Discrete Control for the Internet of Things and Smart Environments

Mengxuan Zhao, Gilles Privat *Orange Labs, Grenoble, France* *(firstname.lastname@orange.com)*
Eric Rutten *INRIA, Grenoble, France* *(eric.rutten@inria.fr)*
Hassane Alla *GIPSA Lab, Grenoble, France* *(hassane.alla@gipsa-lab.grenoble-inp.fr)*

Abstract

The Internet of Things (IoT) requires self-configuration capacities, for which there is a need for design techniques for predictable controllers, and automation in the construction of these controllers. The Feedback Computing approach to autonomic systems proposes to exploit control techniques for this. We present preliminary results in our approach using Discrete Supervisory Control for the generation of the supervisory controllers in IoT and smart environments. A general modeling framework is proposed for the application domain of smart home/building. We formalize the design of the autonomic manager as a Discrete Controller Synthesis (DCS) problem, w.r.t. multiple objectives. We validate our models and manager computations with the BZR language and an experimental simulator.

keywords: Internet of Things, Smart environments, Autonomic management, Discrete Control.

1 Introduction

IoT and Smart Environments. The Internet of Things (IoT) is more than the well-known applications (such as supply chain management) or the technologies (such as RFID) to which it often gets narrowed down. Smart environments have, on the other hand, mostly been studied as an extension and evolution of traditional device-based human interfaces [1, 11] , with a research agenda extending that of traditional computer vision. Much as robots sense and act upon their outer environments, smart environments can be viewed in a non human-centric way as outside-in robots, i.e. environments that sense and act upon their own inner space. Both the IoT and smart environments are, at heart, the result of massive instrumentation of the environment through networked sensors and actuators that serve as monitoring and control intermediaries either to things of directly to the environment itself. Both herald a new relationship between ICT and the physical world, way beyond traditional embedded computing and present-day *M2M* telecom applications. They should be seen, for our purposes, as a large-scale and readily available basis for experimenting a new breed of *Cyber-Physical Systems* [9]. A cyber physical system involves jointly distributed physical and information systems, doubly coupled for monitoring and control through networked sensors and actuators. Such systems have a long history in both automatic control and embedded computing, but the widespread availability of the IoT affords the distribution of such systems over large-scale environments using a shared infrastructure, and their generalization beyond their traditional applications. Due to the highly dynamic nature of environments such as smart homes, smart buildings or smart cities, and the constraints of new mass-market applications in these environments, the design of such systems cannot rely any more on the fully customized, ad hoc solutions that have been dominant so far in *industrial* automatic control and embedded systems. Networked devices and general-purpose software engineering show the way, with self-configuration mechanisms making it possible for distributed systems to adapt to dynamic environments by automatically bootstrapping and reconfiguring themselves, which is part of the self-management process of an autonomic system. The approach we propose is a first step towards bringing distributed control systems with autonomic management closer to these *plug-and-play* models.

Control techniques for autonomic management. We adopt control techniques to design the *MAPE-K* (Monitor, Analyze, Plan, Execute, based on Knowledge) autonomic manager [8]. Formal models are used to describe the possible behaviors of the system under design, and control objectives giving the adaptation policy are specified separately. A controller is then derived based on the system models and objectives. The use of classical control techniques, typically these based on continuous time dynamics and differential equations, has been

explored for various computing systems [4]. A similar approach can be adopted by using *discrete control* techniques, where systems are considered from the viewpoint of events and states. The behaviors can then be modeled in the form of Petri nets or automata for, typically, synchronization or coordination [12].

Discrete control for the IoT. In this paper, we apply *discrete control* for the autonomic management of the IoT and Smart Environments, with a case study in the smart home/building domain. We present first results towards a systematic modeling framework, where the behaviors of entities are modeled by using *Labeled Transition Systems* (LTS) or automata, and management policies and rules are given separately as control objectives, w.r.t. multiple objectives. *Discrete Controller Synthesis* (DCS), supported by a programming language and synthesis tool, is applied to compute an autonomic manager. The result is simulated to validate our proposal.

2 Background

2.1 Interfacing to the IoT and Smart Environments

Our target application domain is the intersection of the IoT and the Smart Environment where we are interested in methods to interface the ICT applications with the environment for monitoring and controlling individual entities such as rooms, appliances, with or without direct network interfaces.

In [6, 5] an intermediate abstraction layer EAL (Entity Abstraction Layer)is proposed for identifying and integrating such entities for which available sensors and actuators play the role of network interface, as illustrated in Figure 1. The automata models considered as dynamic proxies of the entities provide high-level interfaces for monitoring and control services, including supervisory controllers which will be detailed in next sections.

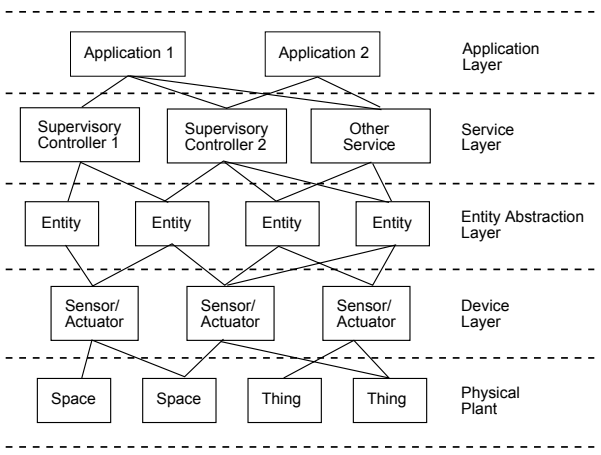

Figure 1: Global architecture of the overall system

Unlike the controllers in the application layer designed for the precise requirements in the given context, the supervisory controllers as service emphasize on genericity of the requirements valid in all kinds of environments considered in the application domain, and will provide high-level interfaces to the upper-level applications.

As the environment is constantly evolving and connected to the network by sensors and actuators, the autonomic computing approach can be appropriate to design the controller enabling the adaptivity and reconfigurability of the system.

2.2 Discrete control

Automata and data-flow nodes. We first briefly introduce the basics of the automata-based modeling framework (details in [3]). Behaviors are modeled in terms of Finite State Machines (FSM), or more precisely Labeled Transition Systems (LTS). They are defined by a finite set of states, between which there are transitions (from source state to target state) with a label of the form `c / a`: a firing condition `c` and an action `a`. At each step, when the FSM is in some current state, if there is a transition for which the condition is true, then it is taken and the next current state will be the target state. At the same time the action part will take the value `true`.

Figure 2(a) illustrates this with a reactive node for the control behavior of a delayable task. It can be idle, waiting or active. When it is in the initial Idle state, the occurrence of the `true` value on input `r` *requests* the starting of the task. Another input `c` can either allow the activation, or temporarily block the request and make the automaton go to a waiting state. Input `e` notifies termination. The outputs represent, resp., `a`: activity of the task, and `s`: triggering starting operation in the system's API.

Such automata and data-flow reactive nodes can be reused by instantiation, and composed in parallel (noted in the concrete syntax "`;`") and in a hierarchical way, as illustrated in the body of the node in Figure 2(b), with two instances of the `delayable` node. They run in parallel: one global step corresponds to one local step for ev-

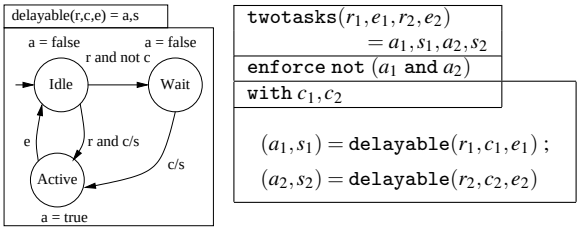

Figure 2: (a) Delayable task ; (b) exclusion contract.

ery node. The compilation produces executable code in target languages such as C or Java: an initialization function *reset*, and a *step* function implementing the transition function of the resulting automaton. It takes incoming values of input flows gathered in the environment, computes the next state on internal variables, and returns values for the output flows. It is called at relevant instants from the infrastructure where the controller is used.

Control and contracts. The formalism of LTSs can be used to apply *discrete controller synthesis* (DCS), a formal operation on automata [2]: given a FSM representing possible behaviors of a system, its variables are partitioned into controllable ones and uncontrollable ones. For a given control objective (e.g., staying invariably inside a subset of states, considered "good"), the DCS algorithm automatically computes, by exploration of the state graph, the constraint on controllable variables, depending on current state, for any value of the uncontrollables, so that remaining behaviors satisfy the objective. This constraint is inhibiting the minimum possible behaviors i.e., it is called *maximally permissive*.

The BZR language (`http://bzr.inria.fr`) has behavioral contracts [3] and its compilation involves DCS. Concretely, using the `with` statement, controllable variables are declared, the value of which are not defined by the programmer. These free variables can be used in the program to describe choices between several transitions. They are defined, in the final executable program, by the controller computed by DCS, according to the expression given in the `enforce` statement. BZR compilation invokes a DCS tool, and inserts the synthesized controller in the generated executable code, which has the same structure as above: *reset* and *step* functions.

Figure 2(b) shows an example of contract coordinating two instances of the `delayable` node of Figure 2(a). The `twotasks` node has a `with` part declaring controllable variables c_1 and c_2, and the `enforce` part asserts the property to be enforced by DCS. Here, we want to ensure that the two tasks running in parallel will not be both active at the same time: `not` (A1 `and` A2). Thus, c_1 and c_2 will be used by the computed controller to block some requests, leading automata of tasks to the waiting state whenever the other task is active. The constraint produced by DCS can have several solutions: the BZR compiler generates deterministic executable code by giving priority, for each controllable variable, to value `true` over `false`, and between them, by following the order of declaration in the `with` statement.

2.3 Discrete control as MAPE-K

Figure 3(a) shows the MAPE-K architecture of an autonomic system with a loop defining basic notions of Managed Element (ME) and Autonomic Manager (AM).

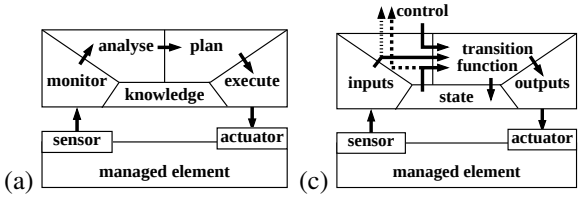

Figure 3: (a) MAPE-K manager; (c) controllable AM.

The managed element, system or resource is monitored through sensors. An analysis of this information is used, in combination with knowledge about the system, to plan and decide upon actions. These reconfiguration operations are executed, using as actuators the administration functions offered by the system API. Self-management issues include self-configuration, self-optimization, self-healing (fault tolerance and repair), and self-protection.

Autonomic managers work in closed loop: for this, one design methodology is to apply techniques from Control Theory [4], with the advantage of ensuring interesting properties on the resulting behavior of the controlled system e.g., stability, convergence, reachability or avoidance of some evolutions. Recently, some works relied on Discrete Event Systems (DES), using supervisory control [2], typically for logical or synchronization purposes e.g., deadlock avoidance in multithreaded programs [12]. They are based on reactive systems models such as Petri nets or Finite State Machines (FSM).

As shown in Figure 3(b), this instantiates the general autonomic loop with knowledge on possible behaviors represented as a formal state machine, and planning and execution in the form of the automaton transition function with a control output, which will trigger the actuator. In this context, observability and controllability correspond to, resp., outputs (exhibiting knowledge and sensor information, raw or analyzed, e.g., on state) and inputs (influencing the decision, used in the guards making choices between different transitions), as shown by dashed arrows in Figure 3(b).

3 DCS for IoT and smart environments

3.1 Target environments

The physical environments we consider comprise a set of entities of diverse nature, and their coexistence is managed by using observation and control possibilities they offer. We are interested in issues of safety, like having a source of light on in the case of a presence in the area. We are also interested in the way they consume instantaneous power, according to their possible local states, and the way cumulated power consumption can be managed at the level of a house, or a building. This total

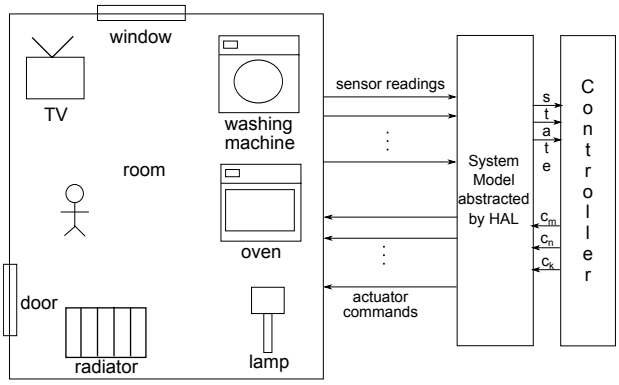

Figure 4: Example of controlled smart environment

power will be bounded by different threshold according to different policies which can be comfort (whatever the cost in power for example), or economy (but still keeping up with safety), or others, like maintaining minimal safety.We will introduce our approach by the example, and illustrate it with a case study shown in Figure 4.

Basic entities in the environment. In this apartment, a person is moving freely, and can open or close the door or the window, which have two states: open or closed. She can turn on/off the lamp, or the TV, which have two states: on or off. The radiator can be off, in frost-free mode, or eco mode or high mode. An oven can be off, or heating up (with high instantaneous power consumption), or maintaining its current temperature (with medium consumption). A washing machine proceeds through a number of phases, with different power consumptions (e.g., washing is more costly than rinsing); the sequence between phases can be suspended in some cases, offering a control point to a coordination manager.

Sensors and actuators. These physical entities are monitored and controlled by sensors and actuators. Available sensors are e.g.: presence in a room, door/window (open or closed), and electric current when e.g. the oven or the washing machine is started.

Actuators are smart plugs for switching on TV or lamp, suspend the wash machine...

Control objectives. Despite the heterogeneity in instances of environment in this specific application domain, the control objectives we are interested in can be classified in at least one of the 4 following categories: safety (have at least one source of light when there is movement in the room), security (closing room in the absence of occupants), energy efficiency (according to occupancy, or to parallel consuming activities, maintaining a global power consumption bound) and comfort. The controller should coordinate all the entities, adapting the controllable entities of a room to its occupancy and activities of the person, to accomplish all the objectives.

A typical scenario can be: when the washing machine is about to proceed to a next phase, at a point where it can be suspended, it is the occasion of suspending it, in order to prevent it to go to a global state where the global power, cumulated with that of the oven being turned on in parallel, would pass above a threshold defined for the economic management policy.

We consider specifically the following rules:

Rule 1: for safety, at least one light source is on when room is occupied;

Rule 2: for security, close window and door when room isn't occupied;

Rule 3: for energy, if window or door is open, the radiator is either off or in frost protection level;

Rule 4: for energy, if the room is not occupied, no light is on and the radiator is off or in frost protection mode;

Rule 5: for safety and energy, total power should be under the threshold configured through an input choosen among three possible values corresponding to different management policy: minimal-safety, comfort, eco.

It should be noted here that the kind of discrete supervisory control service we address here is fundamentally distinct from classical multiple-continuous-variable optimization that would, in our architecture framework, reside in the application layer. The ReActivHome project [7] developed such a comprehensive optimization application for home energy management, taking into account complex cost functions and multiple constraints on different time horizons. Such applications provide a much more fine-tuned optimization of energy use for efficiency purposes, but require a case by case design.

3.2 Formalization as a DCS problem

Due to space limitations, we detail mainly the interactions around the oven, wash machine and radiator, w.r.t. power management for energy efficiency objectives.

System behaviors model Controlled equipments are modeled as entities (automata) identified already completely in the abstraction layer (EAL) [5, 6] on which the supervisory controller relies. Control of the physical world is done via the intermediate EAL which interprets the enforced state transitions to actions towards the actuator interface, as illustrated in Figure 4. Due to space limitations, the door model will represent all the two-state models which are very similar to each other.

Door behavior. As in Figure 6(a), it has an initial state *Closed* and a second state *Open*. Input *push* is uncontrollable command from people while the other input *c_door* is controllable which will prevent the door from opening or force it to shut, to meet the objectives. it indicates the door's current state through output *door_open*.

Radiator behavior. As in Figure 5(a), it has an initial state off, a state $frostprotection$, a lower-energy state

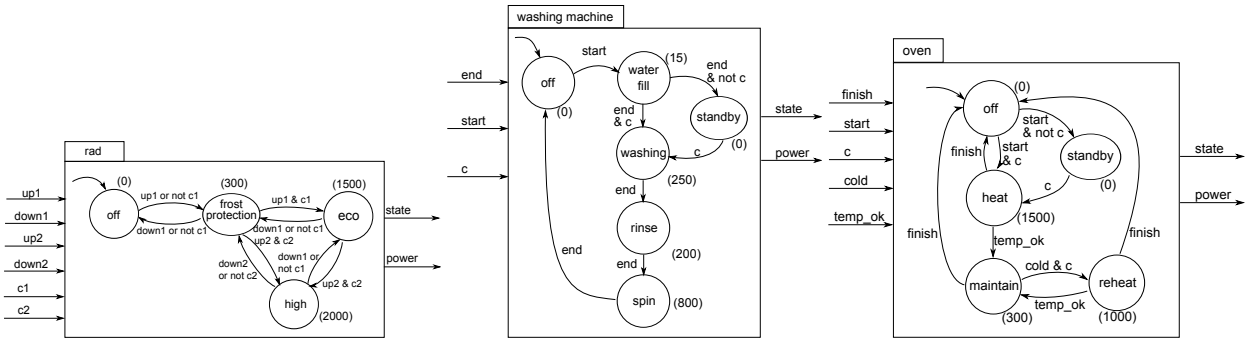

Figure 5: Entities behaviors models: (a) radiator ; (b) washing machine ; (c) oven

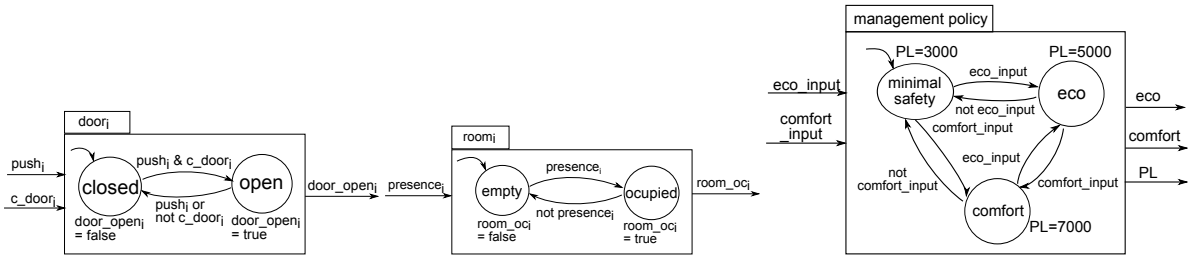

Figure 6: (a) Door behavior ; (b) Room model ; (c) 3 management policies

eco and a max-energy state *high*. $upi, downi, i \in \{1,2\}$ are external commands to put the radiator to the desired state. Controllable variables $ci, i \in \{1,2\}$ are integrated into the model to accomplish the control objectives. The transition between the state off and $frostprotection$ can be forced, while between $frostprotection$,*eco* and *high* the transition from lower to higher level is controlled and can be forced from higher to lower by *ci* to meet the energy objective w.r.t. the peak power constraint. Power consumption in states is given between parentheses.

Washing machine behavior. As in Figure 5(b), it has states for classic cycle steps: washing, rinsing and spinning. It begins upon *start*, and can be suspended to *standby*, using control variable *c*, before going into the high-energy-consumption sequence of states *wash-rinse-spin*. The power is given for each state. This behavior model is inspired by the one presented in [5, 6] which is designed initially to illustrate the identification, auto-configuration and monitoring features of EAL. Power consumption in states is given between parentheses.

Oven behavior. As shown in Figure 5(c), it can go from off upon *start*, depending on the control *c*, either to the quick heating, highly energy-consuming state *heat*, or to *standby*, waiting for *c*. When receiving notification *temp_ok* indicating that the desired temperature has been reached, it goes to *maintain* where it consumes little, until receiving *cold*, then to *reheat* if control *c* allows it. From any active state, the oven goes back to its initial state off at the receipt of $finish$. Power consumption in states is given between parentheses.

Room behavior. It is captured by the automata in Figure 6(b). It has two states *empty* and *occupied*. It takes input from the presence sensor(s) and keeps track of the room's occupancy by output *room_oc*. It is an observer model as it is uncontrollable.

Global system behavior model. The parallel composition of control models for 1 room, 1mode switcher, 1 door, 1 window, 1 lamp, 1 TV, 1 radiator, 1 oven and 1 washing machine gives the global system model, with its initial state. It represents all the possible system behaviors in the absence of control (yet to be integrated).

In the global behavior we define global power consumption functions, in terms of the local ones, in the following equations. The total peak power is defined as the sum of the power of the washing machine, of the radiator and of the oven in their current state.

$totalPower = p(wm) + p(ov) + p(rd)$

Switching between different management policies is captured by the automaton in Figure 6(c), where the power limit *PL* is defined on each of three states: $minimal\_safety$, $comfort$ and *eco*, representing the 3 possible policies chosen by the user. Input *eco_input* and $comfort\_input$ are uncontrollable. Similar to the room model, it is an observer of user's choice.

Control objectives. On these bases, we formalize the objectives of section 3.1 as follows:

Rule 1: $room\_oc \Rightarrow lamp\_on \vee tv\_on$

Rule 2: $\neg room\_oc \Rightarrow \neg(d\_open \vee w\_open)$

Rule 3:$(d\_open \vee w\_open) \Rightarrow (rad\_off \vee rad\_frost)$
Rule 4: $\neg room\_oc \Rightarrow$
$(\neg(lamp\_on \vee tv\_on) \wedge (rad\_off \vee rad\_frost))$
Rule 5: $totalPower \leq PL$

These five Boolean expressions are used as a control objective : DCS will generate a controller such that the system will remain invariantly inside the subset of its global states such that $\bigwedge_{i\in[1..5]} Rule_i$ is true.

4 Implementation and simulation

BZR encoding and DCS. The notation used in the models of Section 3.2 is very close to the BZR language [3]. The automata can be translated easily in BZR, and the global system behavior can be obtained by composing all these models. The costs on states are defined as in Section 3.2 (not detailed here, due to space limitation).

Taking as input the system model and the contract, the BZR compiler can synthesize a controller (in C or Java code) automatically satisfying the defined objectives. There is also a graphical tool enabling the users to perform simulations of the controlled system by combining the controller with the system model.

Simulation. We have developed MiLeSEnS (**Mi**ulti-**Le**vel **S**mart **En**vironment **S**imulator) as a testing ground based on Siafu, an open-source context simulator written in Java [10], which provides a GUI and basic and simple physical models such as moving people and physical areas. Though the models are *toys* compared to complete and realistic models, they provide enough environmental elements and interactions between actuators and sensors.

The Java code generated by the BZR compiler has 2 main functions: *step* and *reset*. *reset* initializes the state of the program and *step* executes one reaction where all necessary events are represented by an input of *step* which, after one execution, returns output values of current state of the program. To validate our approach, we did a simulation on MiLeSEnS which represents the target environment with all entities and possible system behaviors in absence of control.

The generated controller is associated by interfaces provided by the simulation. Unlike the executions without control where undesired system behaviors are observed, e.g. the radiator is heating with high power while the window is open, the controlled executions respect all the objectives. We can see on the GUI that, as illustrated in Figure 7, every time the person does some action or an event occurs, the system never goes to a forbidden state.

5 Conclusion and Perspectives

We have proposed first results in using discrete supervisory controllers applied to the IoT and smart homes and buildings. We apply formalisms supported by a programming language and synthesis tool, to encode and perform the computation of an autonomic manager as a DCS problem. We perform a validation of our proposal on a case study with modeling and simulation.

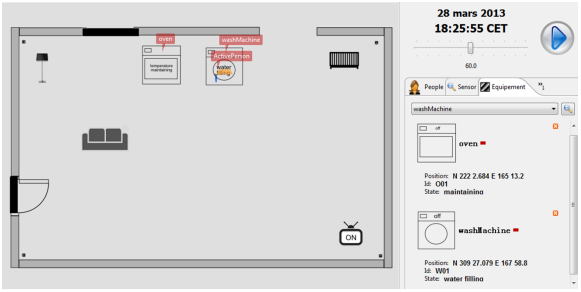

Figure 7: Simulation with synthesized controller

Perspectives include enriching our models with more types of entities and objectives, adaptive control, self-configurability (adapting to different automata for the same physical entity identified by HAL [6]), and to exploit the modular synthesis and compilation [3] in relation with the hierarchical structure of groups of entities and objectives, for scalability.

References

[1] Borkowski, S., and Privat, G. Spatial interaction in ambient communication. In *4th Int. Conf. on Enactive Interfaces* (2007).

[2] Cassandras, C., and Lafortune, S. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 2008.

[3] Delaval, G., Marchand, H., and Rutten, E. Contracts for modular discrete controller synthesis. In *Conf. on Languages, Compilers, and Tools for Embedded Systems* (2010), pp. 57–66.

[4] Hellerstein, J., Diao, Y., Parekh, S., and Tilbury, D. *Feedback Control of Computing Systems*. Wiley, 2004.

[5] Hu, Z., Frenot, S., Tourancheau, B., and Privat, G. Iterative model-based identification of building components and appliances by means of sensor-actuator networks. In *2nd Workshop on eeBuildings Data Models* (2011).

[6] Hu, Z., Privat, G., Frenot, S., and Tourancheau, B. Self-configuration of home abstraction layer via sensor-actuator network. In *Int. J. Conf. on Ambient Intelligence, AmI-11* (2011).

[7] Jacomino, M., and Le, M. Robust energy planning in buildings with energy and comfort costs. *4OR 10*, 1 (2012), 81–103.

[8] Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *IEEE Computer 36*, 1 (Jan. 2003), 41–50.

[9] Lee, E. A., and Seshia, S. A. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. LeeSeshia.org, 2011.

[10] Martin, M., and Nurmi, P. A generic large scale simulator for ubiquitous computing. In *Int. Conf. on Mobile and Ubiquitous Systems: Networking & Services (MobiQuitous)* (2006).

[11] Privat, G. Ambient communication, when devices disappear. In *7th Berliner Werkstatt Mensch-Maschine Systeme* (2007).

[12] Wang, Y., Lafortune, S., Kelly, T., Kudlur, M., and Mahlke, S. The Theory of Deadlock Avoidance via Discrete Control. In *Conf. POPL* (2009).