Internet of Things: A Process Calculus Approach*

Ivan Lanese
Focus Team, University of Bologna/INRIA, Italy
lanese@cs.unibo.it

Luca Bedogni
University of Bologna, Italy
lbedogni@cs.unibo.it

Marco Di Felice
University of Bologna, Italy
difelice@cs.unibo.it

ABSTRACT

This paper presents a process calculus specifically designed to model systems based on the Internet of Things paradigm. We define a formal syntax and semantics for the calculus, and show how it can be used to reason about relevant examples. We also define two notions of bisimilarity, one capturing the behavior seen by the end user of the system, and one allowing compositional reasoning.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed Programming*; D.2.4 [**Software/Program Verification**]: Formal Methods; F.3.2 [**Theory of Computation**]: Semantics of Programming Languages—*Process models*

General Terms

Languages, Theory, Verification

Keywords

Internet of Things, Process calculi, Semantics

1. INTRODUCTION

In the last years, the increasing popularity of portable devices equipped with heterogeneous wireless access technologies has constituted one of the key reasons of the success of the Mobile Internet worldwide. According to the recent previsions, the number of mobile Internet accesses is expected to overcome the fixed counterpart in 2013, and to increase to 50 billions by the end of 2020[1]. At the same time, the intrinsic pervasiveness of end-user devices (e.g. smartphones, tablets, etc.), and their possible integration with environmental devices (e.g. sensors and RFID tags), are opening new horizons for mobile computing, shifting the attention from a network of "end users" to a network of "objects" that autonomously interact and cooperate to reach common goals. This emerging paradigm, usually referred to as the *Internet of Things* (IoT) [1, 3], is envisioned to enable a large number of exciting applications in different domains, including transportation systems, health care and domotics. In a typical IoT scenario, multiple heterogeneous devices (e.g. sensors, smartphones, actuators, etc.) can communicate through short-range wireless communication technologies (e.g. Zigbee, Wifi, NFC, etc.), share context-related data (e.g. the value of a sensor), share resources (e.g. the Internet access of a smartphone), and enable the implementation of distributed applications in a seamless way, through the aggregation of different services offered by the environment [10]. What differentiates the IoT paradigm from other network paradigms is the automatic communication of the devices, which collect pieces of information from various sources and merge them to provide new services to the end users. In fact, in the IoT paradigm, users' interaction should be minimal, only asking for specific services that the devices should understand, build and provide to the end users [14]. At present, research on IoT is evolving mainly in two complementary directions. On the one hand, different IoT architectures and prototype implementations have been proposed for small-scale scenarios, like smart-home environments [21], leading to a proliferation of IoT definitions for different domains. On the other hand, middleware solutions have been proposed to enable service composition and data sharing, by abstracting from the heterogeneity of devices and technologies in use [1, 17]. However, there is still lack of research on the modeling and validation of IoT systems through formal methodologies that might allow to model the interactions among the system components (e.g. sensors, actuators), and to verify the correctness of the network deployment before its practical implementation. A straightforward utilization of these techniques is for *model-checking*, i.e. to assess whether the current network deployment can guarantee the expected behavior of an IoT application. However, they can also be an important aid for *network planning*, for instance to decide the equivalence or not of different network deployments to support a given IoT application.

In this paper, we propose a first contribution in the area of formal methods for IoT systems, by defining the *IoT-calculus*, a process calculus for modeling IoT systems. Process calculi have been successfully used to model distributed and mobile systems (see, e.g., the π-calculus [16]). However, to better describe systems based on a particular paradigm,

*Partly funded by the ANR-11-INSE-007 project REVER.
[1]Source: IDATE, June 2011.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

dedicated calculi are needed. This is proved by the different approaches in the fields of wireless sensor networks [12, 13] or service oriented computing systems (see, e.g., the survey in [5]). This paper follows the same approach, targeting systems based on the IoT paradigm. Devising a calculus for modeling a new paradigm requires understanding and distilling in a clean algebraic setting the main features of the paradigm. Our calculus captures a few features of IoT, i.e. the partial topology of communications and the interaction between sensors, actuators and computing processes to provide useful services to the end user. To reason on these emerging services we propose a new definition of bisimilarity called *end-user bisimilarity* to highlight the fact that it equates systems indistinguishable from the point of view of the end user, and a more standard form of bisimilarity to enable *compositional reasoning*, equating systems indistinguishable from the point of view of other devices.

Both the calculus definition and the presentation of the paper are aimed at making the calculus understandable also to non experts as far as possible. For instance, we avoid full restriction (and the related burden of scope extrusion) by using an opaque form of nodes when needed. This and other extensions of the core calculus are isolated in Section 4 so to keep the core calculus presented in Sections 2 and 3 as simple as possible.

Structure of the work. Section 2 describes the syntax and the intuition underlying the core IoT-calculus. Section 3 describes the semantics of the calculus. Section 4 presents extensions dealing with specific aspects such as broadcast communication or mobility control. Section 5 is devoted to the analysis of the observational semantics of the IoT-calculus. Section 6 discusses related work while Section 7 concludes the paper.

2. THE IOT-CALCULUS

We want to model systems that follow the Internet of Things paradigm. Those systems are composed by *communicating nodes*. Each node may have computational capabilities, including running processes, capability of communicating with other nodes and/or with the environment.

Communication among nodes is point-to-point and partial: a node may communicate only with nodes inside its transmission area. Nodes may move, thus changing the connectivity. We can see any IoT system as a graph whose nodes are labeled by information on the node capabilities and whose edges model communication *links*. For simplicity of representation, we use a term representation of the form

$$L \vdash N$$

where L describes links between nodes and N describes nodes themselves. We consider here bidirectional links, while unidirectional links are briefly discussed in Section 4.3.

A node is a physical entity with communication capabilities. Each node has a unique name n and a body B. The body B contains a parallel composition of up to three kinds of elements: *processes*, representing ongoing computations, *sensors*, and *actuators*. A sensor is a device reading a particular piece of information from the environment and making it available to the rest of the system. We consider simple basic sensors defined just by a name (e.g., "Temperature Sensor TS340") and a value (e.g., 30 degrees). More complex sensors may be modeled by nodes including one or more basic sensors and a process managing them (see Example 4).

$\mathcal{S} ::=$	$L \vdash N$	System
$L ::=$	$n \leftrightarrow m$	Link
	$L \mid L$	Parallel Composition
$N ::=$	$n[B]$	Node
	$N \mid N$	Parallel Composition
$B ::=$	$B \mid B$	Parallel Composition
	$(s \leftarrow v)$	Sensor
	$(a \rightarrow v)$	Actuator
	P	Running Process
$P ::=$	0	Nil
	$\overline{c}\langle v\rangle.P$	Output
	$c(x).P$	Input
	$!c(x).P$	Replicated Input
	$(x) \leftarrow s.P$	Read from Sensor
	$v \rightarrow a.P$	Write to Actuator

Figure 1: Syntax of the core IoT-calculus.

Similarly, an actuator is a device providing a value to the environment, such as a display, an engine (here the value is the current speed), and so on. Actuators too are defined by a name and a value, but now the value is decided by the system and is propagated to the environment. Sensors and actuators are static, i.e. they are never created nor destroyed, since they correspond to physical entities. Running processes provide computational and communication capabilities. We use for processes a π-calculus [16] style formalism featuring channel-based communication, sensor read and actuator write.

The syntax of the core IoT-calculus is in Figure 1. It relies on infinite mutually disjoint sets of names $\mathcal{N}$ for node names, ranged over by $n, m, \ldots$; $\mathcal{C}$ for channel names, ranged over by $c, \ldots$; $\mathcal{S}$ for sensor names, ranged over by $s, \ldots$; and $\mathcal{A}$ for actuator names, ranged over by $a, \ldots$. Values, ranged over by $v, \ldots$, include data values, and all names but node names. For simplicity we consider just integers, booleans and strings as data values, but other kinds of data values can easily be added. We use $x, \ldots$ for variables. Terms $c(x).P$, $!c(x).P$ and $(x) \leftarrow s.P$ bind variable x in the continuation P.

For ensuring that values and variables are used in a suitable way (e.g., it is not reasonable to use an integer as channel for a communication) one could use a simple type system. The definition of such a type system is quite standard [16, Chapter 6], thus we will not describe it.

We assume all parallel composition operators $\mid$ to be associative and commutative (this can be formalized using a structural congruence following [16]).

The syntax of systems and nodes has been commented above. Processes include process 0, that performs no action (for simplicity, we may remove trailing or parallel 0s), output $\overline{c}\langle v\rangle.P$ of value v over channel c with continuation P, the corresponding input $c(x).P$ receiving a value on channel c and replacing it for variable x in the continuation P, and replicated input $!c(x).P$ acting as a parallel composition of infinitely many inputs. We write $\overline{c}\langle\rangle$ for $\overline{c}\langle 0\rangle$ and $c().P$ for $c(x).P$ if x does not occur in P (similarly for replicated input). Read $(x) \leftarrow s.P$ reads the current value of sensor s into variable x. Symmetrically, write $v \rightarrow a.P$ writes the value v into actuator a, thus making value v visible for the environment.

Example 1. We want to model a simple system where a

smartphone reads the temperature from a sensor and sends its value to a display. The system is thus composed by three nodes: (i) a node n_s containing a temperature sensor s_t, written as $S = n_s[(s_t \leftarrow 20)]$, (ii) a node n_a containing a display a_d, written $A = n_a[(a_d \rightarrow 15)]$, and (iii) a node n_t representing the smartphone, written $T = n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d]$. In this case we assume that the sensor s_t is sensing the temperature 20 (degrees), the display a_d is showing the temperature 15 (degrees), and the smartphone provides a functionality *disp* that upon invocation checks the temperature on the sensor and updates the temperature shown by the display. The whole system is the parallel composition of the nodes above, where the smartphone n_t can communicate with both the sensor at n_s and the actuator at n_a:

$$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash S \mid T \mid A$$

As it is now, the system provides an interesting functionality, but this functionality is never triggered. We can add a running process which triggers such a functionality one-shot.

$$T = n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid \overline{disp}\langle\rangle]$$

If we want to trigger such a functionality repeatedly, we can exploit replicated input:

$$T = n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle]$$

Here a message on *trig* triggers the replicated input, which triggers functionality *disp* and then recreates the message on *trig*. The detailed behavior will be clearer after the definition of the semantics.

Example 2. In the Internet of Things world, a device may have no a priori knowledge of which are the sensors/actuators available in his current location. A possible solution is to have a directory service which knows about the available sensors/actuators. When needed, a device may contact the directory to know about the available sensors/actuators. We can model such a directory as follows:

$$D = n_d[!list(ans).\overline{ans}\langle s1\rangle.\ldots.\overline{ans}\langle sn\rangle.\overline{ans}\langle end\rangle \mid !descr(\langle ans, s\rangle).\overline{ans}\langle find(s)\rangle]$$

The directory provides two functionalities, a functionality *list* and a functionality *descr*. The functionality *list* sends the names of the sensors he knows of to the caller using the channel *ans* provided by the caller. The value *end* is used to specify that there are no more available sensors. The functionality *descr* receives a channel *ans* and a sensor name s (we use tuples of values to simplify the modeling) and sends over *ans* a description of the sensor. Here we assume a function $find$ computing the description of the sensor (e.g., a string) from its name.

This example uses quite rough communication patterns, and can be refined in many respects. For instance, the directory could manage a dynamic list of services and take the required information from there. However, at this stage, we just wanted to show how common IoT patterns can be modeled.

3. SEMANTICS

We define here the semantics of the IoT-calculus. Since systems based on the Internet of Things paradigm are interacting and open, we describe the semantics of the calculus using a labeled transition system, able to highlight interactions with the environment and enabling the definition of observational equivalences such as bisimilarity.

A transition is a triple of the form $S \xrightarrow{\alpha} S'$ where S may be a node body B, a group of nodes N or a system $L \vdash N$, and α is the transition label. We use $\alpha, \ldots$ to range over labels.

Definition 1. The labeled transition system semantics of the IoT-calculus is the smallest set of transitions closed under the rules in Figure 2. In the figure $\overline{\alpha}$ denotes the action complementary to α. In particular, we have $\overline{c}\langle v\rangle = \overline{c(v)}$, $s \leftarrow v = \overline{(v \leftarrow s)}$ and $a \rightarrow v = \overline{(v \rightarrow a)}$. We assume $\overline{\overline{\alpha}} = \alpha$.

We distinguish two kinds of labels: communication labels and notification labels. Communication labels are used to make different components of the system interact. This is the case, e.g., of $\overline{c}\langle v\rangle$ or $v \leftarrow s$. Notification labels instead are just used to interact with the external environment. Notification labels can be recognized since they syntactically include a τ, e.g., in τ or $\tau : n \not\leftrightarrow m$. Transitions labeled by notification labels describe the evolution of a full system deployed in its external environment. The distinction between the two kinds of labels will be used later on to define end-user bisimilarity.

Rules from `Out` to `CoWt` execute basic communications by making the communication information available in the label. Rules `Out` and `In` are for communication over channels. The input prefix guesses the received value and replaces it for the formal variable. This is the standard approach in the so-called early semantics (see, e.g., π-calculus early semantics in [16]). Rule `RIn` is for replicated input. A replicated input upon receiving a value spawns a new thread to execute the continuation instantiated by the received value, while the input remains available to read further outputs. Replicated input allows also to specify infinite behaviors, as for the *trig* functionality in Example 1. `Rd` and `CoRd` allow to read the value of a sensor and `Wt` and `CoWt` allow to write a value to an actuator. Rule `Sens` allows a sensor to read a new value from the external environment. This transition uses a notification label. This highlights the fact that this transition is not used to interact with other components of the system, but only with the external environment. Similarly, rule `Act` makes the value of the actuator available to the external environment. The semantics of sensors and actuators is somehow symmetric, with sensors updated by the environment and read by the system, and actuators updated by the system and read by the environment. Rule `Par` allows an action to propagate through parallel composition inside a node. Rule `IntSynch` allows two complementary (communication) labels inside the same node to synchronize. For instance, in Example 1, the output on *trig* is received by the replicated input inside the same node n_t. The two synchronizing labels must have the form α and $\overline{\alpha}$, to ensure that unrelated actions will not synchronize and that the guess on the value in input is correct w.r.t. the matching output. Actions performed inside a node should be lifted to the node level. This is done by rule `Node` for communication labels and by rule `NodePass` for notification labels. The name n of the node performing the communication is added to communication labels, i.e., α becomes $n : \alpha$, while notification labels are unchanged. Rule `NodePar` allows actions to propagate among nodes. Rule `Synch` allows communication between different nodes. It is similar to rule `IntSynch`, but it deals with labels

$$(\textsc{Out})\ \overline{c}\langle v\rangle.P \xrightarrow{\overline{c}\langle v\rangle} P \qquad (\textsc{In})\ c(x).P \xrightarrow{c(v)} P\{^{v}/_{x}\}$$

$$(\textsc{RIn})\ !c(x).P \xrightarrow{c(v)} !c(x).P \mid P\{^{v}/_{x}\}$$

$$(\textsc{Rd})\ (x) \leftarrow s.P \xrightarrow{v \leftarrow s} P\{^{v}/_{x}\}$$

$$(\textsc{CoRd})\ (s \leftarrow v) \xrightarrow{(s \leftarrow v)} (s \leftarrow v) \qquad (\textsc{Wt})\ v \rightarrow a.P \xrightarrow{v \rightarrow a} P$$

$$(\textsc{CoWt})\ (a \rightarrow v) \xrightarrow{(a \rightarrow v')} (a \rightarrow v')$$

$$(\textsc{Sens})\ (s \leftarrow v) \xrightarrow{\tau : s \leftarrow v'} (s \leftarrow v')$$

$$(\textsc{Act})\ (a \rightarrow v) \xrightarrow{\tau : a \rightarrow v} (a \rightarrow v) \qquad (\textsc{Par})\ \frac{B \xrightarrow{\alpha} B'}{B \mid C \xrightarrow{\alpha} B' \mid C}$$

$$(\textsc{IntSynch})\ \frac{B \xrightarrow{\alpha} B' \quad C \xrightarrow{\overline{\alpha}} C'}{B \mid C \xrightarrow{\tau} B' \mid C'}$$

$$(\textsc{Node})\ \frac{B \xrightarrow{\alpha} B' \quad \alpha \notin \{\tau, \tau : s \leftarrow v, \tau : a \rightarrow v\}}{n[B] \xrightarrow{n:\alpha} n[B']}$$

$$(\textsc{NodePass})\ \frac{B \xrightarrow{\alpha} B' \quad \alpha \in \{\tau, \tau : s \leftarrow v, \tau : a \rightarrow v\}}{n[B] \xrightarrow{\alpha} n[B']}$$

$$(\textsc{NodePar})\ \frac{N \xrightarrow{\alpha} N'}{N \mid M \xrightarrow{\alpha} N' \mid M}$$

$$(\textsc{Synch})\ \frac{N \xrightarrow{n:\alpha} N' \quad M \xrightarrow{m:\overline{\alpha}} M' \quad n \leftrightarrow m}{N \mid M \xrightarrow{\tau} N' \mid M'}$$

$$(\textsc{Judg})\ \frac{N \xrightarrow{\alpha} N'}{L \vdash N \xrightarrow{\alpha} L \vdash N'}$$

$$(\textsc{Conn})\ L \vdash N \xrightarrow{\tau : n \leftrightarrow m} L \mid n \leftrightarrow m \vdash N$$

$$(\textsc{Disc})\ L \mid n \leftrightarrow m \vdash N \xrightarrow{\tau : n \not\leftrightarrow m} L \vdash N$$

Figure 2: Semantics of the core IoT-calculus.

including node names, and it checks that the communicating nodes are actually connected. The condition $n \leftrightarrow m$ requires that the link $n \leftrightarrow m$ occurs in the judgment (a more precise but more complex formalization would add the information to the label and check it at the judgment level). Rule Judg propagates an action to the judgment level. Rule Conn adds a new connection to the judgment, while rule Disc removes a connection from the judgment. The label makes the change visible to the environment (or, better, forced by the environment, as we will see in the following). Note that the semantics allows multiple links between the same nodes. They may represent, e.g., links based on different technologies. In the core calculus multiple links are redundant (but nodes are not disconnected till they are all removed), while this is not the case, e.g., in the extension with costs for communication (see Section 4.6).

Example 3. A simple computation of the system described by Example 1 is in Figure 3. All the labels in the computation are notification labels, thus the computation shows a (possible) behavior of a full system deployed in its environment. In the computation, internal communications in the smartphone and communications between smartphone and sensor and smartphone and actuator propagate the value sensed by the sensor to the actuator. The new value of the actuator is propagated to the environment in the last label $\tau : a_d \rightarrow 20$ (the transition leaves the process unchanged). Note that the transition propagating the value of the actuator is always enabled: this holds in any system including an actuator, and it means that the environment can always check the value of actuators. Similarly, transitions updating the value of sensors are always enabled (if at least a sensor exists) since the environment can always change the value sensed by a sensor. Finally, transitions connecting and disconnecting nodes are always enabled (unless mobility control is used, see Section 4.5) meaning that the environment may move nodes so to create or destroy connections.

4. EXTENSIONS

Previous sections presented the core of the IoT-calculus, which is kept minimal to highlight the most relevant features while keeping the complexity reasonable. However extensions may be needed to simplify modeling of specific systems or to give better control on particular aspects. We will describe some possible extensions below, in increasing order of complexity. The first two extensions consider classical process calculi operators, while the other extensions tackle typical features of IoT systems.

4.1 Choice

Nondeterminstic choice is frequently used in process calculi, and can be defined for IoT processes in a standard way, see, e.g., [16]. However, it is needed more for axiomatizations than for modeling, thus we avoid it here. Deterministic choice, i.e. if-then-else, instead can help modeling. We can add the operator `if` v `then` P `else` Q `endif` to IoT processes, with the semantics below.

$$(\textsc{IfT})\ \texttt{if}\ true\ \texttt{then}\ P\ \texttt{else}\ Q\ \texttt{endif} \xrightarrow{\tau} P$$

$$(\textsc{IfF})\ \texttt{if}\ false\ \texttt{then}\ P\ \texttt{else}\ Q\ \texttt{endif} \xrightarrow{\tau} Q$$

Note that `if` v `then` P `else` Q `endif` is blocked if v is not a boolean value.

4.2 Restriction

In the calculus described till now, all interactions are visible from the outside. For instance, an internal communication in a node can also be received by other nodes (if they are in the communication range). In practice, this is not frequent, since different means are used for internal and external communication. A general solution to this problem can be obtained using the restriction operator as in π-calculus [16]. However, this is more powerful and more complex than what is normally needed here. Thus we present a simple solution that equips nodes with a set of closed names, i.e. names which are used only inside the node and not outside. The syntax for such a node is $n[B]^{C}$ where C is a set of

$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash n_s[(s_t \leftarrow 20)] \mid n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle] \mid n_a[(a_d \rightarrow 15)] \xrightarrow{\tau}$

$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash n_s[(s_t \leftarrow 20)] \mid n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{disp}\langle\rangle.\overline{trig}\langle\rangle] \mid n_a[(a_d \rightarrow 15)] \xrightarrow{\tau}$

$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash n_s[(s_t \leftarrow 20)] \mid n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid (x) \leftarrow s_t.x \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle] \mid n_a[(a_d \rightarrow 15)] \xrightarrow{\tau}$

$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash n_s[(s_t \leftarrow 20)] \mid n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid 20 \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle] \mid n_a[(a_d \rightarrow 15)] \xrightarrow{\tau}$

$n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \vdash n_s[(s_t \leftarrow 20)] \mid n_t[!disp().(x) \leftarrow s_t.x \rightarrow a_d \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle] \mid n_a[(a_d \rightarrow 20)] \xrightarrow{\tau:a_d \rightarrow 20}$

Figure 3: Sample computation.

names. To integrate these "opaque" nodes into the semantics we just have to add the condition $n(\alpha) \cap C = \emptyset$ to rule Node, where $n(\alpha)$ is the set of names occurring in α. See Example 7 for an application of this extension.

4.3 Unidirectional links

The core IoT-calculus allows only for bidirectional links. Unidirectional links can be represented as $n \rightarrow m$, meaning that n may send messages to m. These messages may be messages over channels from n to m, or write to actuators in m, or sensors in n sending their value to processes in m. A bidirectional link $n \leftrightarrow m$ can be seen as $n \rightarrow m \mid m \rightarrow n$. In order to accommodate unidirectional links in the semantics we have just to change rule Synch to check that a link in the required direction is available.

4.4 Broadcast

Till now we just considered point-to-point communication, however many IoT systems may exploit also broadcast communication. A broadcast output $\overline{c}!\langle v\rangle$ is received by all the broadcast inputs $c?(x)$ it reaches. The semantics of broadcast has been studied in a fully connected setting in [6, 4], and heavily studied in a partial connected setting in the field of wireless networks, see e.g. [15, 7]. Techniques similar to the ones used in wireless systems can be used to define the semantics of broadcast in the IoT-calculus. Essentially, a broadcast input interacts with a broadcast output and the result is again a broadcast output. At judgment level a broadcast output becomes a τ action.

Example 4. We show here how broadcast can be used to model a way of discovering sensors nearby different from the one in Example 2. Here we assume that sensors are slightly more complex, with the capability of receiving broadcast requests and answering by advertising themselves. Assuming the channel *discover* is used for discovery, such a sensor can be written as:

$$n_s[(s_t \leftarrow 25) \mid !discover?(x).\overline{x}\langle\langle s_t, \text{"Temperature sensor"}\rangle\rangle]$$

A device looking for sensors can send a broadcast request on channel *discover* with the name of the channel to be used for answering as a parameter. It can then choose among the answers the sensor more suitable for its need (we can easily imagine that the sensor provides additional information), and then interact directly with the sensor.

4.5 Mobility control

The core calculus allows for full mobility: at any moment any link can disappear, and any link can appear. In many cases this is not reasonable: some links may be always available, while others impossible to establish (e.g., if the communicating partners are not mobile). Since all the mobility information is available in the computation labels, one may impose these constraints by restricting the attention to a suitable subset of computations, e.g., requiring that the label $\tau : n \not\leftrightarrow m$ disconnecting n from m never occurs ensures that connection $n \leftrightarrow m$ is persistent (if it exists at the beginning, or after its creation).

However, simple restrictions as above can also be hardwired in the syntax and semantics of the calculus. E.g., one can extend the syntax of links with permanent links $n \leftrightarrow^{\bullet} m$, which act as $n \leftrightarrow m$ but cannot be destroyed by rule Disc. Similarly one can add forbidden links $n \not\leftrightarrow^{\bullet} m$ which do not allow communication but forbid to add links $n \leftrightarrow m$ using rule Conn (the rule should be changed to perform the check). See Example 5 for an application of permanent links.

4.6 Cost of actions

A main feature of the Internet of Things (and of other kinds of networks) is that computations and communications may be costly. Costs may refer to energy, bandwidth or also economic cost. Understanding how to model all these kinds of cost, and how to exploit them for analysis, e.g., to understand which communication patterns can be executed within given bounds on resources, is a complex problem, and we plan to devote the due attention to it in future work. However, we want to remark here that the IoT-calculus is suitable to help in this kind of analysis. We give two examples in this direction.

The first example refers to energy cost of computation and communication inside nodes. We want to model the fact that each node has a finite amount of energy e, and each action consumes energy. When energy is exhausted, i.e. $e = 0$, the node is no more operational. To model this, note that all the actions performed by a node traverse the node boundary using rules Node and NodePass. Thus we may add to node terms information about the energy level e in the node, and rules Node and NodePass can be extended to check that e is positive, and to decrease it according to the performed action. One can also imagine a special action modeling battery recharge restoring e to the initial value.

A different kind of cost is strictly related to communication. Assume that communication channels come with a cost that has to be paid for communication. This is for instance the case of telephone networks. This cost can be associated to links, and added to the τ label resulting from the communication. In this way the labels of a computation show its cost, and suitable definitions of bisimulation can allow to distinguish computations performing the same task with different costs. Note that in this setting different links between the same nodes are not redundant if they have different costs.

5. BISIMULATION

A main reason to define a labeled transition system is to allow to check systems for equivalence using coinductive techniques. For IoT systems we distinguish two forms of

equivalence: one from the point of view of the end user (which acts as an environment for the system) that we call *end-user bisimilarity*, and one from the point of view of other components interacting with the system that we simply call *bisimilarity*. We will show that the second one is compositional, thus enabling compositional reasoning on complex systems.

The end user of the system can just interact with the system by providing values to sensors and by checking values provided by actuators. We assume that the end user can also move nodes thus creating or removing connections. End user bisimilarity is designed to compare the overall behavior of fully specified systems. In the π-calculus tradition one would require all the names to be bound. Here however we want to avoid the burden of defining restrictions and the related semantic subtleties, and we just force the system to compute on its own, i.e. to use only notification labels.

As standard, for both end-user bisimilarity and bisimilarity we consider two variants: a strong one requiring perfect match of actions, and a weak one allowing different internal computations provided that the effect on the environment is the same. To define weak bisimilarity we need first to define weak transitions.

Definition 2. We denote with $\Rightarrow$ the reflexive and transitive closure of $\xrightarrow{\tau}$. We write $\stackrel{\alpha}{\Rightarrow}$ for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$ if $\alpha \neq \tau$, for $\Rightarrow$ if $\alpha = \tau$.

We start by defining end-user bisimilarity.

Definition 3. A strong (resp. weak) end-user bisimulation is a relation $\mathcal{R}$ among systems such that $L_1 \vdash N_1 \ \mathcal{R} \ L_2 \vdash N_2$ implies:

- if $L_1 \vdash N_1 \xrightarrow{\alpha} L_1' \vdash N_1'$ where α is a notification label then $L_2 \vdash N_2 \xrightarrow{\alpha} L_2' \vdash N_2'$ (resp. $L_2 \vdash N_2 \stackrel{\alpha}{\Rightarrow} L_2' \vdash N_2'$) and $L_1' \vdash N_1' \ \mathcal{R} \ L_2' \vdash N_2'$;
- viceversa, with the challenge from $L_2 \vdash N_2$.

Strong (resp. weak) end-user bisimilarity, denoted $\sim_e$ (resp. $\approx_e$), is the largest strong (resp. weak) end-user bisimulation.

Two systems with different connections between nodes are not end-user bisimilar, since computations performing actions $\tau : n \not\leftrightarrow m$ allow to count the number of connections between nodes n and m. Thus we will use end-user bisimilarity to compare systems with the same connections.

Example 5. Consider the system in Example 1 and a variant of it where the smartphone is node n_{t1} instead of node n_t. To avoid to distinguish them simply because of connections, assume that in both of them the available links are: $L = n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s \mid n_a \leftrightarrow n_{t1} \mid n_{t1} \leftrightarrow n_s$.

The two systems however are not end-user bisimilar, since the first one can disconnect the link $n_a \leftrightarrow n_{t1}$, and then update the value of the display, while the second one cannot match this computation.

However, they become end-user bisimilar if we consider persistent links $L' = n_a \leftrightarrow^\bullet n_t \mid n_t \leftrightarrow^\bullet n_s \mid n_a \leftrightarrow^\bullet n_{t1} \mid n_{t1} \leftrightarrow^\bullet n_s$. This is easily proved by considering as bisimulation the relation $\mathcal{R}$ below:

$$\mathcal{R} = \{(L'' \vdash N, L'' \vdash N\{^{n_{t1}}/_{n_t}\}) \mid L' \vdash S \mid T \mid A \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} L'' \vdash N \wedge \alpha_1 \ldots \alpha_n \text{ notification labels}\}$$

Actually, with persistent connections the user cannot distinguish whether there is just one smartphone providing the update functionality or there are many of them:

$$L' \vdash S \mid T \mid A \sim_e L' \vdash S \mid T \mid T\{^{n_{t1}}/_{n_t}\} \mid A$$

The properties emerging from the example above can be generalized to suitable axiomatic laws correct w.r.t. end-user bisimilarity. While studying the axiomatization of the IoT-calculus is out of the scope of the present paper we describe below a few rules inspired by the example above. They are valid only for fixed topologies. In the rules below N has no node named n and $L_{n,m}$ denotes a topology where n and m have the same connections, and n is connected to m.

$$L \vdash N \quad \sim_e \quad (L \vdash N)\{^n/_m\} \tag{1}$$

$$L_{n,m} \vdash n[B \mid B'] \quad \sim_e \quad L_{n,m} \vdash n[B] \mid m[B'] \tag{2}$$

The first rule shows that names of nodes are immaterial if the topology is fixed, while the second one shows that the body of a node can be split into two distinct, connected nodes.

Example 6. The example above shows the use of strong end-user bisimilarity. However, strong end-user bisimilarity distinguishes systems which differ only for the amount of computation done to reach a goal. In most of the cases these systems can be considered equivalent. This form of equivalence is formalized by weak end-user bisimilarity.

We present now a scenario which is equivalent to the smartphone scenario above according to weak end-user bisimilarity but not according to strong end-user bisimilarity. In the new scenario a smartphone n_{t1} reads the value of the sensor and sends it to a second smartphone n_{t2}, which updates the display. The system can be modeled as follows:

$$n_s \leftrightarrow n_{t1} \mid n_{t1} \leftrightarrow n_{t2} \mid n_{t2} \leftrightarrow n_a \vdash S \mid n_{t1}[!disp().(x) \leftarrow s_t.\overline{comm}\langle x\rangle \mid !trig().\overline{disp}\langle\rangle.\overline{trig}\langle\rangle \mid \overline{trig}\langle\rangle] \; n_{t2}[!comm(x).x \to a_d] \mid A$$

We assume for both this system and the one with a unique smartphone a static topology and the same connections. According to strong end-user bisimilarity this system is not equivalent to the previous one. This can be proved by noting that in this system at least 3 τ steps are needed to propagate a value from the sensor to the actuator (read of the sensor, communication between smartphones, write to the actuator), while in the previous case 2 τ steps were enough.

However the two systems are equivalent according to weak end-user bisimilarity. This means that if the observer cannot check how much computation and/or communication takes place, the two systems are indistinguishable. This kind of equivalence is reasonable in most of the cases.

End-user bisimilarity captures relevant aspects of IoT systems, but it is not compositional. This can be simply verified by noting that two systems with the same connectivity, the same sensors and with no actuators are equivalent. For instance $L \vdash S \sim_e L \vdash S \mid T$ where S and T are the sensor and the smartphone from the example above. It is clear that adding the actuator A to the two systems breaks the equivalence, since in the second case the actuator is updated while in the first one it is not. Compositionality is a desirable feature, since it allows to prove equivalence of large systems by proving equivalence of smaller ones.

We present now a different form of bisimilarity which is compositional, and which implies end-user bisimilarity. We simply call it bisimilarity.

Definition 4. A strong (resp. weak) bisimulation is a relation $\mathcal{R}$ among systems such that $L_1 \vdash N_1 \mathcal{R} L_2 \vdash N_2$ implies:

- if $L_1 \vdash N_1 \xrightarrow{\alpha} L_1' \vdash N_1'$ where α is a label then $L_2 \vdash N_2 \xrightarrow{\alpha} L_2' \vdash N_2'$ (resp. $L_2 \vdash N_2 \stackrel{\alpha}{\Rightarrow} L_2' \vdash N_2'$) and $L_1' \vdash N_1' \mathcal{R} L_2' \vdash N_2'$;
- viceversa, with the challenge from $L_2 \vdash N_2$.

Strong (resp. weak) bisimilarity, denoted $\sim$ (resp. $\approx$), is the largest strong (resp. weak) bisimulation.

It is clear from the definitions that bisimilarity is stronger than end-user bisimilarity.

PROPOSITION 1. *If $L \vdash N \sim L' \vdash N'$ then $L \vdash N \sim_e L' \vdash N'$. If $L \vdash N \approx L' \vdash N'$ then $L \vdash N \approx_e L' \vdash N'$.*

We have now to show that bisimilarity is compositional. First we have to define composition of IoT systems. We restrict composition to systems whose node parts have disjoint sets of node names.

Definition 5. The composition of two systems $L \vdash N$ and $L' \vdash N'$ such that N and N' have disjoint sets of node names is $L \mid L' \vdash N \mid N'$.

THEOREM 1. *If $L_1 \vdash N_1 \sim L_1' \vdash N_1'$, $L_2 \vdash N_2 \sim L_2' \vdash N_2'$ and $L_1 \vdash N_1$ and $L_2 \vdash N_2$ can be composed then $L_1 \mid L_2 \vdash N_1 \mid N_2 \sim L_1' \mid L_2' \vdash N_1' \mid N_2'$.*

PROOF SKETCH. We have to show that the relation:

$$\mathcal{R} = \{(L_1 \mid L_2 \vdash N_1 \mid N_2, L_1' \mid L_2' \vdash N_1' \mid N_2') \mid L_1 \vdash N_1 \sim L_1' \vdash N_1' \ \wedge \ L_2 \vdash N_2 \sim L_2' \vdash N_2' \wedge L_1 \vdash N_1 \text{ and } L_2 \vdash N_2 \text{ can be composed}\}$$

is a bisimulation. The proof is by coinduction. Assume $L_1 \mid L_2 \vdash N_1 \mid N_2 \xrightarrow{\alpha} I \vdash M$. Note that bisimilar systems have the same links, thus we have to show that $L_1 \mid L_2 \vdash N_1' \mid N_2' \xrightarrow{\alpha} I' \vdash M'$ with $I \vdash M \sim I' \vdash M'$. We have a case analysis on the rule used to derive the transition. For rule Disc the removed link belongs to either L_1 or L_2. Assume it belongs to L_1 (the other case is similar). Then $L_1 \mid L_2 \vdash N_1 \mid N_2 \xrightarrow{\tau:n \not\leftrightarrow m} I_1 \mid L_2 \vdash N_1 \mid N_2$. Similarly $L_1 \mid L_2 \vdash N_1' \mid N_2' \xrightarrow{\tau:n \not\leftrightarrow m} I_1 \mid L_2 \vdash N_1' \mid N_2'$. Since $L_1 \vdash N_1 \xrightarrow{\tau:n \not\leftrightarrow m} I_1 \vdash N_1$ and $L_1 \vdash N_1 \sim L_1 \vdash N_2$ we also have $I_1 \vdash N_1 \sim I_1 \vdash N_1'$ and thus $I_1 \mid L_2 \vdash N_1 \mid N_2 \sim I_1 \mid L_2 \vdash N_1' \mid N_2'$ as desired. Note, in fact, that transitions do not change the sets of nodes, thus the systems above can be composed. The proof is similar for rule Conn (the new connection can be added equivalently to the left or to the right link term). For rule Judg we have to show that also node terms satisfy the compositionality property. The proof is similar to the one for systems, and thus requires a case analysis on the rule used to derive the transition for node terms. For rule Synch if both the interacting nodes and the link they use belong to the same term the thesis follows easily. If two nodes in different terms interact via a link belonging to one of them then by inductive hypothesis the corresponding subterms can perform the same complementary actions, which can be composed again at the level of the composed system. The most tricky case is when two nodes from one term interact via a link provided by the other term. However the same result can be obtained by adding the same link to the term performing the interaction, performing the communication and then removing the link. Since we have already proved the thesis for all these actions, the thesis follows. The case of rule NodePar directly follows by induction. □

Example 7. Let us consider the smartphone T with repeated trigger from Example 1. Instead of using the auxiliary channel *trig* for generating triggers, one can use directly channel *disp*:

$$T' = n_t[!disp().\overline{disp}\langle\rangle.(x) \leftarrow s_t.x \rightarrow a_d \mid \overline{disp}\langle\rangle]$$

Smartphones T' and T are end-user bisimilar, but this gives no hint on their behavior when used as part of a complex IoT system. It is clear that they are not strong bisimilar, since they use a different amount of communication to reach the same goal. As they are, they are not weak bisimilar either, since T can perform communication on channel *trig* while T' can not. However, channels *trig* and *disp* are only meant to be used internally. If one transforms node n_t in the two systems into an opaque node as described in Section 4.2 choosing $C = \{trig, disp\}$ the two smartphones become weak bisimilar. Thus using Theorem 1 one can prove that complex systems using T are weak bisimilar to corresponding systems using T'. Then using Proposition 1 one can show that the systems are also weak end-user bisimilar. For instance, with $L = n_a \leftrightarrow n_t \mid n_t \leftrightarrow n_s$, we have:

$$L \vdash S \mid T \mid A \approx_e L \vdash S \mid T' \mid A$$

6. RELATED WORK

Initial research on IoT was mainly focused on the issues of traceability and addressability of objects equipped with Radio-Frequency IDentification (RFID) tags [2]. However, the RFID-centric vision of IoT has been progressively enlarged in these last years by including also sensors and mobile devices (e.g. smartphones). To this aim, the 6LoWPAN protocol (IPv6 over Low power Wireless Personal Area Networks) [22] has been proposed to guarantee the unique addressability of objects with limited processing capabilities. Moreover, middleware solutions have been investigated to support data aggregation and sharing over heterogeneous IoT network environments, abstracting from the specific software/hardware characteristics of each network device [18]. Among the most recent initiatives, we cite the Smart-M3 [9] middleware, based on an ontology-oriented design, and the SOCRADES [8] platform, that investigates the application of Service Oriented Architecture (SOA) principles to the IoT scenario. At the same time, different IoT architectures and prototype implementations have been proposed for small-scale scenarios, like smart-home environments [21]. In these cases, the modeling and validation of the network protocols and applications is performed by means of testbeds and discrete-event network simulators, such as OMNET++ [20] and NS3 [19]. However, the network simulation covers only a subset of the system behaviors due to the potential high number of parameters affecting the wireless environment. As far as we know this is the first work addressing formal methods for the modeling and validation of

IoT applications through an algebraic language. However, it takes inspiration from algebraic models of similar systems, in particular wireless systems. Algebraic models for wireless systems are normally based on broadcast on a partial topology, while we consider mainly point-to-point communication and we have sensors and actuators as explicit entities in our models. Among these models the nearest to ours is [7], which has a similar way of modeling connectivity. Other works are more oriented towards security, such as [15], analysis of probabilistic and timed behaviors, such as [12], or analysis of low level features such as interferences [11] or network deployment [13].

7. CONCLUSIONS

We see this paper as a first step in a quite unexplored area. In fact, while process calculi have been applied quite extensively to model various paradigms of distributed computation, such as wireless sensor networks [12, 13] or service oriented computing systems (see, e.g., the survey in [5]), we are not aware of other calculi devised for modeling Internet of Things systems. It is quite difficult to ensure that the formalization of a computational paradigm provided by a calculus is the best possible one. For instance, such an agreement has not been reached among the calculi in [5] after years of research. Thus we are not claiming that our formalization is the best possible one, and we will continue to refine and extend it in the future. We will also devote more attention to important aspects that we have just cited here. As already said, understanding the importance of costs for computation and communication is a main aim. Also, how to control mobility is an open problem which we inherit from models for wireless networks, and which is quite relevant here as shown by its impact on bisimilarity between systems. Also, from the point of view of equivalences, we have presented here a form of bisimilarity, end-user bisimilarity, which reminds a barbed equivalence, where barbs are mimicked by notification actions. Analyzing barbed equivalence and congruence in this setting, and their relation to bisimilarity, is interesting to clarify the abstract semantics of IoT systems.

8. REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] Auto-Id Labs. `http://www.autoidlabs.org/`.

[3] E. N. Barnhart and C. A. Bokath. Considerations for Machine-to-Machine communications architecture and security standardization. In *Proc. of Conference on Internet Multimedia Systems Architecture and Application 2011*, pages 1–6. IEEE Computer Society, 2011.

[4] R. Bruni and I. Lanese. PRISMA: A mobile calculus with parametric synchronization. In *Proc. of TGC'06*, volume 4661 of *LNCS*, pages 132–149. Springer, 2006.

[5] L. Caires, R. De Nicola, R. Pugliese, V. T. Vasconcelos, and G. Zavattaro. Core calculi for service-oriented computing. In *Results of the SENSORIA Project*, volume 6582 of *LNCS*, pages 153–188. Springer, 2011.

[6] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Proc. of IPDPS'01*, page 149. IEEE Computer Society, 2001.

[7] J. C. Godskesen. A calculus for mobile ad hoc networks. In *Proc. of COORDINATION'07*, volume 4467 of *LNCS*, pages 132–150. Springer, 2007.

[8] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, 3(3):223–235, 2010.

[9] J. Honkola, H. Laine, R. Brown, and O. Tyrkko. Smart-m3 information sharing platform. In *IEEE Symposium on Computers and Communications 2010*, volume 3, pages 1041–1046. IEEE Computer Society, 2010.

[10] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[11] I. Lanese and D. Sangiorgi. An operational semantics for a calculus for wireless systems. *Theor. Comput. Sci.*, 411(19):1928–1948, 2010.

[12] R. Lanotte and M. Merro. Semantic analysis of gossip protocols for wireless sensor networks. In *Proc. of CONCUR 2011*, volume 6901 of *LNCS*, pages 156–170. Springer, 2011.

[13] L. Lopes, F. Martins, M. S. Silva, and J. Barros. A process calculus approach to sensor networks programming. In *Proc. of SENSORCOMM'07*, pages 451–456. IEEE Computer Society, 2007.

[14] F. Mattern and C. Floerkemeier. From the Internet of Computers to the Internet of Things. In *From active data management to event-based systems and more*, volume 6462 of *LNCS*, pages 242–259. Springer, 2010.

[15] S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theor. Comput. Sci.*, 367(1-2):203–227, 2006.

[16] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[17] Z. Song, A. A. Cárdenas, and R. Masuoka. Semantic middleware for the Internet of Things. In *Proc. of Internet of Things 2010*, volume 120, pages 1–8. IEEE Computer Society, 2010.

[18] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas. Service oriented middleware for the internet of things: A perspective. In *Proc. of ServiceWave 2011*, pages 220–229. Springer-Verlag, 2011.

[19] The network simulator 3. `http://www.nsnam.org/`.

[20] The OMNET++ network simulation framework. `http://www.omnetpp.org/`.

[21] C.-L. Wu and L.-C. Fu. Design and realization of a framework for human-system interaction in smart homes. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 42(1):15–31, 2012.

[22] C. Yibo, K.-M. Hou, H. Zhou, H. ling Shi, X. Liu, X. Diao, H. Ding, J.-J. Li, and C. D. Vaulx. 6LoWPAN Stacks: A Survey. In *Conference on Wireless Communications Networking and Mobile Computing 2011*, pages 1–4. IEEE Computer Society, 2011.