Towards Discrete Control for the Internet of Things and Smart Environments

Mengxuan Zhao, Gilles Privat, Orange Labs, Grenoble, France
Eric Rutten, INRIA, Grenoble, France
Hassane Alla GIPSA Lab, Grenoble, France

June 25, 2013

Outline

Outline

Motivation

IoT and smart environments

- massive instrumentation, networked sensors and actuators
- outside-in robots, sense and act upon their own inner space
- smart homes, smart buildings or smart cities

Control techniques for autonomic management

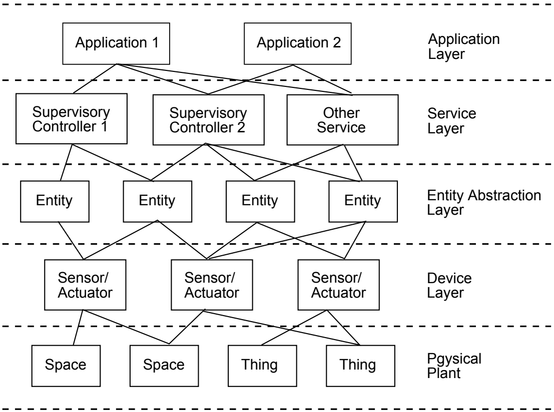
- model possible behaviours, and control objectives, separately
- classically continuous time dynamics and differential equations
- discrete control, events and states, Petri nets or automata

Discrete control for the IoT

- systematic modelling framework ; case study in smart home
- automata ; Discrete Controller Synthesis (DCS)

Outline

Interfacing to the IoT and Smart Environments



monitoring and controlling of entities (rooms, appliances, ...)

- intermediate abstraction layer HAL (Home Abstraction Layer)
- supervisory controllers as service : emphasis on genericity

Reactive languages, synchronous programming

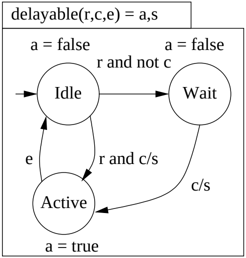
Modelling formalism **and** programming language

- reaction to input flows $\longrightarrow$ output flows
- data-flow nodes and equations ; mode automata (FSM)
- parallel (synchronous) and hierarchical composition

synchronous languages, (25+ years)

tools: compilers (e.g., Heptagon), code generation, verification, ...

example: delayable task control (in Heptagon)




```
node delayable(r,c,e:bool) returns (a,s:bool)
let automaton
state Idle do
   a = false; s = r and c
  until r and c then Active
      | r and not c then Wait
state Wait do a = false; s = c
  until c then Active
state Active do a = true; s=false
  until e then Idle
end tel
```

Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system
on which Φ does not yet hold a priori

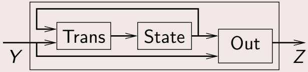
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system on which Φ does not yet hold a priori

Principle (on implicit equational representation)

State memory
Trans transition function
Out output function



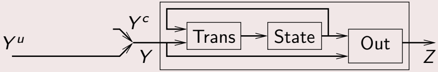
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system
on which Φ does not yet hold a priori

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function



- Partition of variables : controllable (Y^c), uncontrollable (Y^u)

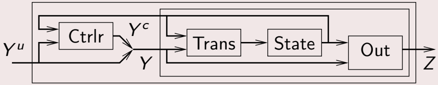
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system
on which Φ does not yet hold a priori

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function

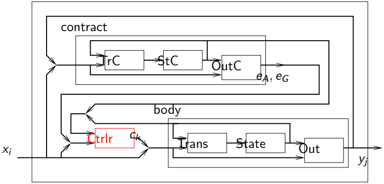


- Partition of variables : controllable (Y^c), uncontrollable (Y^u)
- Computation of a controller such that the controlled system satisfies Φ (invariance, reachability, attractivity, ...)

DCS tool: Sigali (H. Marchand e.a.)

BZR programming language [http://bzr.inria.fr]

- built on top of nodes in Heptagon
- to each contract, associate controllable variables, local
- at compile-time (user-friendly DCS),
  compute a controller for each component
- when no controllable inputs : verification by model-checking
- *step* and *reset* functions ; executable code : C, Java, ...



$\texttt{twotasks}(r_1, e_1, r_2, e_2) = a_1, s_1, a_2, s_2$

enforce not $(a_1$ **and** $a_2)$

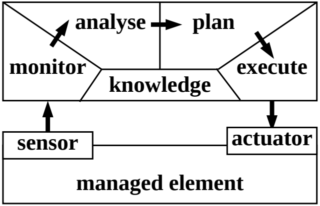
with c_1, c_2

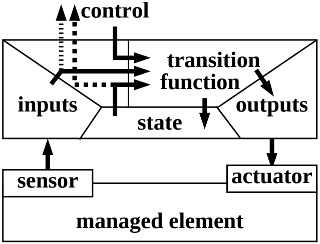
$(a_1, s_1) = \texttt{delayable}(r_1, c_1, e_1);$

$(a_2, s_2) = \texttt{delayable}(r_2, c_2, e_2)$

& G. Delaval & H. Marchand [ACM LCTES'10] [jDEDS13]

Discrete control as MAPE-K

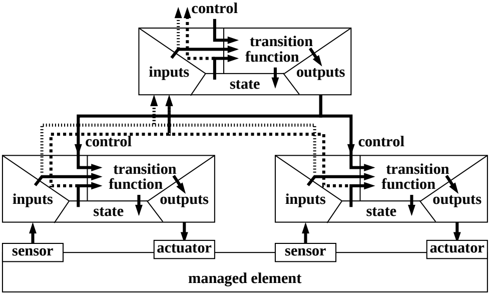




- autonomic MAPE-K
- flows : sensor observations to reconfiguration actions
- reactive language BZR used as DSL for decision

- FSM instanciation of MAPE-K
- exhibit state (observability)
- accept events or conditions (controllability)

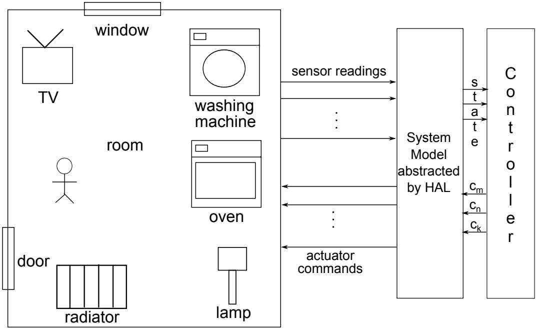
Hierarchical architecture



- hierarchical MAPE-K, through additional interfaces
- in components : composites using life-cycles of subcomponents
- implementation : *step*
  - synthesized and generated off-line
  - called at run-time in composite controller

Outline

Target environments



Smart home example

- set of entities, observation and control possibilities
- issues of safety (source of light on in case of presence),
- issues of economy or comfort (instantaneous power peaks)

Reconfigurable entities

Basic entities in the apartment

- door, window
- lamp, TV, radiator (off, frost-free, eco or high mode)
- oven : off, heating up, maintaining its current temperature
- washing machine : phases ; can be suspended

Sensors and actuators

- presence, door/window (open or closed)
- smart plugs for TV or lamp, suspend the wash machine

Reconfiguration policy

4 categories of objectives

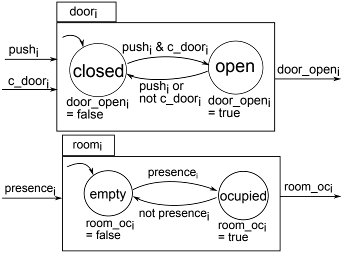
- safety (sa)
- security (se)
- energy efficiency (e)
- comfort (c)

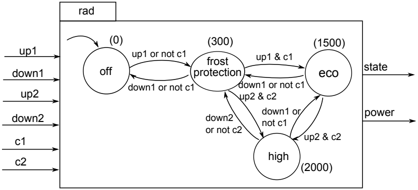
System objectives

1. (sa) at least one light source on when room is occupied
2. (se) close window and door when room isn't occupied
3. (e) if window or door open, radiator either off or frost-free
4. (e) if room inoccupied, no light on and radiator off or frost-free
5. (sa,e) total power under current threshold (3 modes: minimal-safety, comfort, eco)

Outline

System behaviors model (i)





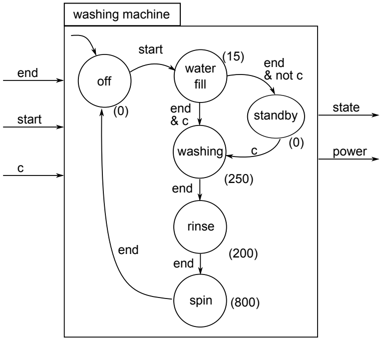
door and room

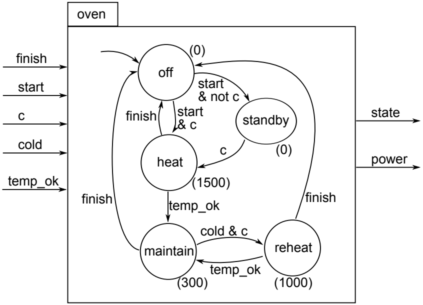
- two states / configurations
- controllability
  - door : prevent opening, close
  - room : observer only

radiator / heater

- four states / configurations
- controllability
  - prevent going higher
  - force going lower

System behaviors model (ii)





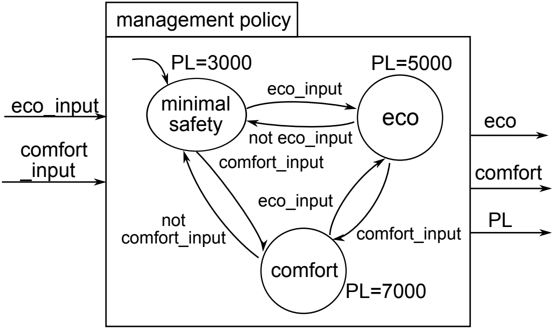
washing machine

- phases, with power
- standby controllable

oven

- 5 states, with power
- standby, reheat controllable

System behaviors model (iii)



Global system behavior model

- parallel composition of instances of automata
- global power consumption functions, in terms of the local ones $totalPower = p(wm) + p(ov) + p(rd)$
- 3 management policies : 3 states with different PL

Control objective specification

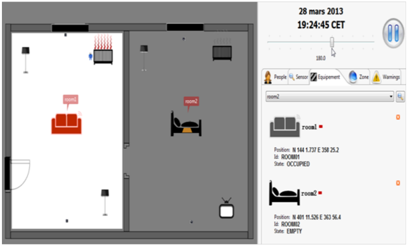
Conjunction of five rules

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

made invariant by control ; controller synthesized
tool-supported : BZR

Outline

Implementation and simulation



BZR encoding and DCS

- textual syntax for automata and contracts
- executable code generation (C or Java)

Simulation

- MiLeSEnS (**M**ulti-**Le**vel **S**mart **En**vironment **S**imulator)
- sensor and actuator models

Outline

Conclusions

DCS for IoT

first results in using discrete supervisory controllers applied to the IoT and smart homes and buildings

Systematic modeling framework

- behavioral modeling of typical entities
- formulation of control objectives of typical categories (safety, security, energy, comfort)
- automatic generation of controllers
- development and experimental validation

Perspectives

Perspectives

- Domain-Specific Language for smart environments
  generation of LTS and objectives
- more DCS : modular, quantitative ; identification of models
- more complete experiments

Part of a general approach using discrete control for computing

- behavioral modeling using LTSs : high level
- automatic generation (DCS) : safe, max.permissive
- tool support (BZR language)
- used also for :
  - coordinating administration loops in the Cloud or HPC
  - managing reconfigurable architectures (FPGA)