Intelligent Environments 2016
P. Novais and S. Konomi (Eds.)

doi:10.3233/978-1-61499-690-3-56

A Formal Model for Event-Condition-Action Rules in Intelligent Environments

Claudia VANNUCCHI [a1], Diletta Romana CACCIAGRANO [a], Flavio CORRADINI [a], Rosario CULMONE [a], Leonardo MOSTARDA [a], Franco RAIMONDI [b], Luca TESEI [a]
[a] *Department of Computer Science, University of Camerino, Italy*
[b] *Department of Computer Science, Middlesex University, London, UK*

Abstract. We present a formal model for modelling Event-Condition-Action Rules by partitioning a state space and evolution function taking into account the features that are typical of Intelligent Environments. This model allows for a precise definition of formal requirements and for their efficient verification.

Keywords. Modelling notations, formal analysis and design, rule-based modelling

1. Introduction

The term "Intelligent Environments" is used to encompass a range of applications that range from smart homes to e-health to e-learning, etc. Typically, these applications are developed and deployed by experts and engineers and are then left in the hands of end-users that are allowed to modify the behaviour of some of the components. For instance, a user may change the minimum temperature of a thermostat if specific events occur, such as motion or sound is detected at night time to indicate that a child may have woken up. It is safe to assume that users interact with existing applications by adding or removing *rules*. In general, these rules take the form of Event-Condition-Action (ECA) rules: an *action* is executed if a certain *event* happens and a specific *condition* is met. We expand on ECA rules in Section 2; for the moment, we remark that a substantial amount of effort has been devoted to techniques and tools to guarantee that rules are "correct" with respect to the intended behaviour of the system (where "correct" can be defined in a number of ways).

In this paper we present a model for ECA rules that exploits the features that are typical of Intelligent Environments (IE). In particular, applications for IE are usually built starting from a set of sensors and actuators that interact by means of message passing over a network. Our model characterises the state space of an application by means of *partitions* between sensors and actuators, and introduces evolution functions that distinguish between "environmental" and "artificial" transitions (in a sense to be defined below). We show with a practical example that this characterisation results in a model that is close to the end-user specification and yet compact and suitable to automatic manipulation for analysis and verification.

The rest of the paper is organised as follows. In section 2 we present an overview of the area and we introduce a modelling language for ECA rules; in Section 3 we describe our modelling approach based on the language for ECA rules described in the previous section; in Section 4 we show how our modelling approach can be applied to a practical example. We conclude in Section 5.

2. Related Work

An Intelligent Environment is a physical or logical space that contains a potentially very large number of devices that work together to provide users access to information and services. It is likely to contain many different types of devices linked together, like sensors and actuators, since it must have a clear representation of the physical space from both a sensory (input) and control (output) perspective. ECA based languages have been proposed by a number of sources in order to control sensors and actuators. ECA rules are used to define responses to events and are specified in the form "on the occurrence of certain events, if some conditions are true, perform these actions".

Implementing applications by using ECA rules is an error-prone process, and therefore various formal approaches have been proposed to check and guarantee the correctness of these rules. In [1] the authors present an efficient policy system that enables policy interpretation and enforcement on wireless sensors. Their approach supports sensor level adaptation and fine-grained access control. In [2] the authors present a rule-based paradigm to allow sensor networks to be programmed at run time in order to support adaptation. The approach presented in [3] describes an ECA based middleware for programming IE. While all the aforementioned approaches provide quite powerful tools for programming IE, they do not provide any automatic means of translating programs into formal specifications that can be automatically verified. In [4] the authors translate a set of ECA rules into a Petri Net in order to perform safety analysis. The approaches presented in [5] and [6] use the model checkers SPIN and SMV in order to verify termination. In [7] a tool-supported method for verifying and controlling the correct interactions of rules, is presented. A formalisation of ECA rule-based system is described in order to perform the translation into Heptagon/BZR program.

In spite of this great variety of approaches, to the best of our knowledge formal verification of ECA rules has not been tailored to the context of Intelligent Environments. Indeed, as we show in this paper, it is possible to exploit the structure of this domain to simplify and optimize the modelling and therefore the verification process.

2.1. IRON

We employ IRON (Integrated Rule on Data) as the underlining formalism for modelling Intelligent Environments. IRON is presented in [8]: it is a limited predicate logic-based language that supports the categorisation of devices into sets [9], allows the definition of properties over sets and supports multicast and broadcast abstractions.

IRON programs are composed of two separate classes of specification: static and dynamic. We report the IRON syntax in Figure 1 we recall the IRON syntax, where $[x]$ means an optional occurrence of x, and boldface denotes keywords of the grammar. The *static part* is composed of variables declarations (these variables can be sets, physical

```
1  program ≡ ( device | rule | var_decl )+
2
3  device  ≡ physicalDevice | logicalDevice | set
4
5  physicalDevice  ≡
6    physical (sensor|actuator) type id [= exp] node(id,id)
7    [in id (, id )* ] [where exp]
8  logicalDevice ≡
9    logical (sensor | actuator) type id = exp
10   [in id (, id )* ] [where exp]
11
12 set ≡ set (sensor | actuator) type id
13 rule ≡ rule id on (id)+ when exp then action
14 action  ≡ [id = exp ]+
15
16 exp ≡ exp op exp | (exp) | term
17 term ≡ id | int | set_op set id | true | false | function
18 type ≡ int | boolean
19 set_op ≡ all | any | one | no | lone
20
21 OP ≡ == | != | < | > | <= | >= |  + | - | * | / |
22   and | or | not
23 var_decl ≡ type id = exp [where exp]
```

Figure 1. The IRON extended BNF.

and logical devices) plus global constraints defined over them using restricted first order formulae. A *physical device* defines a piece of hardware that is physically installed in the environment, it has a type (i.e. integer of boolean) and can be either a sensor or an actuator. A physical device has a name and is characterised by the syntax $node(id,id)$ where the first *id* is an identifier that uniquely identifies the physical node while the second *id* uniquely identifies a sensor/actuator that is installed on the node. The keyword *in* can be added in order to specify a list of sets the physical device belongs to. IRON also supports the definition of *logical devices*. A logical device can be set according to the values observed over different sensors and actuators, and thus it produces information that would be impossible to get by considering a single physical device. A logical sensor/actuator does not specify any $node(id,id)$ keyword but must specify an initial value (line $8-10$ of the grammar). The static part also includes the declaration of *constraints* (specified by the keyword *where*), i.e. laws that various variables, devices and sets must always satisfy. Constraints can be used in order to specify rules that bind variables together. The use of a constraint has two applications: (1) it defines valid states of the system regardless of the rules that are defined, and (2) it is used at run-time to verify whether any physical device is providing erroneous data. *Sets* (line 3 of the grammar) are considered to be logical devices and are used to group together either sensors or actuators of the same type (line 12 of the grammar). A programmer can assign values to a set that contains actuators. This assignment can be used in order to instruct all the actuators to perform a specified action. Effectively, a set assignment is an abstraction of a multicast communication primitive that can be used to communicate to actuators an action to be performed. A programmer can read the value of a set of sensors in order to define events and specify conditions. To this end various set operators are introduced.

The *dynamic part* of IRON is composed of ECA rules that are defined by the programmer. The monitoring and control actions are specified by using ECA rules. A rule has a name and is composed of three different parts that are *on*, *when* and *then* (line 13 of the grammar). A list of variables follow the *on* keyword. Whenever one of them changes its value, the boolean expression that follows the keyword *when* is evaluated. When this expression is evaluated to true the rule can be applied and the actions listed after the keyword *action* can be executed. A boolean expression can include relational and logical operators, integers, devices, variables and functions. An action is a list of assignments to variables, physical actuators and logical devices. Special operators are used to support the definition of a boolean condition over a set: *all*, *any*, *no*, *one* and *lone*. *All* is a universal operator that allows the definition of conditions that must be satisfied by all devices belonging to the set. *Any* is an existential operator that can be used in order to specify that at least one of the element of the set must satisfy the condition. *No* (*one*) is used when we need to express that no (respectively, exactly one) element of the set must verify the specified condition. *lone* is used when we need to express that at most one element of the set must verify the specified condition.

3. A Formal Model for ECA Rules

In this section we present a formal model for Event-Condition-Action Rules that partitions the state space and the evolution function taking into account the features that are typical of Intelligent Environments. The model is based on IRON and it allows for a precise definition of formal requirements and for their efficient verification.

3.1. State Space

Applications for IE are usually built starting from a set of sensors and actuators that interact by means of message passing over a network. In this context, we formalize a model representing a system composed of devices of two categories: sensors and actuators. For the sake of simplicity but without loss of generality, our model does not include the definition of sets and the distinction between logical and physical devices. These could be introduced at the cost of additional notation but do not affect the overall partitioning strategy described below.

Let D be the set of labels that identify the devices of the system. We represent D as the union of two disjoint sets, I and O, whose elements are, respectively, the sensors and the actuators of the system. We use the notation $D = \{i_1, \ldots, i_m, o_1, \ldots, o_n\} = I \cup O$ where $I = \{i_1, \ldots, i_m\} (m \in \mathbb{N}_0)$ and $O = \{o_1, \ldots, o_n\} (n \in \mathbb{N}_0)$.

Definition 1. *A* state *of the system is defined as the function $\varphi : D \to Val$ where Val is a finite set of integer or boolean values. We add to this set a special value ω to denote an undefined value in D.*

We can represent the function φ as $\varphi = \{i_1 \mapsto v_{i_1}, \ldots, i_m \mapsto v_{i_m}, o_1 \mapsto v_{o_1}, \ldots, o_n \mapsto v_{o_n}\}$, where $v_{i_j} \in Val$ for $j = 1, \cdots, m$ and $v_{o_h} \in Val$ for $h = 1, \cdots, n$. In other words, a state φ is a set of associations between labels in D and their specific values in Val. We use the notation $\varphi(d)$ for representing the value v_d associated to the generic device d by mean of φ. Given a generic state φ, if it does not contain any association for a certain device

$d \in D$ we use the notation $\varphi(d) = \perp$. We will also use the notation $\varphi = < I_\varphi, O_\varphi >$ instead of $\varphi = I_\varphi \cup O_\varphi$, where $I_\varphi = \{i_1 \mapsto v_{i_1}, \ldots, i_m \mapsto v_{i_m}\}$ and $O_\varphi = \{o_1 \mapsto v_{o_1}, \ldots, o_m \mapsto v_{o_n}\}$. Given the definition of a state, we now introduce the definition of universe.

Definition 2. *The* universe Φ *of a system is the set of all possible states of the system. In other words, it is the set of all possible functions* φ *defined in Def. 1.*

By adding constraints to the system, i.e. conditions that must be satisfied, we can define the admissible state space as follows.

Definition 3. *Let* Φ *be the universe. The* admissible state space Φ_a *is the subset of* Φ *whose elements are all the states* φ *that verify the constraints of the system.*

For instance, if all sensors and actuators in D are boolean, the set Val consists of three values (i.e. two boolean values and the value ω), and the set Φ has cardinality 3^{m+n}. By applying the static constraints we obtain the *admissible state space* $\Phi_a \subseteq \Phi$ having cardinality less or equal than 3^{m+n}. The set of static invariants must be satisfied independently from the set of ECA rules of the system. We will denote the cardinality of Φ_a with z and a generic state in Φ_a with φ. The notation φ_{out} will be used to represent a generic state that is not a member of the set Φ_a.

3.2. ECA Rules

Given the set D and the state space Φ, we consider the finite set R of labels for ECA rules $R = \{r_1, r_2, ..., r_k\}, \quad k \in \mathbb{N}_0$. Following [10], we use the notation $Event[Condition]/Action$, to represent a generic ECA rule labelled with r. Therefore, a generic rule r in R is represented as $e_r[c_r]/a_r$, where e_r, c_r, a_r are labels for the event, the condition and the action of r respectively. Let now define each component of the ECA rule r. The generic event e_r is represented as a subset of labels in D, i.e. $e_r = \{d_{w_1}, \cdots, d_{w_f}\} \subset D$. The event is the trigger for the ECA rule, i.e. when a change concerning the value of at least one of the labels in e_r occurs, the condition is evaluated. The condition c_r is a restricted first-order logic predicate (as defined in the IRON grammar) having variables in D, i.e. $c_r = P(d_{\beta_1} \cdots, d_{\beta_l}), \quad d_{\beta_1} \cdots, d_{\beta_l} \in D$. If the condition is true, the action is applied to the state of the system. A generic action a_r is defined as a set of assignments for a subset of actuators in O, i.e. $a_r = \{o_{\alpha_1} \leftarrow v_{\alpha_1}, \cdots o_{\alpha_p} \leftarrow v_{\alpha_p}\}$, where $o_{\alpha_1}, \ldots, o_{\alpha_p}$ are in O. For the state φ, for all $o_j \in O_\varphi$ we use the following notation:

$$a_r(o_j) = \begin{cases} v_{\alpha_k} \text{ if } o_j = o_{\alpha_k} \\ v_{o_j} \text{ if } o_j \neq o_{\alpha_k} \end{cases}$$

Notice that we have exploited here the features that are typical of IE, taking into account the fact that a generic action defined by the user can only change actuator configurations.

3.3. System Evolution

In this section we introduce a specific formalism for representing the evolution of our system. The formalisation is based on the following observation: according to the fea-

tures typical of IE that are described in our model, the evolution of the system can be partitioned into two sets, i.e.

- the set of *artificial transitions* resulting from the application of ECA rules (these changes concern only actuator values);
- the set of *natural transitions* that result from the spontaneous changes of the environment (these variations concern only sensor values).

According to this partition, we can introduce the following concepts:

- a state is called *stable* if it is the source state of a natural transition;
- a state is called *unstable* if it is the source state of an artificial transition.

We can observe that a natural transition can be formalised using the function $t_N : I \times O \rightarrow I$ that changes only natural values, while a generic artificial transition can be represented with the function $t_A : I \times O \rightarrow O$ that affects only actuator values. In our model, artificial transitions correspond to the application of ECA rules.

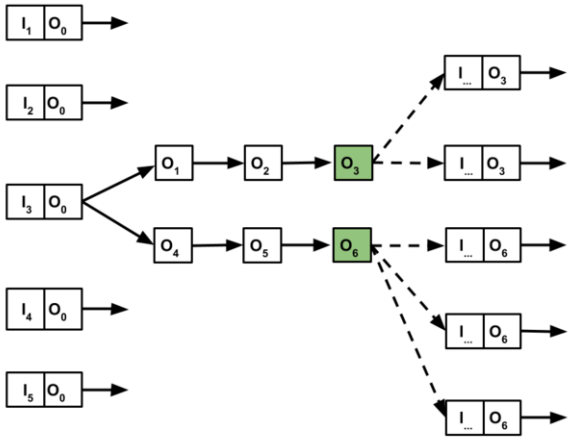

Figure 2. System evolution.

Using the previously introduced formalism, the function t_N leads a generic state $\varphi =< I_\varphi, O_\varphi >$ into the state $\varphi' =< I'_\varphi, O_\varphi >$ changing only the sensor value configuration, while t_A leads a state $\varphi =< I_\varphi, O_\varphi >$ into the state $\varphi'' =< I_\varphi, O''_\varphi >$ changing only the actuator value configuration. Figure 2 shows the evolution of an example system from the initial state $< I_3, O_0 >$: the successor states are obtained by applying two artificial transitions (depending on the condition that is satisfied) that lead the system from $< I_3, O_0 >$ either to O_3 (notice that I_3 is omitted as it does not change) or to O_6 . The states $< I_3, O_3 >$ and $< I_3, O_6 >$ are *stable* and only natural transitions can be applied to these states.

Using the evolution functions t_N and t_A, it is possible to perform a partition over the set Φ_a. We use the generic example represented in Figure 3 for a clarification of how t_N and t_A interact. In this figure, white circles are unstable states in Φ_a, while green states are stable states. Natural transitions are represented by dotted arrows, while artificial transitions are represented by solid arrows. States are grouped together into three

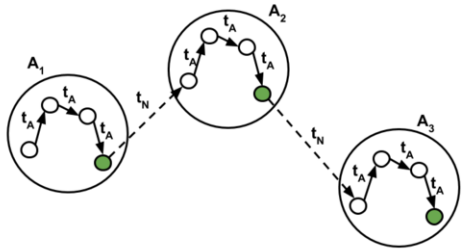

Figure 3. System behaviour.

sets, A_1, A_2, A_3 in which the input sensors do not change value. Natural transitions link together these sets. Inside each set, the states are linked to each other via artificial transitions, since changes are due only to actuator values.

In summary, in our model artificial transitions correspond to the application of ECA rules, while natural transitions correspond to changes in the environment. We do not model here the spontaneous evolution of the environment but we only give a representation of the natural evolution in terms of "the minimum natural transition that links a stable state to an unstable state".

The representation of the evolution of the system in our model is based on the two hypotheses: the first one concerns the representation of initial states of the system, the second one is related to the execution time of transitions.

1. The initial configuration of actuators is given by an external entity. Let O_0, be the initial configuration of the actuators, and we additionally assume that each configuration verifies the static constraints. As a consequence of this hypothesis, the number of initial states in our representation corresponds to the number of all possible configurations I_j of sensors such that $< I_j, O_0 >$ is an admissible unstable state for the system, i.e. there is at least one ECA rule that can be applied to this state.
2. Natural transitions take a longer time than artificial evolutions, that is to say the maximum execution time of the chains of artificial transitions is always strictly less then the minimum execution time of natural transitions.

Intuitively, looking at Figure 2, the configuration O_0 is known and the initial states of the system are $< I_j, O_0 >$ with $j = 1, ..., 5$. Taking into account the example in Figure 3 we can represent the second hypothesis as follows:

$$\min_{t_N | A_j \xrightarrow{t_N} A_{j+1}, j=1,2} elapsed\_time(t_N) > \max_{i=1,2,3} \sum_{t_A \in A_i} (exec\_time(t_A)) .$$

4. A Concrete Example

Monitoring and automatic control of a building environment is a case study considered quite often in literature, see for instance [11,12]. Home automation can include a number of functionalities, such as centralised light control, emergency control systems

(home burglar security alarm, fire alarm); (iii) heating, ventilation, and air conditioning (HVAC) systems, etc. Consider the example represented in Figure 4. The house is composed of two rooms, the living room (denoted with L) and the bedroom (B). We assume that only one person has access to the house at any given time. We assume that a 360 Degree Passive Infrared Motion Sensor (PIR) is placed on the ceiling of both rooms, and each room is completely covered by this sensor. The occupancy sensor placed in every room can be used for controlling lights automatically. We also hypothesise that a light switch is installed in every room to turn on and off the light manually (for example, when someone goes to sleep). The set of devices placed in the house is given by $D = \{L_m, L_l, L_s, L_a, B_m, B_l, B_s, B_a\}$ where L stands for living room, B stands for bedroom, m represents a PIR detector, l is a natural light sensor, s is a light switch, and a is a lamp. For example the light sensor in bathroom is labelled with B_l. The entrance is in the living room and the bedroom is accessible from the living room.

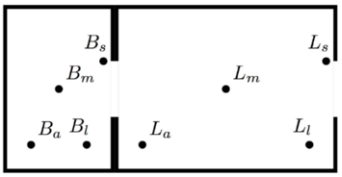

Figure 4. Floormap.

As described above, a generic state of the system is a function $\varphi : D \to Val$ where $Val = \{0, 1, \omega\}$. As above, $\varphi =< I_\varphi, O_\varphi >$ where, in our case, $I_\varphi = \{B_m \mapsto v, B_l \mapsto v, B_s \mapsto v, L_m \mapsto v, L_l \mapsto v, L_s \mapsto v\}$, and $O_\varphi = \{B_a \mapsto v, L_a \mapsto v\}$. We define the following static constraints for the system:

(i) $[\neg(L_m \wedge B_m)]$
(ii) $[\neg(L_a \wedge B_a)]$

The first constraint states that the person cannot stay in both rooms simultaneously. The second one states that light actuators cannot be both on at the same time. As a consequence, according to the constraint (i), a state having both light actuators on is not admissible. By applying these constraints, we obtain the admissible state space Φ_a. This set cannot contain states with $\varphi(B_m) = \varphi(L_m) = 1$ nor states having $\varphi(B_a) = \varphi(L_a) = 1$. The dynamic of the system is then defined using the following ECA rules:

r_1: $L_m[L_m \wedge \neg L_l]/L_a \leftarrow 1$
r_2: $B_m[\neg L_m \wedge B_m \wedge \neg B_l]/L_a \leftarrow 0, B_a \leftarrow 1$
r_3: $B_m[\neg B_m \wedge L_m \wedge \neg L_l]/B_a \leftarrow 0, L_a \leftarrow 1$
r_4: $L_m[\neg L_m \wedge L_a]/L_a \leftarrow 0$

For instance, r_1 encodes the fact that when a person enters in the living room and the light is off, the light is turned on (and similarly for r_2 to r_4 encoding the dynamic of the light actuators in all rooms). Let be the initial actuator configuration of the system given by $O_0 = \{B_a \mapsto 0, L_a \mapsto 0\}$, i.e., all lights off, and supposed that in this initial state a person enters the room. This is captured by φ_1 in Table 1.

A sample evolution of the system is represented in Figure 5. The key point here is that the full state space of the system would have 3^8 states (due to the ω value for

Table 1. States.

	I						O	
	L_m	L_l	L_s	B_m	B_l	B_s	L_a	B_a
φ_1	1	0	0	0	0	0	0	0
φ_2	1	0	0	0	0	0	1	0
φ_3	0	0	0	1	0	0	1	0
φ_4	0	0	0	1	0	0	0	1
φ_5	1	0	0	0	0	0	0	1
φ_6	0	0	0	0	0	0	1	0
φ_7	0	0	0	0	0	0	0	0

not admissible or unknown values). However, thanks to our approach, a partition of sets having the same sensor configuration can be obtained and we only need to deal with changes in actuators satisfying the constraints for the system. This results in a state space including only 3 valid combination for output variables that need to be considered over 6 sensors, not all of which admissible. Indeed, the admissible sensor values are $3 \cdot 3 \cdot 4 = 36$, instead of 64. This is because the two lights cannot be on at the same time (hence only 3 possible values for light actuators), and the same for the two PIR sensors as a person can only be in one room. The two light switches, instead, are not affected by our constraint rules.

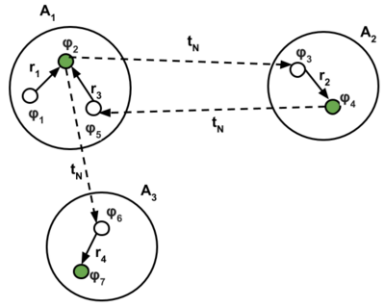

Figure 5. System evolution.

5. Conclusions

In this paper we have presented a model for Intelligent Environment that formalises Event-Condition-Action rules and exploits the partition of devices into sensors and actuators. Our example provided in the previous section shows that the state space can be reduced substantially when constraints are added to the separation between input and output devices. Moreover, our model represents ECA rules by means of restricted first-order expressions without nested quantifiers and using only linear arithmetic operators.

While our model could be used as is to reason about the correctness of requirements for specific systems, our aim for the future is to explore the use of verification algorithms operating directly on our representation. In particular, we are currently investigating the

use of SMT solvers such as CVC4 [13] in conjunction with a formal characterisation of the correctness of rules, such as non-confluence, non-redundancy, etc. We also plan to investigate the extension of the formalism to deal with noisy sensors and probabilistic reasoning.

References

[1] Y. Zhu, S. L. Keoh, M. Sloman, E. Lupu, N.Dulay, and N. Pryce. An Efficient Policy System for Body Sensor Networks. In *14th International Conference on Parallel and Distributed Systems (ICPADS 2008)*, pages 383–390, 2008.

[2] X. Fei and E. H. Magill. REED: Flexible rule based programming of wireless sensor networks at runtime. *Computer Networks*, **56**(14):3287–3299, 2012.

[3] G. Russello, L. Mostarda, and N. Dulay. A policy-based publish/subscribe middleware for sense-and-react applications. *Journal of Systems and Software*, **84**(4):638–654, 2011.

[4] X. Jin, Y. Lembachar, and G. Ciardo. Symbolic Verification of ECA Rules. In Daniel Moldt, editor, *PNSE+ModPE*, volume 989 of *CEUR Workshop Proceedings*, pages 41–59. CEUR-WS.org, 2013.

[5] E.-H. Choi, T. Tsuchiya, and T. Kikuno. Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier*, **2**:826–839, 2006.

[6] I. Ray and I. Ray. Detecting Termination of Active Database Rules Using Symbolic Model Checking. In *Advances in Databases and Information Systems, 5th East European Conference, ADBIS 2001, Vilnius, Lithuania, September 25-28, 2001, Proceedings*, pages 266–279, 2001.

[7] J. Cano, G. Delaval, and E. Rutten. *Coordination of ECA Rules by Verification and Control*, pages 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[8] F. Corradini, R. Culmone, L. Mostarda, L. Tesei, and F. Raimondi. A Constrained ECA Language Supporting Formal Verification of WSNs. In L. Barolli, M. Takizawa, F. Xhafa, T. Enokido, and J. H. Park, editors, *29th IEEE International Conference on Advanced Information Networking and Applications Workshops, AINA 2015 Workshops, Gwangju, South Korea, March 24-27, 2015*, pages 187–192. IEEE Computer Society, 2015.

[9] L. Mostarda, S. Marinovic, and N. Dulay. Distributed Orchestration of Pervasive Services. In *24th IEEE IAINA 2010, Perth, Australia, 20-13 April 2010*, pages 166–173, 2010.

[10] M. Dumas and A. M. ter Hofstede. UML Activity Diagrams As a Workflow Specification Language. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 76–90, London, UK, 2001. Springer-Verlag.

[11] D.-M. Han and J.-H. Lim. Smart home energy management system using IEEE 802.15.4 and Zigbee. *Consumer Electronics, IEEE Transactions on*, **56**(3):1403–1410, Aug 2010.

[12] K. Gill, S.-H. Yang, F. Yao, and X. Lu. A Zigbee-based home automation system. *Consumer Electronics, IEEE Transactions on*, **55**(2):422–430, May 2009.

[13] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.