Architectural Invariants and Correctness of IoT-Based Systems

Christian Attiogbé(✉) and Jérôme Rocheteau

Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, 44000 Nantes, France
{christian.attiogbe,jerome.rocheteau}@ls2n.fr

Abstract. Systems based on the Internet of Things impact more and more industrial areas such as smart manufacturing, smart health monitoring and home automation. Ensuring their correct construction, their well functioning and their reliability is an important issue for some of these systems which can be critical in case of dysfunction. The main requirements on physical architectures and control software are common in most of IoT-based systems. Therefore, we propose on the basis of their common architectural properties and behaviour, a generic formal model of IoT-based systems together with the rigorous analysis of their consistency properties; specific properties may be gradually added and checked. The proposed generic formal model is implemented as a parametrised model and experimented using the Event-B framework. This parametrised model is extensible; it can be profitably adapted to more general hybrid or cyber-physical systems. Moreover, our generic model is independent of the target formal modelling tools, it can be implemented in various other formal analysis environments.

Keywords: IoT Applications · Generic formal model · Invariant properties · Event-B

1 Introduction

Internet of Things *systems* impact industrial areas such as smart manufacturing, smart vehicles, smart logistics and transportation, smart farming, etc. These applications share a well-established architectural structuring, reference models and some functional and non-functional properties [2,6,8]. However, well-established engineering methods and techniques are still needed [19] to ensure that the applications are reliable, secure, scalable, well-integrated, and extensible. In this context, we are motivated by proposing methods and tools for mastering the modelling, the analysis and the development of such IoT-based applications. The challenge is that these applications can become rapidly complex because of their heterogeneous and evolving environment.

In order to ensure the consistency and the well-functioning of an IoT-based application, the latter should integrate as a parameter, the complete description of the physical context that it controls. Therefore, a global model can be built and analysed with respect to intrinsic consistency and to the required specific properties. We propose such a global formal modelling and the related analysis.

P. Fournier-Viger et al. (Eds.): MEDI 2022, CCIS 1751, pp. 75–88, 2022.
https://doi.org/10.1007/978-3-031-23119-3_6

The contributions of this paper are manifold: we propose *i)* a generic formal description of the physical architecture of an IoT-based system; *ii)* a formal description of the control application parametrised by its physical environment; *iii)* the modelling and analysis of the invariant architectural properties of such IoT-based systems and the description of some specific properties. They are described to be customizable for other application cases. We design using Event-B, a formal parametrised model, to support the full modelling and analysis approach. Moreover, we build a tool that generates systematically for a given IoT-based application, the specific Event-B parts that are used to instantiate the parametrised Event-B model.

The organisation of the article is as follows. In Sect. 2, we introduce the background concepts. Section 3 is devoted to the generic formal model of IoT-based applications. In Sect. 4, we deal with the invariant consistency properties, formalised with operational semantic rules. In Sect. 5, we show how we have implemented our proposed generic formal model and analysis technique using Event-B. We compare our work to the related ones in Sect. 6. Finally, Sect. 7 presents some perspectives and future work.

2 Basic Concepts and Architecture of IoT Systems

Based on state-of-the-art references [2,8,14,20] of IoT technologies, challenges, comparisons, and reference models, we consider the following main elements of IoT.

A *thing*[1] is a physical object equipped with: *i)* sensors that collect and gather data from the environment; *ii)* actuators that allow the control of the *thing* or allow the *thing* to act on its environment.

An IoT-based *application* is a software made of services, built on top of the physical infrastructure made of one or several things. An example of the architecture of a control application is depicted in Fig. 1 where we can distinguish: *i)* a physical part made of the controlled devices equipped with sensors and actuators; *ii)* a software part made of the (sub-)controllers which interact with the physical part through an event dispatcher. This abstraction covers the four-layers architecture widely admitted [2] now for service-oriented architecture (SOA) IoT systems.

A *control application* sends orders (including signals) to actuators, according to information collected by sensors. A control application often uses *rules* stated by a human expert or systematically computed from a specific database, to issue control orders. Therefore, the main components to deal with for an IoT-based system are: a physical part made of sensors, actuators, *things*, and network infrastructure; a software part made of a *control application* and potentially specific control or monitoring services. Additionally, the components interact through low level or application level communication protocols such as WiFi, bluetooth, ZigBee and MQTT [4].

3 Formal Model of IoT-Based Systems

We propose in the following an abstract formal model of IoT-based systems. We use set theory and relation notations to structure the model components. Sets are written with

[1] we keep the vocabulary of IoT domain.

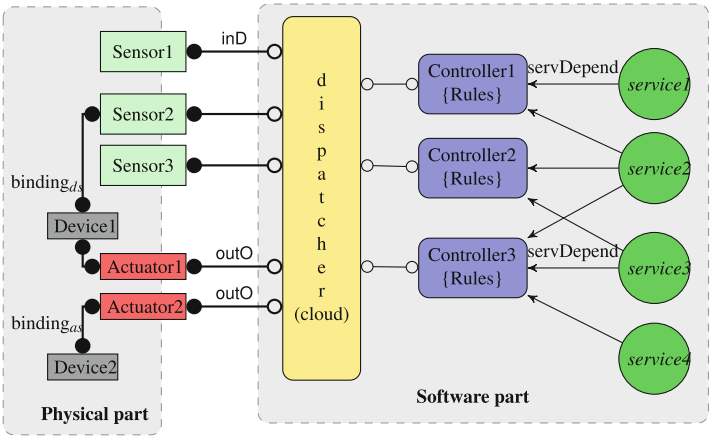

Fig. 1. An abstract architectural view of an IoT-based system

capital letters, the standard set operators ($\in, \subseteq, \cdots$) are used. A relation r defined over the sets S and P is written: $r : S \leftrightarrow P$ or more conventionally $r \subseteq S \times P$; a function f over S and Q is written: $f : S \rightarrow Q$; the relational operators *ran* and *dom* denote respectively the *range* and the *domain* of a relation or a function.

An IoT-based application is composed at least of: a set of connected IoT devices ($\mathcal{D}$), sensors ($\mathcal{S}$) and actuators ($\mathcal{A}$), that make the physical part; a set of controllers ($\mathcal{C}$), sometimes together with a dedicated server (or dispatcher) which collects the data from sensors and distributes them to the controllers. Sensors and actuators are bound to the devices. A controller is linked to the sensors from which it reads inputs, and to the actuators it manages. Sensors and actuators may be connected to several interconnected controllers.

3.1 Elementary Components of the Model

Sensors. A sensor s provides a value in a given range; a value will correspond to a state of the device that it senses. Each category (c_s) of sensors may have various value ranges (R_{cs}). A sensor interacts with its environment through communication protocols. Let *CommProto* be such a set of protocols; a sensor s of the set of sensors $\mathcal{S}$ ($s \in \mathcal{S}$) is defined by a 4-tuple $(categ_s, range_{sc}, value_s, comm_{p_s})$ with a category $categ_s \in C_s$, a range $range_{sc} \subseteq R_{cs}$, a current value in its range $value_s \in r_{sc}$ and a set of communication protocols $comm_{p_s}$. We use the following relations to get each element of the 4-tuple:

$$categ_s : \mathcal{S} \rightarrow C_S \qquad range_s : \mathcal{S} \leftrightarrow R_{cs}$$
$$value_s : \mathcal{S} \rightarrow R_{cs} \qquad comm_{p_s} : \mathcal{S} \leftrightarrow CommProto$$

Actuators. An actuator a of the set of actuators $\mathcal{A}$ ($a \in \mathcal{A}$) receives an input in a specific range of order values ($Order_A$), and delivers accordingly an output signal (from a set $Signal_A$) towards its environment. We use the triple $(inputOrd_a, ouputSign_a, comm_{p_a})$ to describe an actuator and the following relations to determine its elements:

$$inputOrd_a : \mathcal{A} \leftrightarrow Order_A \quad ouputSign_a : \mathcal{A} \leftrightarrow Signal_A \quad comm_{p_a} : \mathcal{A} \leftrightarrow CommProto$$

They give the set of inputs, outputs and protocols of an actuator. We define later the links between an actuator and a device or a controller.

Devices. A device ($d \in \mathcal{D}$) is modelled by its state s_d in such a way that a range of measured values V_v of some sensors s (linked d), corresponds to s_d (that is, $V_v \mapsto s_d$). In the same way, the output signals of an actuator can, upon the reception of an order, set the device d in a state s_d. Therefore, a device d is characterised by its set of states (*STATED*), corresponding to its behaviour which is, moving from state to state according to the received stimulation signals. Accordingly, d is modelled with a transition system $\langle STATED,\ SignalA,\ S_0,\ \delta \rangle$ which abstracts its behaviour; with *SignalA* the set of received signals, $\delta : STATED \times SignalA \rightarrow STATED$ a transition function, and S_0 an initial state of the device. We use a function $curState_d : \mathcal{D} \rightarrow STATED$ to denote the current state.

Services. The tasks performed in controller applications consist in applying a set of control rules (R) to analyse data (D_S) collected from the sensors ($\mathcal{S}$) and to compute, the orders (*Order*) to be sent to the actuators. The services implement these control rules. A controller c is then equipped with a function $ComputeOrder_R : D_S \rightarrow Order$. When the controller collects values from sensors bound to it, it computes the appropriate order using the rules R, and outputs this order to the actuator bound to it.

3.2 Abstractions for IoT-Based Systems

We describe an abstract model of an IoT-based system according to the two main components presented in Sect. 2: we build the abstract model M_{phys} of the physical part and the abstract model M_{soft} of the control software part. We consider the previous sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{D}$. The physical architecture is modelled with a n-tuple $M_{phys} = (S,\ A,\ D,\ binding_{(d,s)},\ binding_{(a,d)})$ where $S \subseteq \mathcal{S}$ is a subset of sensors; $A \subseteq \mathcal{A}$ is a subset of actuators; $D \subseteq \mathcal{D}$ is a set of sensed or controlled devices; $binding_{(d,s)} \subseteq D \times S$ is a relation that describes the binding between the sensed devices and their sensors, and $binding_{(a,d)} \subseteq A \times D$ is a relation that describes the binding between the actuators and the controlled devices. The control dependency between devices is defined later.

A control software (or application) part of an IoT system is modelled with the tuple $M_{soft} = (\mathcal{C}_R, Serv, servDepend_{(c,s)})$ where $\mathcal{C}_R \subseteq \mathcal{C}$ is a set of controllers which use a set of the control rules R for their control tasks; *Serv* is a set of services used or provided by the controllers; $servDepend_{(c,s)} \subseteq \mathcal{C}_R \times Serv$ is the dependencies between the controllers and the used services.

To model the link between the control application and the sensors and actuators, the physical (M_{phys}) and software (M_{soft}) models are linked with the following relations:

- $inD \subseteq S \times C_R$ which models the links between the sensors and the controller; it supports data input from the sensors;
- $outO \subseteq C_R \times A$ which models the links between the controller and the actuators; it supports order output to actuators.

At this stage the control application (M_{soft}) can communicate with the physical part via its abstract model (M_{phys}). Typically the application receives data from the sensors

(via *inD*) and issues orders sent to the actuators (via *outO*). The orders are computed from the application services using defined rules.

But, in order to state the *architectural invariants* and to analyse the IoT system properly, we need to address one of the identified defects leading to inconsistencies, which is the lack of explicit declaration of dependencies between sensors and related controlled devices. When the control of a given device depends on some sensors, this dependency should be made explicit. The devices should have been equipped by an actuator which share the same controller with the involved sensors. Therefore, it is necessary to make explicit in the model, the control dependency relation between involved sensors and controlled devices; this is done with a relation $CtrlDepend_{(s,d)} \subseteq S \times D$.

This relation describes which sensors are used to control which devices, so that we can reason later on the consistency of the functioning of the global system. Both the relations $CtrlDepend_{(s,d)}$ and $binding_{(d,s)}$ are necessary since the impacted devices described by $CtrlDepend_{(s,d)}$ can be different from the sensed ones described by $binding_{(d,s)}$. Consequently, given a control part $M_{soft} \mathrel{\widehat{=}} (C_R,\ Serv,\ servDepend_{(c,s)})$ with a physical architecture $M_{phys} \mathrel{\widehat{=}} (S,\ A,\ D,\ binding_{(d,s)},\ binding_{(a,d)})$, and their interconnection with the relations *inD*, *outO* and $CtrlDepend_{(s,d)}$, the model of the complete control system *Sys* integrating them is described by the 5-tuple:

$$Sys = (M_{phys},\ M_{soft},\ inD,\ outO,\ CtrlDepend_{(s,d)})$$

For the systematic construction and analysis of the model, we structure it with parameters made of all, or some parts, of the global system; hence the genericity; different parameters lead to specific systems: $Sys[M_{phys},\ M_{soft},\ inD,\ outO,\ CtrlDepend_{(s,d)}]$. This enables us to build separately the different parts, and also to modify them easily as well as their interconnections; we can freeze a physical architecture and check different versions of the control part or as done in the following, freeze the software and check some configurations of the physical parts. Then, the global model is a parametrised structure with the parameters M_{phys}, *inD*, *outO* and $CtrlDepend_{(s,d)}$, and a frozen software part:

$$Sys_{(M_{soft})}[M_{phys},\ inD,\ outO,\ CtrlDepend_{(s,d)}]$$

3.3 Behavioural Description of a Control Application

A control application continuously reacts to the data collected by sensors and changes the state of its environment by sending orders to the involved actuators. We use operational semantic rules to describe this general behaviour. First, we assume that the application is *consistent* so that it can react properly to the sensed data. In Sect. 4.1 we show how the consistency properties are defined and then how they can be checked in Sect. 4.2. Given $Sys_{(M_{soft})}[M_{phys},\ inD,\ outO,\ CtrlDepend_{(s,d)}]$, a consistent application with $M_{phys} = (S,\ A,\ D,\ binding_{(d,s)},\ binding_{(a,d)})$ and $M_{soft} = (C_R,\ Serv,\ servDepend_{(c,s)})$, when a controller c_i (with a function $computeOrder_{c_i}$) of C_R, receives a value val from a sensor s_i of S bound to a device d_s of D, considering that the control of a device d_c depends on the sensor s_i, and that there is an actuator a_s bound to d_c, then an order ord_i, computed by the controller c_i linked to a_s, is sent to the actuator a_s.

Control Abstraction and Genericity. A sensor covers a set of ranges that will be matched to a set of states of the device. We consider this matching, as a (explicitely provided) function σ from a set of values to a set of states. Consequently, given a sensor s_i with its range of values r_{sc}, a device d_c characterised by $\langle STATED, Signal, S_0, \delta \rangle$, in order to control d_c, we need a function $\sigma_{sd} : r_{sc} \rightarrow STATED$ (considered as a *parameter* of a controler) that links s_i and d_c. This is provided through the services on which the controller depends. Therefore, a controller c_i is parametrised by s_i, d_c and σ_{sd} (denoted by $c_i[s_i, d_c, \sigma_{sd}]$) in such a way that given a value v from s_i, $\sigma_{sd}(v)$ provides the corresponding state s of the device d_c. Before changing the state of d_c according to values sensed by s_i and the predefined behaviour δ of d_c, the controller should compute, provided that σ_{sd} is already defined through its services, the appropriate order to be sent to the device. Hence, the following generic semantic rule (compOrd) to compute orders. Note that, when $curState(d_c) = s$, nothing should be changed.

$$\frac{\begin{array}{c} s_i \mathrel{\widehat{=}} (c_s, r_{sc}, v_s, comm_{p_s}) \\ d_c \mathrel{\widehat{=}} \langle STATED, Signal, S_0, \delta \rangle \\ \sigma_{sd} : r_{sc} \rightarrow STATED \qquad v_s \in r_{sc} \\ \sigma_{sd}(v_s) = s \qquad curState(d_c) \neq s \\ \mathsf{thisOrder} \in Signal \quad (curState(d_c), \mathsf{thisOrder}, s) \in \delta \end{array}}{ComputeOrder_{c_i[s_i,d_c,\sigma_{sd}]}(v_s) = \mathsf{thisOrder}} (\mathsf{compOrd})$$

We formally define the general behaviour of the IoT application by the following operational semantic rule where the operators $\Downarrow$ and $\Uparrow$ denote respectively the reception of a value from a given sensor by a controller and the sending of an order by a controller to an actuator. Thus $c_i \Downarrow (s_i, \mathsf{val})$ expresses that the controller c_i receives the value val sent by the sensor s_i; similarly $c_i \Uparrow (a_s, \mathsf{ord}_i)$ expresses that the controller c_i sends the order ord_i to the actuator a_s. The rule (RoS - react on sense) captures the traditional sense-decision-control paradigm of control systems.

$$\frac{\begin{array}{c} M_{soft} \mathrel{\widehat{=}} (\mathcal{C}_R,\ Serv,\ servDepend_{(c,s)}) \\ M_{phys} \mathrel{\widehat{=}} (S, A, D, binding_{(d,s)}, binding_{(a,d)}) \\ Sys_{(M_{soft})}[M_{phys},\ inD,\ outO,\ CtrlDepend_{(s,d)}] \\ s_i \in S \quad a_s \in A \quad d_s \in D \quad d_c \in D \quad c_i[s_i, d_c, \sigma_{sd}] \in C_R \\ binding_{(d,s)}(d_s) = \{s_i\} \qquad (a_s, d_c) \in binding_{(a,d)} \\ (s_i, c_i) \in inD \quad (c_i, a_s) \in outO \quad (s_i, d_c) \in CtrlDepend_{(s,d)} \\ c_i \Downarrow (s_i, \mathsf{val}) \quad \mathsf{val} \in range_s(s_i) \\ \mathsf{ord}_i = ComputeOrder_{c_i[s_i,d_c,\sigma_{sd}]}(\mathsf{val}) \end{array}}{c_i \Uparrow (a_i, \mathsf{ord}_i)} (\mathsf{RoS})$$

A consequence of the RoS rule is the *Integrity of orders*: any order sent to an actuator results from one of the services (they provide σ_{sd}) of the control application. Since the computation of orders is due to the controller whose services implement (via σ_{sd}) the control application rules (previously denoted R), the orders sent to the actuators are the right ones. However, the integrity checking should be propagated until the application implementation level.

The RoS rule expresses one step of the cyclic behaviour of the control application; the repetition of the step is captured by the continuous enabling of the rule. We have

also generalized the RoS rule by considering an extension of $computeOrder_{c_i}$, with a function $ComputeGlobalOrder_{Sys}$ that computes an order, according to the global state of the system. A design flaw may happen at the system level; we formalise this situation with a semantic rule (dFlaw rule) which expresses that when there is no transition from the current state of the controlled device d_c to a state corresponding to the value sent by a related sensor (i.e. $(curState(d_c), *, s) \notin \delta$), the controller is not able to send an order to the actuator, since the order is undefined. In the case of inconsistent values we have to implement (based on the dFlaw rule) a rule that raises inconsistency of sensors.

Further Generalisation. The rule compOrd considers the computation of an order related to the state of one device; this rule may be extended to the states of several devices, and also the computation of several orders to be emitted to change the state of the system; it will involve an evolution with a sequence of orders. This requires the extension of the emission operator to the emission of the sequence of orders.

From now on, we have captured the correct behaviour and the possible design flaw of an IoT-based control application.

4 Consistency Properties and Analysis of the Formal Model

The generic formal model built in the Sect. 3 is enhanced with consistency properties.

4.1 Invariant Consistency Properties

We focus on the architectural consistency and then, on the consistency of the functioning of IoT-based applications. Let us consider in the following, an application defined with

$$M_{phys} \widehat{=} (S, A, D, binding_{(d,s)}, binding_{(a,d)}) \qquad M_{soft} \widehat{=} (C_R,\ Serv,\ servDepend_{(c,s)})$$
$$Sys_{(M_{soft})}[M_{phys},\ inD,\ outO,\ CtrlDepend_{(s,d)}]$$

We describe a list of properties required for an IoT architecture to be consistent so that a model satisfying these properties will be consistent.

*Well-Structuring of Physical Components (*wsHW*).* An IoT architecture involves devices, sensors and actuators; it is described with two binding relations. To be controlled, the architecture requires the following property which expresses partial connectivity (with a disjunction) in order to be less restrictive, instead of full connectivity: $binding_{(d,s)} \neq \varnothing \ \vee \ binding_{(a,d)} \neq \varnothing$

*Well-Structuring of Controllers (*wsCtrl*).* An IoT control application requires a connection with at least one sensor and one actuator: $\forall c_i \in C_R.(inD(c_i) \neq \varnothing \ \wedge \ outO(c_i) \neq \varnothing)$

*Weak Consistency of Components Involved in Interactions (*wkCst*).* For consistency purpose, sensors or actuators involved in the interactions should be those described in the physical support: $\mathrm{dom}(inD) \subseteq S \ \wedge \ \mathrm{ran}(outO) \subseteq A$

However, this consistency is weak, because it does not constrain the linking of the involved sensors and actuators. For more accuracy, we define the following stronger property.

*Consistency of Control Dependencies (*CstBind*).* The consistency of the device controls requires that: the actuators to whom a controller sends its orders (via $outO$), are those actuators bound (via $binding_{(a,d)}$) to the devices which are controlled (via $CtrlDepend_{(s,d)}$) by the sensors bound (via inD) to the controller. Thus and interaction consistency property is required; it is expressed by the equality of the composition of the involved relations: $inD; outO = ctrDepend; binding_{ad}^{-1}$

The *Consistency of control dependencies* property ensures that: if a sensor s_i impacts the control of a given device d_c (via $CtrlDepend_{(s,d)}$), and the sensor s_i is connected to a controller c_i (via inD) then the actuator a_k bound (via $binding_{(a,d)}$) to the device d_c is also linked (via $outO$) to the controller c_i.

$$\frac{\begin{array}{c} s_i \in S \qquad c_i \in C_R \qquad\qquad a_k \in A \qquad d_c \in D \\ (s_i, c_i) \in inD \qquad (a_k, d_c) \in binding_{(a,d)} \\ (s_i, d_c) \in CtrlDepend_{(s,d)} \end{array}}{(c_i, a_k) \in outO}\,(\mathsf{CstDep})$$

*Well-Structured Connection of Actuators and Sensors (*wsS2A*).* The controllers which are connected to sensors should also be connected to some actuators, otherwise the collected inputs are not used for the control: $\mathrm{ran}(inD) \subseteq \mathrm{dom}(outO)$

This property can be relaxed if one considers applications without actuators, or with a pool of interacting controllers.

*Consistency for Communication Protocols (*wsProt*).* The consistency of protocols requires that the pairs of sensor-controller and controller-actuator use compatible communication protocols: each sensor interacts with the bound controller using an appropriate communication protocol; each controller interacts with the bound actuators using an appropriate communication protocol. Consider the set of communication protocols ($CommProto$) used by the components of the architecture and $c_i \in C_R$, $s_i \in S$, $a_s \in A$, $(s_i, c_i) \in inD$, $(c_i, a_s) \in outO$ such that $comm_{p_s}(s_i) \subseteq CommProto \ \wedge \ comm_{p_c}(c_i) \subseteq CommProto \ \wedge \ comm_{p_a}(a_s) \subseteq CommProto$. The protocol consistency requires:

$$(comm_{p_s}(s_i) \cap comm_{p_c}(c_i) \neq \varnothing) \ \wedge \ (comm_{p_c}(c_i) \cap comm_{p_a}(a_i) \neq \varnothing)$$

4.2 Consistency Analysis of IoT-Based Control Applications

Given the previous consistency properties, we state the following two definitions for the consistency analysis of IoT-based control applications. If a given model satisfies the properties then it is consistent.

Definition 1. *(Architectural correctness) A given physical architecture M_{phys} is said consistent if it preserves the property* wsHW.

The physical architecture can be given without any controller, while the other properties involve the relations with the controller and a specific connection of the architectures.

Definition 2. *(Behavioural correctness) An application* $Sys_{(M_{soft})}$*, parametrised with* M_{phys}*, inD, outO and* $CtrlDepend_{(s,d)}$ *is consistent if:* M_{phys} *is consistent, and if the properties* wsCtrl, wsS2A, wsProt, wkCst, CstBind, CstDep *hold.*

A formal model of an application, which has these properties established, will be consistent by construction. This is the main idea implemented in the following section.

5 Checking the Consistency Properties Using Event-B

The abstract and generic model and properties defined in Sects. 3 and 4 should be implemented and analysed in a given formalism with its related tools, for any specific IoT-based application. We propose, using the Event-B formalism, a *parametrised Event-B model* as a generic base to model and analyse IoT-based applications. This parametrised approach is more interesting and more general than the straight translation or implementation of the abstract formal model in Event-B or in any other formal language; indeed, only a few part of the abstract model is specific to a given application, the remaining major part (is common to all applications and should) stay unchanged.

Overview of Event-B Modelling. Event-B [1,10] is a modelling and development method where components are modelled as *abstract machines* which are composed and refined into concrete machines called *refinements*. An *abstract machine* comprises a state space invariant and guarded events; it describes a mathematical model of a system behaviour as a discrete transition with the guarded events. *Proof obligations* are defined to establish model consistency via invariant preservation. Specific properties (included in the invariant) of a system are also proved in the same way.

5.1 A Parametrised Model for Consistency Checking of IoT Applications

We capture the common requirements and properties of IoT-based applications within an abstract and generic model; the analysis of the consistency properties does not depend on a specific application; it may be done through a generic parametrised model. We implement the generic model[2] in Event-B following the structure of a *parametrised model* interconnecting in a systematic way, a physical part and a control software part.

This justifies the structuring of our Event-B model where some *contexts* and *machines* are to be adapted to specific applications but other machines/contexts are defined once for all. The parametrised model, as a composition of Event-B components (see Fig. 2), is not only designed and used to implement our proposed method of modelling and analysis, it aims at being an easily reusable base. For this purpose, we adopted a layered structuring of the Event-B components in order to have a systematic approach for building, generating or extending the generic model. For extensibility, we consider categories of IOT-based systems; for instance, a home control category where the main components of systems are always the same: lights, windows, doors, garage,

[2] The complete Event-B development can be found at https://gitlab.univ-nantes.fr/attiogbe-c/iot_with_eventb.

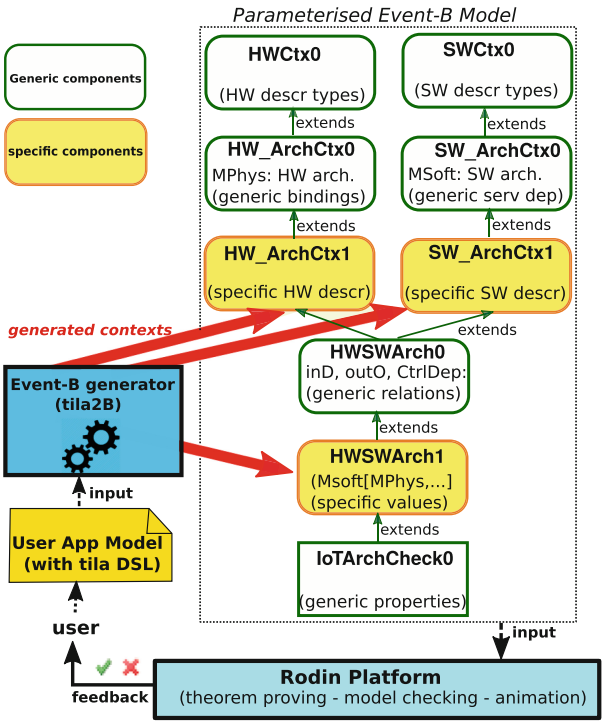

Fig. 2. The architecture of our generic Event-B modelling and analysis framework

heating, etc. Therefore the modelling components are not varying and can be gathered as reusable components in our formal modelling.

The generic architecture is depicted in Fig. 2. A first basic layer comprises fixed predefined Event-B contexts (whose names end with 0) which gather all elementary types and relations required in any application within a given category. Another layer comprises Event-B contexts and machines (whose names end with 1) which are the specific instantiations of the predefined contexts. Considering the category of home automation applications, at the hardware level, the context HWCtx0 contains the basic sets (LIGHTSENSOR, MOTIONSENSOR, LIGHTACTUATOR, etc.); at the software level the context SWCtx0 contains all the basic sets (SERVICE, CONTROLLER) for the applications of this category.

The context HW_ArchiCtx0 implements M_{phys}; it contains the generic structuring of a physical architecture (the formal bindings between the devices); that is, the relations $binding_{(d,s)}$ and $binding_{(a,d)}$ (see Sect. 3.2); similarly SW_ArchiCtx0 implements M_{soft}; it contains the generic structuring of the software part (with *servDepend*).

The context HW_ArchiCtx1 contains a specific instantiation for the physical architecture; it comprises the declarations of the objects (the sensors of each type, the needed controllers, etc.) and their assembly in a given application. Similarly, SW_ArchiCtx1 contains a specific instantiation of the software part; it comprises the controllers and the services on which they depend. Thus, only these two contexts will be modified to con-

sider new instances of physical or software part. The generic interconnection between the two parts (with the relations inD, $outO$, $CtrlDepend$, see Sect. 3.2) is implemented with the Event-B context HWSW_Archi0. In the same way, the context HWSW_Archi1 models a specific instantiation; it is the only context to be modified in order to build an interconnection for a specific control application; it gathers M_{soft} and the four parameters M_{phys}, inD, $outO$, $CtrlDepend$.

The analysis machine (IoTArchiCheck0, detailed in Fig. 3) is defined once for all; it contains the invariant properties (wsHW, wkCst, CstBind, CstDep, wsS2A defined in Sect. 4.1) to be checked for any given IoT-based application.

Each specific application is given as a parameter (HWSW_Archi1) of IoTArchiCheck0; that justifies the structuring with the Event-B SEES clause: a way to implement the genericity. Note that the effective implementation of an example system, is orthogonal to the preliminary step of consistency checking. If the IoTArchiCheck0 machine parametrised with HWSW_Archi1 is proved correct, then all the architectural and consistency properties are satisfied and consequently the related model is consistent.

MACHINE IOTArchiCheck0
SEES HWSWArchi1
INVARIANTS
wsHW: $\langle$theorem$\rangle$ $(binding\_ds \neq \{\}) \wedge (binding\_ad \neq \{\})$
wsS2A: $ran(inD) \subseteq dom(outO)$
wkCtrl: $dom(inD) \subseteq SENSOR \wedge ran(outO) \subseteq ACTUATOR$
CstBind: $(inD; outO) = (ctrlDepend; (binding\_ad^{-1}))$
wsCtrl: $(inD \neq \varnothing \wedge (dom(SENSOR \lhd inD) \subseteq SENSOR)) \vee (outO \neq \varnothing \wedge (dom(CONTROLLER \lhd outO) \subseteq CONTROLLER))$
EVENTS ...
END

Fig. 3. The generic analysis Event-B machine

The aim is now, given a model describing an IoT-based application, to prove at least all the properties of interest listed above. We use Rodin to discharge these proofs. But, the users can add other specific properties as needed. To facilitate the use of our method, it is supported by a tool-assisted process (where the contexts HW_ArchiCtx1, SW_ArchiCtx1, HWSW_Archi1 are generated) that is presented below.

5.2 Putting into Practice and Assessment

To ease the modelling and the analysis of IoT-based applications using our approach, we design a process to support it and to facilitate its reuse. We define τila, a tiny IoT domain specific language that helps to describe any application, by defining the used objects, their relations and their behaviours. Then we build a tool τila2B[3], that uses as

[3] https://gitlab.univ-nantes.fr/attiogbe-c/iot_with_eventb.

input the description in the *τila* DSL and generates the Event-B models as the specific parameters (HW_ArchiCtx1, SW_ArchiCtx1, HWSW_Archi1) to be plugged in our parametrised model, (see Fig. 2); the resulting Event-B model can then be submitted to Rodin for analysis. We demonstrate our generic model and the process with several examples. As for the analysis process, if the machine IoTArchiCheck0 is proved correct using Rodin then the model used as parameter is consistent.

6 Related Work

Existing related often take into account a specific concern (QoS, security, time performanxe, protocols, etc.), and as such they can be considered as complementary. But, in our knowledge there is no widely shared abstract model that can help the interoperability between the existing proposals and results. We target this objective by proposing, compared to some of the existing works, an open and extensible abstract model. In [20] the authors compare state-based and rule-based models of smart home apps for analysis scalability purpose. Their models are not generic and they focus on detecting misleading coordinations of components. A comprehensive survey in [9], emphasizes security aspects, proposes a hybrid security analysis system, but also shows that few attention are paid to abstract models and verification aspects. In [5] the authors introduce SysML4IoT to define a model compliant with the IOT-A reference model, and they translate the SysML model into NuSMV programs for the analysis concern. Their focus was on the verification of quality of service (QoS) properties. The authors of [14] focus on a multiview modelling together with workflows for implementing cloud-based Industrial IoT systems. For the modelling they combine several views through various models, and integrate them using the Automation Markup Language; they chose Uppaal for verification aspects and combine the Uppaal Timed Automata models with action patterns of timing behaviour to verify the timing performance to guarantee timing properties. The concerns of [7,16] are related to IoT services for health-care. In [15] the authors propose a development methodology and an associated framework to ease the development of IoT applications, but formal analysis was not their concern. In [3,11] verification of communication protocols such as MQTT are dealt with; Timed process-algebra [3] and Probabilistic timed automata and statistical model checking [11] are used for this purpose. In [13] the authors focus on the verification of the correct composition of IoT objects described as labelled transitions equipped with input and output interfaces. The objects composition results in a synchronous composition of the objects LTS leading to a composite service; then the notion of (concurrent processes) compatibility is used to ensure that the composite service has a correct interaction of its component services. Well-composed objects are then deployed on the basis of their mutual dependency. We share the formal modelling and verification objectives with [13], but our approach is more focused on building correct control-oriented IoT-applications; while their composition is restricted to binary parallel composition of object behaviours, we propose a more flexible global interaction based on the sense-decision-control paradigm which ensures flexible n-ary composition. We do not deal with deployment aspects, we rather provide means to generate the model, to analyse and simulate the targeted applications.

7 Conclusion

We have designed an abstract formal model of IoT-based applications learning from the common properties of IoT-based systems. An IoT-based application is then structured in a generic way by distinguishing different parts related to the physical part, the software part and the interconnection relations between both parts. These different parts serve as parameters in order to favour extension and reusability of the abstract formal model. We make explicit in the model, the architectural and consistency properties that have been formalised as the invariants of many IoT-based applications. These properties are systematically checked during the construction of an application. We then proposed a parametrised Event-B model as an implementation of our generic abstract model. The Event-B implementation is used for experiments that confirm the effectiveness of the proposed approach. To facilitate the modelling of IoT-based application and the reuse of our analysis approach, we design a tiny IoT application description language (τila) and a tool that generates for a given application expressed in τila, the Event-B components to be used to instantiate the parametrised model; thus the process is fully automatised.

Perspectives. Considering that IoT-based applications are a subset of cyber-physical systems, mostly characterised by their heterogeneous features, the proposed method here may be generalised to heterogeneous systems. For this purpose, we plan to connect our framework with existing DSLs which enable one to describe IoT systems; their descriptions will thus benefit from the formal modelling and the rigorous analysis of the designed systems prior to implementation. We already identified such DSLs for further investigation: openIoT [12], SDL-IoT [18], ide4dsl [17], UML4IoT [21], SysML4IoT [5,14], OpenHAB[4].

In order to ensure the robustness of the physical part, both its abstract model and its physical implementation may coexist during the live of the IoT system; both interacting with the sensors and actuators environment, in order to anticipate defects and also security issues. The abstract model, extended for these needs, will then behave as the digital twin of the real system and will enable one to check and to monitor it. Such interaction between models of different abstraction levels is planned for future work.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Al-Fuqaha, A.I., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015)
3. Aziz, B.: A formal model and analysis of the MQ telemetry transport protocol. In 2014 9th International Conference on Availability, Reliability and Security, pp. 59–68 (2014)
4. Banks. A., Gupta, R.: MQTT Version 3.1.1Plus Errata 01. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf. OASIS Standard Inc. (2015)
5. Costa, B., Pires, P.F., Delicato, F.C., Li, W., Zomaya, A.Y.: Design and analysis of iot applications: a model-driven approach. In 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, pp. 392–399 (2016)

[4] https://www.openhab.org/docs/.

6. da Cruz, M.A.A., Rodrigues, J.J.P.C., Al-Muhtadi, J., Korotaev, V.V., de Albuquerque, V.H.C.: A reference model for internet of things middleware. IEEE Internet Things J. **5**(2), 871–883 (2018)
7. Fattah, S.M.M., Sung, N.M., Ahn, I.Y., Ryu, M., Yun, J.: Building IoT services for aging in place using standard-based IoT platforms and heterogeneous IoT products. Sensors **17**(10), 2311 (2017)
8. Guth, J., et al.: A detailed analysis of IoT platform architectures: concepts, similarities, and differences. In: Di Martino, B., Li, K.-C., Yang, L.T., Esposito, A. (eds.) Internet of Everything. IT, pp. 81–101. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-5861-5_4
9. Hamza, A.A., Abdel Halim, I.T., Sobh, M.A., Bahaa-Eldin, A.M.: HSAS-MD analyzer a hybrid security analysis system using model-checking technique and deep learning for malware detection in iot apps. Sensors **22**, 1079 (2022)
10. Hoang, T.S., Kuruma, H., Basin, D., Abrial, J.R.: Developing topology discovery in Event-B. Sci. Comput. Program. **74**(11–12), 879–899 (2009)
11. Houimli, M., Kahloul, L., Benaoun, S.: Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things. In: 2017 International Conference on Mathematics and Information Technology (ICMIT), pp. 214–221 (2017)
12. Kim, J., Lee, J.: OpenIoT: an open service framework for the Internet of Things. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 89–93. IEEE (2014)
13. Krishna, A., Le Pallec, M., Mateescu, R., Noirie, L., Salaün, G.: Rigorous design and deployment of IoT applications. In: Proceedings of the 7th International Workshop on Formal Methods in Software Engineering, FormaliSE@ICSE 2019, Montreal, QC, Canada, 27 May 2019, pp 21–30 (2019)
14. Muthukumar, N., Srinivasan, S., Ramkumar, K., Pal, D., Vain, J., Ramaswamy, S.: A model-based approach for design and verification of industrial internet of things. Future Gener. Comput. Syst. **95**, 354–363 (2019)
15. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. J. Syst. Softw. **103**, 62–84 (2015)
16. Salahuddin, M.A., Al-Fuqaha, A., Guizani, M., Shuaib, K., Sallabi, F.: Softwarization of IoT infrastructure for secure and smart healthcare. IEEE Comput. **50**(7), 74–79 (2017)
17. Salihbegovic, A., Eterovic, T., Kaljic, E., Ribic, S.: Design of a domain specific language and IDE for Internet of things applications. In: 2015 38th International Conference on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp, 996–1001 (2015)
18. Sherratt, E., Ober, I., Gaudin, E., Fonseca i Casas, P., Kristoffersen, F.: SDL - the IoT language. In: Fischer, J., Scheidgen, M., Schieferdecker, I., Reed, R. (eds.) SDL 2015. LNCS, vol. 9369, pp. 27–41. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24912-4_3
19. Sosa-Reyna, C.M., Tello-Leal, E., Alabazares, D.L.: Methodology for the model-driven development of service oriented IoT applications. J. Syst. Architect. - Embed. Syst. Des. **90**, 15–22 (2018)
20. Stevens, C., Alhanahnah, M., Yan, Q., Bagheri, H.: Comparing formal models of IoT app coordination analysis. In: ACM SIGSOFT WOrkshop on Software Security (SEAD'20), pp. 3–10. ACM (2020)
21. Thramboulidis, K., Christoulakis, F.: UML4IoT-A UML-based approach to exploit IoT in cyber-physical manufacturing systems. Comput. Ind. **82**, 259–272 (2016)