows. The next section introduces our approach with the help of an illustrative example. In Sect. 3 we briefly introduce the process calculus IoT-LySA, while we present our CFA in Sect. 4. Concluding remarks and related work are in Sect. 5.

2 A smart street light control system

The IoT European Research Cluster (IERC) has recently identified *smart lighting* in *smart cities* [14] as one of most relevant applications for the Internet of Things. Recent studies, e.g. [10,11], show that smart street light control systems represent effective solutions to improve energy efficiency. Many proposed solutions are based on sensors that acquire data about the physical environment and regulate the level of illumination according to the detected events. In this section we show how this kind of scenario can be easily modelled in IoT-LySA and what kind of information our CFA provides to designers.

We consider a simplified system made of two integrated parts, working on a one-way street. The first consists of smart lamp posts that are battery powered and can sense their surrounding environment and can communicate with their neighbours to share their views. If (a sensor of) the lamp post perceives a pedestrian and there is not enough light in the street it switches on the light and communicates the presence of the pedestrian to the lamp posts nearby. When a lamp post detects that the level of battery is low, it informs the supervisor of the street lights, N_s, that will activate other lamp posts nearby. The second component of the street light controller uses the electronic access point to the street. When a car crosses the checkpoint, besides detecting if it is enabled to, a message is sent to the supervisor of the street accesses, N_a, that in turn notifies the presence of the car to N_s. This supervisor sends a message to the lamp post closest to the checkpoint that starts a forward chain till the end of the street. The structure of our control light system is in Fig. 1.

We first define the checkpoint N_{cp} as an IoT-LySA node that only contains a visual sensor S_{cp} to take a picture of the car detected in the street, defined as

$$S_{cp} = \mu h.(\tau.1 := v_p).\tau.\ h$$

where v_p is the picture of the car. The sensor communicates the picture to the node by storing it in the location 1 of the shared store. In our model we assume that each sensor has a reserved store location in which records its readings. The action τ denotes internal actions of the sensor, which we are not interested to model, e.g. reading from the environment; the construct $\mu h.$ implements the iterative behaviour of the sensor. Then, the taken picture is enhanced by the process P_{cp} and sent to the supervisor N_a

$$P_{cp} = \mu h.(z := 1).(z' := noiseRed(z)).\langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}.\ h$$

where ℓ_a is the label of the node N_a (note that 1 is the identifier of the sensor S_{cp} in the assignment $z := 1$). Hence, the checkpoint N_{cp} is defined as

$$N_{cp} = \ell_{cp} : [P_{cp} \parallel S_{cp} \parallel B_{cp}]$$

where ℓ_{cp} is the identifier of N_{cp} and B_{cp} abstracts other components we are not interested in. The node N_a (checks if the car is allowed to enter the street and) communicates its presence to the lamp posts supervisor N_s:

$$N_a = \ell_a : [\ \mu h.(;\ x).\ \langle\!\langle car, x \rangle\!\rangle \triangleright \{\ell_s\}.\ h \parallel B_a\ ]$$

where ℓ_s is the identifier of N_s (see below for the intuition of the general format of the input $(;x)$). The supervisor N_s contains the process $P_{s,1}$ that receives the picture from N_a and sends a message to the node closest to the checkpoint, call it N_1, labelled with ℓ_1:

$$P_{s,1} = \mu h.(car;\ x).\ \langle\!\langle x \rangle\!\rangle \triangleright \{\ell_1\}.\ h$$

The input $(car; x)$ is performed only if the corresponding output matches the constant car, and the store variable x is bound to the value of the second element of the output (see below for the full definition of N_s).

In our intelligent street light control system there is a node N_p for each lamp post, each of which has a unique identifier $p \in [1, k]$. The lamp posts have four sensors to sense (1) the environment light, (2) the solar light, (3) the battery level and (4) the presence of a pedestrian. Each of them is defined as follows

$$S_{p,i} = \mu\, h.\, (i := v).\ \tau.\ h$$

where v is the perceived value and $i \in [1, 4]$ are the store locations for the sensors. After some internal actions τ, the sensor $S_{p,i}$ iterates its behaviour. The actuator for the lamp post p is defined as

$$A_5 = \mu\, h.\, (\!|5, \{\mathsf{turnon}, \mathsf{turnoff}\}|\!).\ h$$

It only accepts a message from N_c whose first element is its identifier (here 5) and whose second element is either command turnon or turnoff and executes it.

The control process of a lamp post node is composed by two parallel processes. The first process $P_{p,1}$ is defined as follow

$$\begin{aligned} P_{p,1} = \mu h.(x_1 := 1.\, x_2 := 2.\, x_3 := 3.\, x_4 := 4).\\ (x_4 = true)\ ?\\ (x_1 \leq th_1 \wedge x_2 \leq th_2)\ ?\\ (x_3 \geq th_3)\ ?\ \langle 5, \mathsf{turnon}\rangle.\, \langle\!\langle x_4 \rangle\!\rangle \triangleright L_p.\ h\\ :\ \langle\!\langle \mathsf{err}, \ell_p \rangle\!\rangle \triangleright \{\ell_s\}.\ h\\ :\ h\\ :\ \langle 5, \mathsf{turnoff}\rangle.\, h \end{aligned}$$

The process reads the current values from the sensors and stores them into the local variables x_i. The actuator is turned on if (i) a pedestrian is detected in the street (x_4 holds), (ii) the intensity of environment and solar lights are greater than the given thresholds th_1 and th_2, and (iii) there is enough battery (at least th_3). In addition, the presence of the pedestrian is communicated to the lamp posts nearby, whose labels, typically ℓ_{p-1} and ℓ_{p+1}, are in L_p. Instead, if the level battery is insufficient, an error message, including its identifier ℓ_p, is sent to the supervisor node, labelled ℓ_s. The second process $P_{p,2}$ is defined as follows:

$$P_{p,2} = \mu h.(; x).\, (x = true \vee is\_a\_car(x))\ ?\ (\langle 5, \mathsf{turnon}\rangle.\, \langle\!\langle x \rangle\!\rangle \triangleright L_p).h : \langle 5, \mathsf{turnoff}\rangle.\, h$$

It waits for messages from its neighbours or from the supervisor node N_s. When one of them is notified the presence of a pedestrian ($x = true$) or of a car ($is\_a\_car(x)$ holds), the current lamp post orders the actuator to switch the light on. Each lamp post p is described as the IoT-LySA node below:

$$N_p = \ell_p : [\Sigma_p \parallel P_{p,1} \parallel P_{p,2} \parallel S_{p,1} \parallel S_{p,2} \parallel S_{p,3} \parallel S_{p,4} \parallel A_{p,5}]$$

where Σ_p is the store of the node ℓ_p, shared with its components. The supervisor node N_s of lamp posts is defined as

$$N_s = \ell_s : [\mu h.\, (err; x).\, \langle\!\langle true \rangle\!\rangle \triangleright L_x.\ h \ \parallel P_{s,1} \parallel B_s]$$

where $P_{s,1}$ is the process previously defined. As above the input $(err; x)$ is performed only if the corresponding output matches the constant err, and the store variable x is bound to the value of the second element of the output i.e. the label of the relevant lamp post. If this is the case, after some internal elaborations N_s warns the lamp posts nearby x (included in L_x) of the presence of a pedestrian.

Therefore, the whole intelligent controller N of the street lights is described as the parallel composition of the checkpoint node N_{cp}, the supervisors nodes N_a and N_s, and the nodes of lamp posts N_p, with $p \in [1, k]$:

$$N = N_{cp} \mid N_a \mid N_s \mid N_1 \mid \cdots \mid N_k$$

We would like to statically predict how the system behaves at run time. In particular, we want to compute: (i) how nodes interact each other; (ii) how data spread from sensors to the network (tracking); and (iii) which computations each node performs on the received data. To do that, we define a Control Flow Analysis (CFA), which abstracts from the concrete values by only considering their provenance and how they are manipulated. Consider e.g. the picture sent by the camera of S_{cp} to its control process P_{pc}. In the analysis we are only interested in tracking where the picture comes from, and not in its actual value; so we use the abstract value $1^{\ell_{cp}}$ to record the camera that took it. The process P_{pc} reduces the noise in the pictures and sends the result to N_a. Our analysis keeps track of this manipulation through the abstract value $noiseRed^{\ell_{cp}}(1^{\ell_{cp}})$, meaning that the function $noiseRed$, computed by the node ℓ_{cp}, is applied to data coming from the sensor with identifier 1 of ℓ_{cp}.

In more detail, our CFA returns for each node ℓ in the network: an abstract store $\hat{\Sigma}_\ell$ that records for each variable a super-set of the abstract values that it may denote at run time; a set $\kappa(\ell)$ that approximates the set of the messages received by the node ℓ; and the set $\Theta(\ell)$ of possible abstract values computed and used by the node ℓ.

In our example, for each lamp post labelled ℓ_p, the analysis returns in $\kappa(\ell_p)$ both the abstract value $noiseRed^{\ell_{cp}}(1^{\ell_{cp}})$ and the sender of that message, i.e. ℓ_{p+1}. The result of our analysis can be exploited to perform several verifications. For instance, since the pictures of cars are sensitive data, one would like to check whether they are kept secret. By inspecting κ and Θ we discover that the sensitive data of cars is sent to all lamp posts, so possibly violating privacy. Another example is detecting whether there are redundant communications, e.g. since the street is one-way, when a car is present the lamp post at position p needs not to alert the one at $p-1$. From κ it is easy to detect a redundant communication from the next lamp post.

3 The calculus IoT-LySA

We adapt the LySA calculus [3,4,6], based on the π- [17] and Spi-calculus [1], to model IoT applications. For that we introduce: (i) systems of nodes, in turn consisting of sensors, actuators and control processes, plus a shared store Σ within each node for internal communications; (ii) primitives for reading from sensors, and for triggering actuator actions; (iii) an asynchronous multi-party communication modality among nodes, subject to constraints, mainly concerning proximity; (iv) functions to process data; (v) explicit conditional statements. For brevity, we do not include here encryption and decryption primitives as in LySA.

Syntax. Systems have a two-level structure and consist of a fixed number of labelled nodes, hosting a store, control processes, sensors and actuators. The label ℓ uniquely identifies the node $\ell : [B]$ and may represent further characterising information (e.g. its location or other contextual information). Finally, the operator $|$ describes a node system of nodes obtained by parallel composition. The

syntax of nodes is as follows.

$\mathcal{N} \ni N ::=$	*systems of nodes*	
	$\mathsf{0}$	inactive node
	$\ell : [B]$	single node ($\ell \in \mathcal{L}$, the set of labels)
	$N_1 \mid N_2$	parallel composition of nodes

$\mathcal{B} \ni B ::=$	*node components*	
	Σ_ℓ	node store
	P	process
	S	sensor, with a unique identifier $i \in \mathcal{I}_\ell$
	A	actuator, with a unique identifier $i \in \mathcal{J}_\ell$
	$B \parallel B$	parallel composition of node components

We impose that in $\ell : [B]$ there is a *single* store $\Sigma_\ell : \mathcal{X} \cup \mathcal{I}_\ell \ \rightarrow \mathcal{V}$, where $\mathcal{X}, \mathcal{V}$ are the sets of variables and of values, respectively. Our store is essentially an array of fixed dimension, so intuitively a variable is the index in the array and an index $i \in \mathcal{I}_\ell$ corresponds to a single sensor (no need of α-conversions). We assume that store accesses are atomic, e.g. through CAS instructions [13]. The other node components are obtained by the parallel composition of control processes P, and of a fixed number of (less than $\#(\mathcal{I}_\ell)$) sensors S, and actuators A (less than $\#(\mathcal{J}_\ell)$). The syntax of processes is as follows

$\mathcal{P} \ni P ::=$	*control processes*	
	$\mathsf{0}$	inactive process
	$\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L.\, P$	asynchronous multi-output L$\subseteq \mathcal{L}$
	$(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).\, P$	input (with matching)
	$E?P : Q$	conditional statement
	h	iteration variable
	$\mu h.\ P$	tail iteration
	$x := E.\, P$	assignment to $x \in \mathcal{X}$
	$\langle j, \gamma \rangle.\, P$	output of action γ to actuator j

The prefix $\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L$ implements a simple form of multi-party communication among nodes: the tuple $E_1, \ldots, E_k$ is asynchronously sent to the nodes with labels in L and that are "compatible" (according, among other attributes, to a proximity-based notion). The input prefix $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k)$ is willing to receive a k-tuple, provided that its first j elements match the input ones, and then binds the remaining store variables (separated by a ";") to the corresponding values (see [6,2] for a more flexible choice). Otherwise, the k-tuple is not accepted. A process repeats its behaviour, when defined through the tail iteration construct $\mu h.\ P$, where h is the iteration variable.

A sensor can perform an internal action τ or store the value v, gathered from the environment, into its store location i. An actuator can perform an internal action τ or execute one of its action γ, possibly received from its controlling process. Both sensors and actuators can iterate. For simplicity, here we neither

provide an explicit operation to read data from the environment, nor to describe the impact of actuator actions on the environment. Sensors and actuators (uniquely labelled) have the form:

$\mathcal{S} \ni S ::=$ *sensors*		$\mathcal{A} \ni A ::=$ *actuators*	
$\mathsf{0}$	inactive sensor	$\mathsf{0}$	inactive actuator
$\tau.S$	internal action	$\tau.A$	internal action
$i := v.\, S$	store of $v \in \mathcal{V}$	$(\!\|j, \Gamma\|\!).\, A$	command for actuator j
	by the i^{th} sensor	$\gamma.A$	triggered action ($\gamma \in \Gamma$)
h	iteration var.	h	iteration var.
$\mu\, h\, .\, S$	tail iteration	$\mu\, h\, .\, S$	tail iteration

The syntax of terms follows.

$\mathcal{E} \ni E ::=$ *terms*	
v	value ($v \in \mathcal{V}$)
i	sensor location ($i \in \mathcal{I}_\ell$)
x	variable ($x \in \mathcal{X}$)
$f(E_1, \cdots, E_n)$	function on data

The term $f(E_1, \cdots, E_n)$ is the application of function f to n arguments; we assume given a set of primitive functions, typically for aggregating or comparing values, be them computed or representing data in the environment.

Operational Semantics. Our reduction semantics assumes the following *structural congruence* $\equiv$ on nodes, processes and sensors. It is standard except for the last rule that equates a multi-output with no receivers to the inactive process.

- $(\mathcal{N}/_{\equiv}, |, \mathsf{0})$ and $(\mathcal{B}/_{\equiv}, \|, \mathsf{0})$ are commutative monoids
- $\mu\, h\, .\, X \equiv X\{\mu\, h\, .\, X/h\}$ for $X \in \{P, A, S\}$
- $\langle\langle E_1, \cdots, E_k\rangle\rangle : \emptyset.\ \mathsf{0} \equiv \mathsf{0}$

We have a two-level *reduction relation* defined as the least relation on nodes and its components, denoted by $\rightarrow$, satisfying the set of inference rules in Table 1. We assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms.

The first two rules implement the (atomic) asynchronous update of shared variables inside nodes, by using the standard notation $\Sigma\{-/-\}$. According to (S-store), the i^{th} sensor uploads the value v, gathered from the environment, into the store location i. According to (Asgm), a control process updates the variable x with the value of E. The rules (Ev-out) and (Multi-com) drive asynchronous multi-communications among nodes. In the first a node labelled ℓ willing to send a tuple of values $\langle\!\langle v_1, ..., v_k\rangle\!\rangle$, obtained by the evaluation of $\langle\!\langle E_1, ..., E_k\rangle\!\rangle$, spawns a new process, running in parallel with the continuation P; its task is to offer the evaluated tuple to all its receivers L. In the rule (Multi-com), the message coming from ℓ_1 is received by a node labelled ℓ_2. The communication succeeds, provided that (i) ℓ_2 belongs to the set L of possible receivers, (ii) the two nodes are compatible according to the compatibility function *Comp*, and (iii) that the first j values match with the evaluations of the first j terms in the input.

(S-store)
$$\frac{}{\Sigma \parallel i := v.\, S_i \parallel B \rightarrow \Sigma\{v/i\} \parallel S_i \parallel B}$$

(Asgm)
$$\frac{[\![E]\!]_\Sigma = v}{\Sigma \parallel x := E.\, P \parallel B \rightarrow \Sigma\{v/x\} \parallel P \parallel B}$$

(Ev-out)
$$\frac{\bigwedge_{i=1}^{k} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L.\, P \parallel B \;\rightarrow\; \Sigma \parallel \langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L.\mathsf{0} \parallel P \parallel B}$$

(Multi-com)
$$\frac{\ell_2 \in L \wedge \; Comp(\ell_1, \ell_2) \wedge \; \bigwedge_{i=1}^{j} v_i = [\![E_i]\!]_{\Sigma_2}}{\begin{array}{c}\ell_1 : [\langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L.\, \mathsf{0} \parallel B_1] \mid \ell_2 : [\Sigma_2 \parallel (E_1, \cdots, E_j; x_{j+1}, \cdots, x_k).Q \parallel B_2] \\ \rightarrow \\ \ell_1 : [\langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L \setminus \{\ell_2\}.\, \mathsf{0} \parallel B_1] \mid \ell_2 : [\Sigma_2\{v_{j+1}/x_{j+1}, \cdots, v_k/x_k\} \parallel Q \parallel B_2]\end{array}}$$

(Cond)
$$\frac{[\![E]\!]_\Sigma = \mathtt{b_i}}{\Sigma \parallel E?\, P_1 : P_2 \parallel B \;\rightarrow\; \Sigma \parallel P_i \parallel B} \text{ where } \mathtt{b_1} = \mathtt{true}, \mathtt{b_2} = \mathtt{false}$$

(A-com)
$$\frac{\gamma \in \Gamma}{\langle j, \gamma \rangle.\, P \parallel (\!| j, \Gamma |\!).\, A \parallel B \;\rightarrow\; P \parallel \gamma.\, A \parallel B}$$

(Act)
$$\frac{}{\gamma.A \;\rightarrow\; A}$$

(Int)
$$\frac{}{\tau.\, X \;\rightarrow\; X}$$

(Node)
$$\frac{B \;\rightarrow\; B'}{\ell : [B] \;\rightarrow\; \ell : [B']}$$

(ParN)
$$\frac{N_1 \rightarrow N_1'}{N_1 | N_2 \rightarrow N_1' | N_2}$$

(ParB)
$$\frac{B_1 \rightarrow B_1'}{B_1 \| B_2 \rightarrow B_1' \| B_2}$$

(CongrY)
$$\frac{Y_1' \equiv Y_1 \rightarrow Y_2 \equiv Y_2'}{Y_1' \rightarrow Y_2'}$$

Table 1. Reduction semantics, where $X \in \{S, A\}$ and $Y \in \{N, B\}$.

Moreover, the label ℓ_2 is removed by the set of receivers L of the tuple. The spawned process terminates when all its receivers have received the message (see the last congruence rule). The role of the compatibility function *Comp* is crucial in modelling real world constraints on communication. A basic requirement is that inter-node communications are proximity-based, i.e. that only nodes that are in the same transmission range can directly exchange messages. This is easily encoded here by defining a predicate (over node labels) yielding true only when two nodes are in the same transmission range. Of course, this function could be enriched in order to consider finer notions of compatibility expressing various policies, e.g. topics for event notification. Note that if *Comp* varies along time, we recover a simple way of expressing dynamic network topologies. According to the evaluation of the expression E, the rule (Cond) says that the process continues as P_1 (if $[\![E]\!]_\Sigma$ is true) or as P_2 (otherwise). A process commands the j^{th} actuator through the rule (A-com), by sending it the pair $\langle j, \gamma \rangle$; γ prefixes the actuator, if

it is one of its actions. The rule (Act) says that the actuator performs the action γ. Similarly, for the rules (Int) for internal actions. The last rules propagate reductions across parallel composition ((ParN) and (ParB)) and nodes (Node), while the (CongrY) are the standard reduction rules for congruence.

4 Control Flow Analysis

Our CFA aims at safely approximating the abstract behaviour of a system of nodes N. The analysis follows the same schema of that for LySA [4], and for the time being we only conjecture that computing its results requires the same low polynomial time complexity. Here, we track the usage of sensor values inside the local node where they are gathered and their propagation in the network of nodes both as raw data or processed via suitable functions. We resort to abstract values for sensor and functions on abstract values, as follows, where $\ell \in \mathcal{L}$:

$\hat{\mathcal{V}} \ni \hat{v} ::=$	*abstract terms*	
	$\top^\ell$	special abstract value denoting cut
	i^ℓ	sensor abstract value ($i \in \mathcal{I}_\ell$)
	v^ℓ	node abstract value
	$f^\ell(\hat{v}_1, \cdots, \hat{v}_n)$	function on abstract data

Since the dynamic semantics may introduce function terms with an arbitrarily nesting level, we have new special abstract values $\top^\ell$ that denote all those function terms with a depth greater that a given d. In the clauses defining our analysis, we will use $\lfloor - \rfloor_d$ to keep the maximal depth of abstract terms less or equal to d, defined as expected. Note that, once given the set of functions f occurring in a node N, the abstract values are finitely many.

The result of our CFA is a triple $(\hat{\Sigma}, \kappa, \Theta)$ (a pair $(\hat{\Sigma}, \Theta)$ for terms E, resp.), called *estimate* for N (for E, resp.), that satisfies the judgements defined by the rules of Tables 3 and 2. For this we introduce the following *abstract domains*:

- *abstract store* $\hat{\Sigma} = \bigcup_{\ell \in \mathcal{L}} \hat{\Sigma}_\ell : \mathcal{X} \cup \mathcal{I}_\ell \rightarrow 2^{\hat{\mathcal{V}}}$ where each *abstract local store* $\hat{\Sigma}_\ell$ approximates the concrete local store Σ_ℓ, by associating with each location a set of abstract values that represent the possible concrete values that the location may store at run time.
- *abstract network environment* $\kappa : \mathcal{L} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \hat{\mathcal{V}}^i$ (with $\hat{\mathcal{V}}^{i+1} = \hat{\mathcal{V}} \times \hat{\mathcal{V}}^i$ and k maximum arity of messages), that includes all the messages that may be received by the node labelled ℓ.
- *abstract data collection* $\Theta : \mathcal{L} \rightarrow 2^{\hat{\mathcal{V}}}$ that, for each node labelled ℓ, approximates the set of values that the node computes.

For each term E, the judgement $(\hat{\Sigma}, \Theta) \models_\ell E : \vartheta$, defined by the rules in Table 2, expresses that $\vartheta \in \hat{\mathcal{V}}$ is an acceptable estimate of the set of values that E may evaluate to in $\hat{\Sigma}_\ell$. A sensor identifier and a value evaluate to the set ϑ, provided that their abstract representations belong to ϑ. Similarly a variable x evaluates to ϑ, if this includes the set of values bound to x in $\hat{\Sigma}_\ell$. The last rule analyses

$$\frac{i^\ell \in \vartheta \subseteq \Theta(\ell)}{(\hat{\Sigma}, \Theta) \models_\ell i : \vartheta} \qquad \frac{v^\ell \in \vartheta \subseteq \Theta(\ell)}{(\hat{\Sigma}, \Theta) \models_\ell v : \vartheta} \qquad \frac{\hat{\Sigma}_\ell(x) \subseteq \vartheta \subseteq \Theta(\ell)}{(\hat{\Sigma}, \Theta) \models_\ell x : \vartheta}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\hat{\Sigma}, \Theta) \models_\ell E_i : \vartheta_i \wedge \\ \forall\, \hat{v}_1, \cdots, \hat{v}_k : \bigwedge_{i=1}^{k} \hat{v}_i \in \vartheta_i \;\Rightarrow\; \lfloor f^\ell(\hat{v}_1, \cdots, \hat{v}_k) \rfloor_d \in \vartheta \subseteq \Theta(\ell)\end{array}}{(\hat{\Sigma}, \Theta) \models_\ell f(E_1, \cdots, E_k) : \vartheta}$$

Table 2. Analysis of terms $(\hat{\Sigma}, \Theta) \models_\ell E : \vartheta$.

the application of a k-ary function f to produce the set ϑ. Recall that the special abstract value $\top^\ell$ will end up in ϑ if the depth of the abstract functional term exceeds d, and it represents all the functional terms with nesting greater than d. To do that (i) for each term E_i, it finds the sets ϑ_i, and (ii) for all k-tuples of values $(\hat{v}_1, \cdots, \hat{v}_k)$ in $\vartheta_1 \times \cdots \times \vartheta_k$, it checks if the abstract values $f^\ell(\hat{v}_1, \cdots, \hat{v}_k)$ belong to ϑ. Moreover, in all the rules for terms, we require that $\Theta(\ell)$ includes all the abstract values included in ϑ. This guarantees that only those values actually used are tracked by Θ, in particular those of sensors.

In the analysis of nodes we focus on which values can flow on the network and which can be assigned to variables. The judgements have the form $(\hat{\Sigma}, \kappa, \Theta) \models N$ and are defined by the rules in Table 3. The rules for the *inactive node* and for *parallel composition* are standard. Moreover, the rule for a single node $\ell : [B]$ requires that its component B is analysed, with the further judgment $(\hat{\Sigma}, \kappa, \Theta) \models_\ell B$, where ℓ is the label of the enclosing node. The rule connecting actual stores Σ with abstract ones $\hat{\Sigma}$ requires the locations of sensors to contain the corresponding abstract values. The rule for sensors is trivial, because we are only interested in who will use their values, and so is that for actuators. The rules for processes are in Table 3, and all require that an estimate is also valid for the immediate sub-processes. The rule for k-ary *multi-output* (i) finds the sets ϑ_i, for each term E_i; and (ii) for all k-tuples of values $(\hat{v}_1, \cdots, \hat{v}_k)$ in $\vartheta_1 \times \cdots \times \vartheta_k$, it checks if they belong to $\kappa(\ell' \in L)$, i.e. they can be received by the nodes with labels in L. In the rule for *input* the terms $E_1, \cdots, E_j$ are used for matching values sent on the network. Thus, this rule checks whether (i) these first j terms have acceptable estimates ϑ_i; (ii) the two nodes can communicate ($Comp(\ell', \ell)$); and whether (iii) for each message $(\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_j, \hat{v}_{j+1}, \ldots, \hat{v}_k \rangle\!\rangle)$ in $\kappa(\ell)$ (i.e. in any message predicted to be receivable by the node with label ℓ) the values $\hat{v}_{j+1}, \ldots, \hat{v}_k$ are included in the estimates for the variables $x_{j+1}, \cdots, x_k$. The rule for *assignment* requires that all the values $\hat{v}$ in ϑ, the estimate for E, belong to $\hat{\Sigma}_\ell(x)$. The rule for $\mu h.\, P$ reflects our choice of limiting the depth of function applications: the iterative process is unfolded d times. The remaining rules are as expected.

To show our analysis at work, consider again the example in Sect. 2 and the process $P_{cp} = \mu h.(z := 1).(z' := noiseRed(z)).\langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}.\ h$. Every valid CFA

$$\frac{}{(\hat{\Sigma},\kappa,\Theta) \models \mathsf{0}} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \models_\ell B}{(\hat{\Sigma},\kappa,\Theta) \models \ell : [B]} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \models N_1 \wedge (\hat{\Sigma},\kappa,\Theta) \models N_2}{(\hat{\Sigma},\kappa,\Theta) \models N_1 \mid N_2}$$

$$\frac{\forall\, i \in \mathcal{I}_\ell.\ i^\ell \in \hat{\Sigma}_\ell(i)}{(\hat{\Sigma},\kappa,\Theta) \models_\ell \Sigma} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \models_\ell S} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \models_\ell A}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k}\ (\hat{\Sigma},\Theta) \models_\ell E_i : \vartheta_i\ \wedge\ (\hat{\Sigma},\kappa,\Theta) \models_\ell P \wedge \\ \forall \hat{v}_1,\cdots,\hat{v}_k :\ \bigwedge_{i=1}^{k}\ \hat{v}_i \in \vartheta_i\ \Rightarrow \forall \ell' \in L : (\ell, \langle\!\langle \hat{v}_1,\cdots,\hat{v}_k \rangle\!\rangle) \in \kappa(\ell')\end{array}}{(\hat{\Sigma},\kappa,\Theta) \models_\ell \langle\!\langle E_1,\cdots,E_k \rangle\!\rangle \triangleright L.\, P}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{j}\ (\hat{\Sigma},\Theta) \models_\ell E_i : \vartheta_i\ \wedge Comp(\ell',\ell)\ \wedge \\ \forall (\ell', \langle\!\langle \hat{v}_1,\cdots,\hat{v}_k \rangle\!\rangle) \in \kappa(\ell) :\ \bigwedge_{i=j+1}^{k}\ \hat{v}_i \in \hat{\Sigma}_\ell(x_i)\ \wedge \\ (\hat{\Sigma},\kappa,\Theta) \models_\ell P\end{array}}{(\hat{\Sigma},\kappa,\Theta) \models_\ell (E_1,\cdots,E_j;\, x_{j+1},\cdots,x_k).\, P}$$

$$\frac{\begin{array}{c}(\hat{\Sigma},\Theta) \models_\ell E : \vartheta\ \wedge \\ \forall\, \hat{v} \in \vartheta\ \Rightarrow \hat{v} \in \hat{\Sigma}_\ell(x)\ \wedge (\hat{\Sigma},\kappa,\Theta) \models_\ell P\end{array}}{(\hat{\Sigma},\kappa,\Theta) \models_\ell x := E.\, P} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \models_\ell P}{(\hat{\Sigma},\kappa,\Theta) \models_\ell \langle j,\gamma \rangle.\, P}$$

$$\frac{(\hat{\Sigma},\Theta) \models_\ell E : \vartheta\ \wedge\ (\hat{\Sigma},\kappa,\Theta) \models_\ell P_1\ \wedge\ (\hat{\Sigma},\kappa,\Theta) \models_\ell P_2}{(\hat{\Sigma},\kappa,\Theta) \models_\ell E?P_1 : P_2}$$

$$\frac{}{(\hat{\Sigma},\kappa,\Theta) \models_\ell \mathsf{0}} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \models_\ell \lfloor \mu h.\, P \rfloor_d}{(\hat{\Sigma},\kappa,\Theta) \models_\ell \mu h.\, P} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \models_\ell h}$$

Table 3. Analysis of nodes $(\hat{\Sigma},\kappa,\Theta) \models N$, and of node components $(\hat{\Sigma},\kappa,\Theta) \models_\ell B$.

estimate must include at least the following entries (assuming $d = 4$):

$(a)\ \hat{\Sigma}_{\ell_{cp}}(z) \supseteq \{1^{\ell_{cp}}\}$ $\quad$ $(b)\ \hat{\Sigma}_{\ell_{cp}}(z') \supseteq \{noiseRed^{\ell_{cp}}(1^{\ell_{cp}}), 1^{\ell_{cp}}\}$
$(c)\ \Theta(\ell_{cp}) \supseteq \{1^{\ell_{cp}}, noiseRed^{\ell_{cp}}(1^{\ell_{cp}})\}$ $\quad$ $(d)\ \kappa(\ell_a) \supseteq \{(\ell_{cp}, \langle\!\langle noiseRed^{\ell_{cp}}(1^{\ell_{cp}}) \rangle\!\rangle)\}$

Indeed, all the following checks must succeed:

- $(\hat{\Sigma},\kappa,\Theta) \models_{\ell_{cp}} \mu h.(z := 1).(z' := noiseRed(z)).\langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}.h$ because
- $(\hat{\Sigma},\kappa,\Theta) \models_{\ell_{cp}} (z := 1).(z' := noiseRed(z)).\langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}$, that in turn holds
- because (i) $1^{\ell_{cp}}$ is in $\hat{\Sigma}_{\ell_{cp}}(z)$ by (a) $((\hat{\Sigma},\Theta) \models_\ell 1 : \vartheta \ni 1^{\ell_{cp}})$; and because (ii) $(\hat{\Sigma},\kappa,\Theta) \models_{\ell_{cp}} (z' := noiseRed(z)).\langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}$, that in turn holds
- because (i) $noiseRed^{\ell_{cp}}(1^{\ell_{cp}})$ is in $\hat{\Sigma}_{\ell_{cp}}(z')$ by (b) since $(\hat{\Sigma},\Theta) \models_{\ell_{cp}} noiseRed(z) : \vartheta \ni noiseRed^{\ell_{cp}}(1^{\ell_{cp}})$; and because (ii) $(\hat{\Sigma},\kappa,\Theta) \models_{\ell_{cp}} \langle\!\langle z' \rangle\!\rangle \triangleright \{\ell_a\}$ that holds because $(\ell_{cp}, \langle\!\langle noiseRed^{\ell_{cp}}(1^{\ell_{cp}}) \rangle\!\rangle)$ is in $\kappa(\ell_a)$ by (d).

Correctness of the analysis. Our CFA respects the operational semantics. The proof of this fact benefits from an instrumented denotational semantics for expressions, the values of which are pairs $\langle v, \hat{v}\rangle$. Consequently, the store (Σ^i_ℓ with a $\perp$ value) and its updates are accordingly extended (the semantics used in Table 1 is $[\![v]\!]^i_{\downarrow_1}$, the projection on the first component of the instrumented one).

Just to give an intuition, we will have $[\![v]\!]^i_{\Sigma^i_\ell} = (v, v^\ell)$, and the assignment $x := E$ will result in the updated store $\Sigma^i_\ell\{(v, v^\ell)/x\}$, where E evaluates to (v, v^ℓ). Clearly, the semantics of Table 1 is $[\![v]\!]^i_{\downarrow_1}$, the projection on the first component of the instrumented one. In our example, the assignment $z' := noiseRed(z)$ of the process P_{cp} stores the pair $(v, noiseRed^{\ell_{cp}}(1^{\ell_{cp}}))$ made of the actual value v and of its abstract counterpart.

Since the analysis only considers the second component of the extended store, it is immediate defining when the concrete and the abstract stores agree: $\Sigma^i_\ell \bowtie \hat{\Sigma}_\ell$ iff $w \in \mathcal{X} \cup \mathcal{I}_\ell$ such that $\Sigma^i_\ell(w) \neq \perp$ implies $(\Sigma^i_\ell(w))_{\downarrow_2} \in \hat{\Sigma}_\ell(w)$.

The following theorems establish the correctness of our CFA and the existence of a minimal estimate. Their proofs have the usual schema.

Theorem 1 (Subject reduction).
If $(\hat{\Sigma}, \kappa, \Theta) \models N$ and $N \to N'$ and $\forall \Sigma^i_\ell$ in N it is $\Sigma^i_\ell \bowtie \hat{\Sigma}_\ell$, then $(\hat{\Sigma}, \kappa, \Theta) \models N'$ and $\forall \Sigma^{i'}_\ell$ in N' it is $\Sigma^{i'}_\ell \bowtie \hat{\Sigma}_\ell$.

Theorem 2 (Existence of estimates).
Given N, its estimates form a Moore family that has a minimal element.

The following corollary of subject reduction justifies the title of this paper: we do track the ingredients of IoT data. The first item makes it evident that our analysis determines whether the value of a term may indeed be used along the computations of a system, and clarifies the role of the component Θ; the second item guarantees that κ predicts all the possible inter-node communications.

Corollary 1.

- *Let $N \xrightarrow{E_1,\ldots,E_n}_\ell N'$ denote a reduction in which all E_i are evaluated at node ℓ. If $(\hat{\Sigma}, \kappa, \Theta) \models N$ and $N \xrightarrow{E_1,\ldots,E_n}_\ell N'$ then $\forall k \in [0, n]$ it is $([\![E_k]\!]^i_{\Sigma^i_\ell})_{\downarrow_2} \in \Theta(\ell)$.*
- *Let $N \xrightarrow{\langle\langle v_1,\ldots,v_n\rangle\rangle}_{\ell_1,\ell_2} N'$ denote a reduction in which the message sent by node ℓ_1 is received by node ℓ_2. If $(\hat{\Sigma}, \kappa, \Theta) \models N$ and $N \xrightarrow{\langle\langle v_1,\ldots,v_n\rangle\rangle}_{\ell_1,\ell_2} N'$ then it holds $(\ell_1, \langle\langle \hat{v}_1, \ldots, \hat{v}_n\rangle\rangle) \in \kappa(\ell_2)$, where $\hat{v}_i = v_{i\downarrow_2}$.*

Back again to our example, we have that $1^{\ell_{cp}} \in \Theta(\ell_{cp})$, where $([\![1]\!]^1_{\Sigma^1_{\ell_{cp}}})_{\downarrow_2} = 1^{\ell_{cp}}$, and where v is the actual value received by the first sensor. Similarly, we have that $(\ell_{cp}, \langle\langle \hat{v}\rangle\rangle) \in \kappa(\ell_a)$, where $\hat{v} = v_{\downarrow_2}$.

Extending the analysis For simplicity, above we have presented a CFA that only tracks the ingredients of the data handled by IoT nodes. Now, we sketch a few possible extensions.

As it is, our analysis tracks the actual usage of sensor data through the component Θ. It is straightforward to also detect which actions of actuators are actually triggered. The result might suggest to use a simpler actuator if some of its actions are never exercised, or even to remove it if it is never used. Technically, a new analysis component α suffices, that for every actuator j collects the actions γ triggered by the control process in the node ℓ. Then, one has only to change the rule for the command to the actuator, as follows:

$$\frac{\gamma \in \alpha_\ell(j) \wedge (\hat{\Sigma}, \kappa, \alpha, \Theta) \models_\ell P}{(\hat{\Sigma}, \kappa, \alpha, \Theta) \models_\ell \langle j, \gamma \rangle . P}$$

To improve the precision of our CFA, we can refine the abstract store by replacing it with the pair $\hat{\Sigma}_{in}, \hat{\Sigma}_{out}$, similarly to the treatment of side effects in [19]. This extension is more invasive, because it requires modifying the rules for accurately handling the store updates. We can obtain a further improvement of the precision by making the analysis more context-sensitive. In particular, an additional component can record the sequence of choices made in conditionals while traversing the node under analysis. One can thus obtain better approximations of the store or detect causal dependencies among the data sent by sensors and the actions carried out by actuators, as well as casuality among nodes.

5 Conclusions

This paper is a first step towards a formal design framework for IoT, which will support the definition of techniques and tools for certifying properties of IoT applications. We proposed the process calculus IoT-LySa, with primitive constructs to describe the activity of sensors and of actuators, and suitable primitives for managing the coordination and communication capabilities of smart objects. We equipped our calculus with a CFA that statically predicts the interactions among nodes, how data spread from sensors to the network, and how data are put together. We sketched how the result of the analysis can be exploited as the basis for checking and certifying various properties of IoT systems.

Besides the extensions mentioned at the end of Sect. 4, we plan to accurately investigate the exact complexity of the analysis and to implement it. We intend to address with our analysis security and privacy "since IoT deals not only with huge amount of sensitive data (personal data, business data, etc.) but also has the power of influencing the physical environment with its control abilities" [14]. In particular, we can assign specific confidentiality levels to sensors and nodes and by inspecting the result of the analysis, we can detect if nodes with a lower level can access data of entities with a higher level. Also, we will enrich our design framework with security policies, e.g. for access control. By tracking the actions of actuators as suggested in Sect. 4, one can predict if an actuator is maliciously triggered by an attacker, as happened in the recent attack performed through a vehicular infotainment network (`http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/`). For brevity, we neglected here the

cryptographic primitives LySA offers natively, although both the current operational semantics and the static analysis can easily be extended to cover them. An analysis that identifies where encryption and decryption are really needed would be very useful for designers of IoT systems, because cryptography is expensive since many smart devices have limited battery power. Beneficial to such an analysis may be the preliminary work on an enhanced version of IoT-LySA [5] that estimates costs of cryptographic primitives. Finally, in the IERC words: "there is still a lack of research on how to adapt and tailor existing research on autonomic computing to the specific characteristics of IoT" [14]. To contribute to these issues, we plan to extend our calculus with linguistic mechanisms and a verification machinery to deal with adaptivity in the style of [9].

To the best of our knowledge, only a limited number of papers addressed the specification and verification of IoT systems from a process calculi perspective, within the formal methods approach. The IoT-calculus [15] is one of the first proposals in this setting. It explicitly includes sensors and actuators, and smart objects are represented as point-to-point communicating nodes of heterogeneous networks. Differently from ours, their interconnection topology can vary at run time. The authors propose two notions of bisimilarity that capture system behaviour from the point of view of end-users and of the other devices. The timed process calculus `CIoT` [8] specifies physical and logical components, addresses both timing and topology constraints, and allows for node mobility. Furthermore, communications are either short-range or internet-based. The focus of this paper is mainly on an extensional semantics that provides a *fully abstract* characterisation of the proposed contextual equivalence.

Many design choices of the above-discussed proposals are similar to ours. The main difference is that our coordination model is based on a shared store à la Linda instead of a message-based communication à la π-calculus. Furthermore, differently from [15,8], we are here mainly interested in developing a design framework that includes a static semantics to support verification techniques and tools for certifying properties of IoT applications.

The calculi above and ours are built upon those previously introduced for wireless, sensor and ad hoc networks ([16,20,18] to cite only a few). In particular, the calculus in [18] is designed to model so-called broadcast networks, with a dynamically changing topology. It presents some features very similar to ours: an asynchronous local broadcast modality, while intra-node communication relies on a local tuple space. Also, the analysis of the behaviour of broadcast networks is done by resorting to a multi-step static machinery.

References

1. Abadi, M., Gordon, A.: A calculus for cryptographic protocols: The Spi calculus. In: Procs. of the 4th ACM Conference on Computer and Communications Security. pp. 36–47. CCS '97, ACM (1997)
2. Bodei, C., Brodo, L., Focardi, R.: Static evidences for attack reconstruction. In: Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 162–182. Springer (2015)

3. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Automatic validation of protocol narration. In: Computer Security Foundations Workshop (CSFW-16 2003). pp. 126–140. IEEE Computer Society (2003)
4. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. Journal of Computer Security 13(3), 347–390 (2005)
5. Bodei, C., Galletta, L.: Securing IoT communications: at what cost? In: Procs. HotSpot 2016 (V. Cortier, Ed.) (2016), http://www.loria.fr/~cortier/HotSpot2016/HotSpot2016-proceedings.pdf
6. Buchholtz, M., Nielson, H.R., F. Nielson, F.: A calculus for control flow analysis of security protocols. International Journal of Information Security 2(3), 145–167 (2004)
7. Carriero, N., Gelernter, D.: A computational model of everything. Commun. ACM 44(11), 77–81 (2001)
8. Castiglioni, V., Lanotte, R., Merro, M.: A semantic theory for the internet of things. CoRR abs/1510.04854 (2015)
9. Degano, P., Ferrari, G.L., Galletta, L.: A Two-Component Language for Adaptation: Design, Semantics, and Program Analysis. IEEE Transactions on Software Engineering TSE. In press (2016)
10. Elejoste, P., Perallos, A., Chertudi, A., Angulo, I., Moreno, A., Azpilicueta, L., Astrain, J., Falcone, F., Villadangos, J.E.: An easy to deploy street light control system based on wireless communication and led technology. Sensors 13(5), 6492–6523 (2013)
11. Escolar, S., Carretero, J., Marinescu, M., Chessa, S.: Estimating energy savings in smart street lighting by using an adaptive control system. IJDSN 2014 (2014)
12. Gelernter, D.: Generative communication in linda. ACM Trans. Program. Lang. Syst. 7(1), 80–112 (1985)
13. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. 13(1), 124–149 (1991)
14. IERC: The Internet of Things 2012 – New Horizons (2012), http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2012_WEB.pdf
15. Lanese, I., Bedogni, L., Felice, M.D.: Internet of things: a process calculus approach. In: Procs of the 28th Annual ACM Symposium on Applied Computing, SAC '13. pp. 1339–1346. ACM (2013)
16. Lanese, I., Sangiorgi, D.: An operational semantics for a calculus for wireless systems. Theor. Comput. Sci. 411(19), 1928–1948 (2010)
17. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Inf. Comput. 100(1), 1–40 (1992)
18. Nanz, S., Nielson, F., Nielson, H.R.: Static analysis of topology-dependent broadcast networks. Inf. Comput. 208(2), 117–139 (2010)
19. Nielson, H.R., Nielson, F.: Flow logic: A multi-paradigmatic approach to static analysis. In: The Essence of Computation, Complexity, Analysis, Transformation. LNCS, vol. 2566, pp. 223–244. Springer (2002)
20. Singh, A., Ramakrishnan, C.R., Smolka, S.: A process calculus for mobile ad hoc networks. Sci. Comput. Program. 75(6), 440–469 (2010)