An Effective Approach for Home Services Management

Patrice Moreaux, Fabien Sartor, Flavien Vernier

An Effective Approach for Home Services Management

P. Moreaux
LISTIC, Université de Savoie
74944 Annecy-le-Vieux, France
patrice.moreaux@univ-savoie.fr

F. Sartor
LISTIC, Université de Savoie
74944 Annecy-le-Vieux, France
fabien.sartor@univ-savoie.fr

F. Vernier
LISTIC, Université de Savoie
74944 Annecy-le-Vieux, France
flavien.vernier@univ-savoie.fr

Abstract

Domotic systems aim to offer functionalities like energy management, security, conveniences, and much more. Many domotic networks exist to provide a subset of these applications, but these networks are not necessarily compatible due to different communication mediums or protocols. Literature presents different studies that introduce high-level systems that solve the lack of incompatibility, but it does not explore how to create a network behavior. This paper concentrates on studying how to model the unit behaviors of home devices and a global behavior of a network of these devices: the automated living area. The global behavior is set-up with rules and constraints. The behaviors are modeled with an extended automaton input-output symbolic transition systems. To finish this paper, a use case shows the interest of building global behavior with unit behaviors.

1 Introduction

Domotic aims to offer functionalities like energy management, security, conveniences, and much more. Home automation covers a wide range of independent applications like opening control, light monitoring, access control... Many domotic networks exist to provide and to manage a subset of these applications, but these networks are not necessarily compatible due to different communication mediums or protocols. Thus, home automation is nowadays a set of heterogeneous protocol networks that cannot communicate together.

Literature presents different studies that introduce high-level systems to solve the lack of incompatibility. Through these systems, the access to a device using a protocol from other protocols is possible. To reach the domotic objectives, these systems are essential but not sufficient. From the users perspective, the domotic system must react to its environment. These reactions are the behaviors of the domotic network and are constructed from the behaviors of each device and rules expressed by the user.

This paper proposes models and methodologies to automatically construct domotic network behaviors. This contribution extends the formal Input Output Symbolic Transition Systems (IOSTS) model [2] to characterize the behaviors of communicating devices and the behaviors of the global network. The methodology to produce final models from rules is given. The rules can be constraints on communicating devices, or reactions between them. The produced models are IOSTS models, thus they are compatible with IOSTS model-checking methodology [3].

This paper is organized as follows: In Section 2, the classical domotic networks and the research to interconnect these networks are presented. Section 3 introduces two models. The first one, the device behaviors automata, models the behaviors of each device to express the knowledge about the devices for the domotic system. The second one, the reflex automata, permits the domotic system to react on events. Section 4 presents how to translate a constraint on a device to a reflex automata. This section also presents how to interconnect remote devices through a reflex automata. This paper is concluded and different future works are presented in Section 5.

2 Domotic Background

Automated living area means a dwelling and its surrounding equipped with communicating devices. A device can be an actuator to interact with its environment, a sensor to acquire data from its environment or an interface to exchange data with a user. A Domotic System (DS) provides a set of services that interact with devices according to predefined rules. The rules are defined with concepts that can be interpreted by human (watching TV, romantic ambience, secure home...). The DS services are based on high-level functionalities, but there is no standard network infrastructure that can provide all the range of these functionalities. Thus, the DS assures the translation between the services and the network infrastructures.

Home automation products have been on the market since 1975 with X10 product. Other solutions were devel-

oped, and are still being developed, but there is no consensus to adopt a standard network architecture. Dedicated network architectures were developed to increase the functionalities like Konnex (KNX), or to reduce electrical consumption like ZigBee (ZB) and Z-Wave(ZW). Some of them are open like ZB, X10 and KNX: manufactures can develop their own hardware. And others are closed like ZW: manufactures need a special ZW component to use the network. Now, with the introduction of IPv6 and standardized low-power link like 802.15.4, IP architecture becomes a research field for sensor networks domain [6]. More details about IP and communication devices are presented in [10].

All these technologies are not compatible. To integrate different network technologies and their devices, the DS needs a way to communicate with all home devices. In the literature, systems are studied and developed to integrate devices from different network architectures into a single abstract network. To provide abstraction of network technologies, the DS uses a distributed or a centralized system.

The first possibility is to agree with a common set of network technologies and data format to interact with upper services. One of the best example of an agreement of a common set of networks is UPnP [7]. The second possibility to create interoperability is to abstract the network technologies with centralized system that accesses to each domotic network and translates from/to its own language each command/notification to/from the destination/source domotic language. In this case, the DS uses an existing language as its own language or redefines a new one [1, 8].

As the DS services need to adapt their behaviors of an automated living area, they need to know the current state of the environment: the context [5]. In [9], authors compare methods to model the context, they underline that the context modeling based on ontologies offer a better level of formality and can reason. A modeling based on ontologies is the method chosen by Dog [1]. Dog aims to create an intelligent gateway to handle communicating object. It uses an ontology, DogOnt [4], to describe automated living area and model this way to communicate. Dog and DogOnt are available at `http://elite.polito.it/`.

In this paper, models are presented to manage the behaviors of a DS. Our work uses DogOnt [4] to abstract the devices from various network technologies. In this way, we extend the capacities of DogOnt by introducing how evolve the devices.

3 Home Behavior Models

The communicating devices have their own way to communicate depending on their network technologies. Our work requires mechanism that shares the status between the devices. Indeed, the DS needs to know the context of the automated living area to bring it a behavior. These communications are asynchronous, the responses can take time and during this time, the state of the device is unknown and another command can break the execution of a previous one.

A particularity of an automated living area is the number of sources of control. Indeed, each user and the DS have a direct access to each communicating device. Thus, the DS and each user can control a communicating device at the same time. That is why there is not always the expected response of a command. The DS needs a mechanism to describe the behavior of a device, to manage the asynchronous communications and the multiple control points. The IOSTS automaton [2] are chosen to model all these requirements.

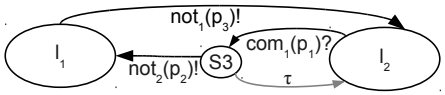
3.1 Device Behavior

To manage a set of devices, the DS needs a knowledge about the behavior of these devices. We introduce the Behavior Automaton (BA) to express this knowledge. Let us note that a BA is the device behavior from the point of view of the DS, and not from the point of view of the device. This remark is very important due to the fact that the communication between the DS and the device is asynchronous and a device can be commanded from many others command systems like remote controls. Thus, the DS cannot be sure that a command sent to a device is executed until it receives an acknowledgement.

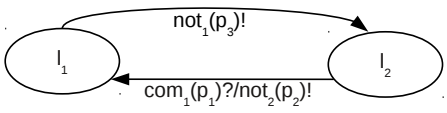
According to this remark, a state change can occur in two cases. The first one is when the DS sends a command and the device executes it. The second one is when a device receives a command from another command system. In the former, a particular case can appear when the command is received – in the point of view of the DS – but never executed. In the later, the DS does not send a command, but receives a notification.

To model this behavior, we define the BA (see Definition 2) a particular IOSTS [2] (see Definition 1). A notification is an output action: the device sends a notification. It is graphically identified by the symbol "!". As a notification is from the device, the destination state (or location in our model) resulting of this action is known. Thus, in a BA the destination of output action is always a known location that we call stable location $l \in \underline{L}$. A command is an input action that is graphically identified by the symbol "?". The DS sends a command, considers the device receives the command, but does not know if it will be executed. In this case, the DS cannot know in which state or location the device is after sending a command. Thus, a virtual unstable location $l \in \overline{L}^v$ is introduced to represent the fact that the system does not know the state of a device. According to this criterion, in a BA the origin of an input action is always a stable location – the system knows where it is when it wants to send a command – and the destination of an input

action is always a virtual unstable location. Figure 1(a) illustrates a generic device behavior model without guard or assignment. As the command transition – an input action



(a) BA model of a commanded device.



(b) Short model of a command path.

Figure 1. IOSTS and Short model of a BA.

– is always built in the same model: from a stable location to a virtual unstable location and followed by a notification – an output action – from the virtual unstable location to another stable location if the command is executed or followed by an internal retro-action from the virtual unstable location to the original stable location, this particular path can be simplified in the notation (see Figure 1(a)).

Thus, to simplify the model, let us note $T^{?!}$ the set of transitions – short representation – "$com_i?/not_j!$". According to this representation the transitions $\langle l_2, com_1(p_1)?, g_1(p_1), a_1, s3\rangle$ and $\langle s3, not_2(p_2)!, g_2(p_2), a_2, l1\rangle$ can be rewritten as $\langle l_2, com_1(p_1)?/not_2(p_2)!, g_1(p_1)\&g_2(p_2), (a_1.a_2), l1\rangle$

Definition 1 *An IOSTS is a tuple $\langle D, \Theta, L, l_0, \Sigma, T\rangle$ where:*

- *D is a finite set of typed Data, partitioned into a set V of variables and a set P of parameters. For $d \in D$, type(d) denotes the type of d;*
- *Θ is the initial condition, a predicate on V ;*
- *L is a non-empty, finite set of locations and $l_0 \in L$ is the initial location;*
- *Σ is a non-empty, finite alphabet, which is the disjoint union of a set $\Sigma^?$ of input actions, a set $\Sigma^!$ of output actions, and a set Σ^τ of internal actions. For each action $a \in \Sigma$, its signature $sig(a) = \langle p1, ..., pk\rangle \in P^k (k \in \mathbb{N})$ is a tuple of parameters. The signature of internal actions is the empty tuple;*
- *T is a set of transitions. Each transition is a tuple $\langle l^o, a, G, A, l^d\rangle$ made up of:*
  - *$l^o \in L$, the origin of the transition;*
  - *$a \in \Sigma$, the action of the transition;*
  - *G on $V \cup sig(a)$, the guard predicate;*
  - *A, an assignement which is a set of expressions in the form $(x := A^x)_{x\in V}$ such that, for each $x \in V$, the right-hand side A^x of the assignment $x := A^x$ is an expression of $V \cup sig(a)$[1];*
  - *$l^d \in L$, the destination of the transition.*

Definition 2 *A BA is a tuple $\langle D, \Theta, L_{BA}, L_0, \Sigma, T\rangle$ where:*

- *$L_{BA} = \underline{L} \cup \overline{L}^v$ (or $L_{BA} = \underline{L}$ in short representation) is a non-empty, finite set of locations ($L_{BA} \subset L$) and L_0 is the set of possible initial locations ($L_0 = \underline{L}$ in most cases).*
- *$T = T^! \cup T^? \cup T^\tau$ is a set of transitions. Each transition in $T^\#$ is a tuple $\langle l^o, a, G, A, l^d\rangle$ where $a \in \Sigma^\#$ and $\# \in \{?, !, \tau\}$. Moreover, $\forall\langle l, a, G, A, l'\rangle \in T^\tau, G = \top, A = \emptyset, l \in \overline{L}^v, l' \in \underline{L}$.*
- *$\underline{L}$ is the non-empty set of stable locations such that: $l \in \underline{L} \iff \forall\langle l', a, G, A, l\rangle, a \in \Sigma^! \cup \Sigma^\tau$.*
- *$\overline{L}^v$ is the set of virtual unstable locations such that: $l^v \in \overline{L}^v \iff \exists!\langle l^o, a^?, G, A, l^v\rangle$ with $a^? \in \Sigma^? \wedge \exists!\langle l^v, a^\tau, \top, \emptyset, l^o\rangle$ with $a^\tau \in \Sigma^\tau \wedge \exists!\langle l^v, a^!, G', A', l^d\rangle$ with $a^! \in \Sigma^!$.*

The BA permits the DS to know how a device evolves. With this knowledge about the devices, the DS can define the path from a state to another (see Figure 2). This path is

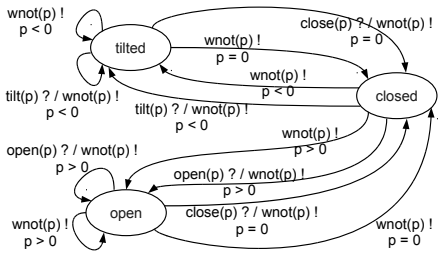


Figure 2. T&T windows BA.

a command execution and can be defined as follows:

Definition 3 *A command execution is a sequence of alternating locations and input/output transitions $l^0t^0l^1 \dots t^{n-1}l^n \in L_{BA}.(T^{?!}_{BA}.L_{BA})^*$ such that $\forall t^i \quad l^o_{t^i} = l^i \wedge l^d_{t^i} = l^{i+1}$.*

3.2 Reflex Automata

The main action of a DS is to react to events. Thus, we introduce Reflex Automata (RA) to model a reactive system. A RA is a particular IOSTS that models the constraint on a device or the interactions between devices. It models the life behavior of the system managed by the DS. In a RA,

an unstable location can represent an unknown state, like in BA, it is called a virtual unstable location $\overline{l}^v$ or can represent a transient location that is stable in BA, in this case it is called transient unstable location $\overline{l}^t$. From the point of view of the DS, an unstable transient location represents a known state of the system in which the DS does not want to stay. The behavior of a RA always begins with a notification – an output action – that brings the system out of a stable location and is followed by a set of actions to move from an unstable transient location to a stable location using unstable locations. That means the stable locations can only be the source or the destination of notifications. Figure 4 and 5 illustrate use cases of a RA model.

Definition 4 *A RA is a tuple* $\langle D, \Theta, L_{RA}, L_0, \Sigma, T \rangle$ *where:*

- $L_{RA} = L_{BA} \cup \overline{L}^t = \underline{L} \cup \overline{L}^v \cup \overline{L}^t$ *(or* $L_{RA} = \underline{L} \cup \overline{L}^t$ *in short representation) is a non-empty, finite set of locations (*$L_{RA} \subset L$*) and* L_0 *is the set of possible initial locations (*$L_0 = \underline{L}$ *in most cases).*
- $\overline{L}^t$ *is the set of transient unstable locations such that:* $l^t \in \overline{L}^t \iff \exists \langle l^o, a, G, A, l^t \rangle$ *with* $a \in \Sigma^! \wedge \exists \langle l^t, b, G, A, l^d \rangle$ *with* $b \in \Sigma^?$

A RA permits the DS to react to an event. The RA is the knowledge about the reactions: it defines the reflex executions:

Definition 5 *A reflex execution is a sequence of alternating locations and input/output transitions* $l^0 t^0 l^1 \dots t^{n-1} l^n \in L_{RA}.(T_{RA}^{?!}.L_{RA})^*$ *such that* $\forall t^i \quad l^o_{t^i} = l^i \wedge l^d_{t^i} = l^{i+1}$. *A reflex execution always starts from an unstable transient location* $l^0 \in \overline{L_{RA}}^t$*, stops on a stable location* $l^n \in \underline{L_{RA}}$ *and any intermediate location is an unstable transient location* $l^{i, i \neq 0 \wedge i \neq n} \in \overline{L_{RA}}^t$.

4 Home Behavior Description

The behavior of a system can be defined by two concepts: interactions between devices and constraints that limit the possible behavior of a device.

4.1 Event Condition Change

An Event Condition Change (ECC) is used to command a device from another. For example, a device button (see Figure 3) commands a device light. This approach is very flexible and dynamic: a device can command many devices and it can be also commanded by many devices.

The ECC is a generalization of the classical Event Condition Action (ECA) introduced in the database. An ECC expresses that, when the event E occurs in a device and the condition C is satisfied, the change C must be applied in another device. The ECC introduces two types of changes, the classical action of ECA and the goal. A goal is a state for which the set of actions to reach is unknown. Then, the Event Condition Goal (ECG) is also introduced.

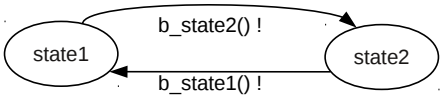


Figure 3. Two states button BA.

4.1.1 Event Condition Action

An Event Condition Action (ECA) rule is defined as a tuple: $\text{eca} = \langle \lambda! \,/\, c \,/\, \alpha? \rangle \in (\Sigma^! \times C \times \Sigma^?)$, where $\Sigma^!$ is from the BA $\mathcal{A}1_{BA}$ of a device and $\Sigma^?$ is from the BA $\mathcal{A}2_{BA}$ of another device. Moreover, we impose the event is not in the alphabet of the commanded device: $\lambda! \in \Sigma^!_{BA1} \wedge \lambda! \notin \Sigma^!_{BA2}$.

4.1.2 Event Condition Goal

An Event Condition Goal (ECG) rule noted $\langle \lambda! \,/\, c \,/\, \gamma \rangle$ is given by the event $\lambda!$, the condition c and the goal γ. A goal γ is a state $s \in l$ where $l \in \underline{L}$ and $\underline{L} \in \mathcal{A}_{BA}$ with $\mathcal{A}_{BA} = \langle D, \Theta, L, l_0, \Sigma, T \rangle$ the BA of the commanded device. A state is defined by a location and the values of V. The RA $\mathcal{A}_{RA}$ corresponding to an ECG rule $\langle \lambda! \,/\, c \,/\, \gamma \rangle$ is deduced from the BA $\mathcal{A}_{BA}$ of the commanded device. The RA contains the stable locations of BA, the output actions of the BA from a stable location to another stable location and all reflex executions from any location to the goal (see Figure 4).

4.2 Constraint Expression

The constraints are used to restrict the behavior of a device. The expression of a constraint can be written in many logic languages. In this paper, the constraints are only applied to the variables v_i such that $v_i \in V$ and only the comparisons between one variable and a constant are allowed. The comparison operators used are $(<, >, \geq, \leq, =, \neq)$ and the Boolean algebra $(\vee, \wedge, \neg)$ is used to link the comparisons.

Let us recall that a device can be commanded by another system. Thus, the device can violate the constraint. In this case the DS must correct the state of the device to warrant the constraint. To define this correction, a RA is built from the BA of the device and the constraint on this device.

The RA contains the stable locations of BA that satisfy and do not satisfy the constraint and the output actions of the BA from a stable location to another stable location. The locations of the BA that partially satisfy the constraints

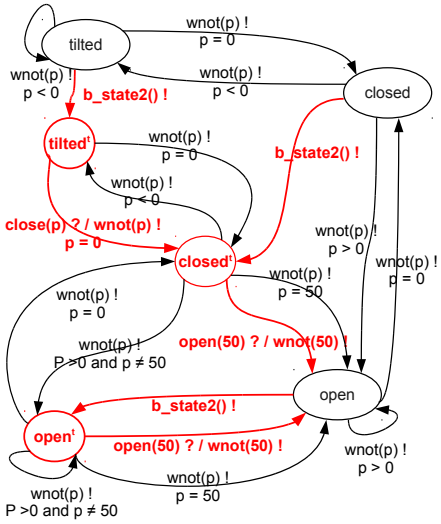


Figure 4. RA of T&T window commanded by the button with ECG $\langle b\_etat2()!/ \top / val_f = 50\rangle$.

are split in two: satisfying and not satisfying the constraint. In this way, in the RA a location satisfies either or not the constraint. Each location that does not satisfy the constraint is duplicated such that only one output action comes into it and each duplicate is an unstable transient location. The goal of the RA is to define the actions to come back into the previous location that satisfies the constraint when this one is broken. Thus, in the RA, for any output actions that go into a location that does not satisfy the constraint, a reflex execution comes back into the previous location that satisfies the constraint (see Figure 5).

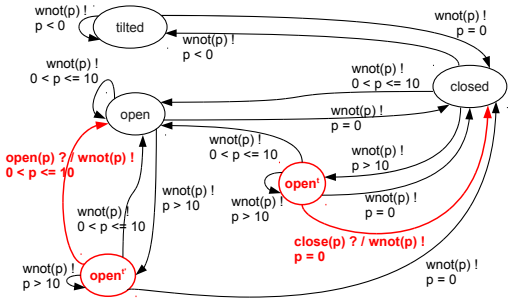


Figure 5. RA of constrained T&T window with $val_f <= 10$.

A particular case can appear if the system starts in a constrained state. In this case, there is no previous location that satisfies the constraint. Thus, a location l that contains all states that do not satisfy the constraints is added to the transient locations and to the possible initial location, a rescue state that satisfies the constraint is defined and reflex executions are defined to go from l to the rescue state.

5 Conclusion and Future Work

In this paper, models to manage by Behavior Automata (BA) and to interconnect by Reflex Automata (RA) the behavior of different devices in an asynchronous network are presented. The behavior of a system is defined by the behavior of each device, constraints on these devices and the interactions between devices. The BA is used to model the possible behavior of a device. The RA is used to describe the reflex commands that the Domotic System (DS) must execute. The RA is built from a BA and a constraint or a conditioned event.

In this paper a constraint is applied on only one device and an event links only two devices. In the future work, the definitions of constraint on many devices and event reactions on more than two devices are planned. Another point is that in this approach a goal is a state. As one of the objectives is to reach a goal expressed by a human expression, a goal as location will be studied.

References

[1] D. Bonino, E. Castellina, and F. Corno. Dog: An ontology-powered osgi domotic gateway. In *ICTAI*, pages 157–160. IEEE Computer Society, 2008.

[2] C. Constant, T. Jéron, H. Marchand, and V. Rusu. Integrating formal verification and conformance testing for reactive systems. *IEEE Transactions on Software Engineering*, 33(8):558–574, Aug. 2007.

[3] C. Constant, T. Jron, H. Marchand, and V. Rusu. Validation of reactive systems. *Modeling and Verification of Real-TIME Systems - Formalisms and software Tools*, pages 51–76. Herms Science, Jan. 2008.

[4] F. C. Dario Bonino. Dogont - ontology modeling for intelligent domotic environments. In *International Semantic Web Conference*, pages 790–803, 2008.

[5] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[6] J. Hui and D. Culler. Ipv6 in low-power wireless networks. *Proceedings of the IEEE*, 98(11):1865 –1878, Nov. 2010.

[7] S. H. Kim, J. S. Kang, K. K. Lee, H. S. Park, S. H. Baeg, and J. H. Park. A upnp-zigbee software bridge. In *ICCSA*, pages 346–359, 2007.

[8] G. Nain, E. Daubert, O. Barais, and J.-M. Jézéquel. Using mde to build a schizophrenic middleware for home/building automation. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet*, ServiceWave '08, pages 49–61, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[10] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP - The Next Internet*. Morgan Kaufmann, 2010.