Discrete control-based design of adaptive and autonomic computing systems

Xin An [1] Gwenaël Delaval [2] Jean-Philippe Diguet [3]
Abdoulaye Gamatié [4] Soguy Gueye [2] Hervé Marchand [5]
Noël de Palma [2] **Eric Rutten** [6]

1: Hefei University of Technology, Hefei, China, xin.an@hfut.edu.cn

2: LIG, Grenoble, France, gwenael.delaval@imag.fr, Soguy-Mak-Kare.Gueye@imag.fr, noel.depalma@imag.fr

3: Lab-STICC, Lorient, France, jean-philippe.diguet@univ-ubs.fr

4: LIRMM, Montpellier, France, abdoulaye.gamatie@lirmm.fr

5: INRIA, Rennes, France, herve.marchand@inria.fr

6: **INRIA, Grenoble, France**, eric.rutten@inria.fr, https://team.inria.fr/ctrl-a/members/eric-rutten

January 12, 2015

Outline

Outline

Adaptive computing systems

two complementary, and sometimes contradictory, requirements:

- **adaptability** to changes in their environment or functionality
- **dependability** w.r.t. their goal and persons in contact

administration loops in computing systems

reacting to changes in:

- operation environment
- implementation platform
- application objectives

automated

too large or complex
sometimes too fast

for manual administration

⟶ **Autonomic Computing** : self-managed systems
automated administration in the form of a feedback loop

Control for dependability

- w.r.t. damage in system finality (information, business, ...)
- w.r.t. safety (goods, persons, ...)

specificity of autonomic systems : automated feedback loop

need for control of automated behaviors

- they can oscillate, diverge, react too slowly, ...
- objectives can be multiple and interfere

⟶ design theories and techniques from **Control Theory**

new interaction between control and computer science

- computer science for control systems : embedded systems
- theoretical informatics and control theory : hybrid systems

→ **control theory for computing systems** considered here
for well-behaved automated computer management loops

Outline

Autonomic computing

Autonomic Computing Initiative (ACI) initiated by IBM, early 2000

networked computing systems able to manage themselves, trough decisions made automatically, without direct human intervention

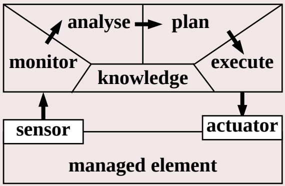
autonomic objectives

- Self-configuration
- Self-healing
- Self-optimization
- Self-protection

can interact, interferences can require coordination

autonomic loop

MAPE-K



Control for feedback computing

for guarantees on behavior of automated closed-looped systems

control theory: framework of methods and techniques

to build automated systems with well-mastered behavior
sensors and actuators connected to given "plant" to be controlled
model of the dynamic behavior of the process,
control objective specified explicitly
$\longrightarrow$ on these bases the control solution is formally derived

Control for computing systems: Feedback Computing

not usual in Computer Science, still only emerging
advantages: rigorous, supports uncertainties, stability, robustness,
difficulties: modeling and translating management objectives
to actual system-level sensors and actuators,
to appropriate, useable control models
most computing systems not designed to be controllable

Reactive languages, synchronous programming

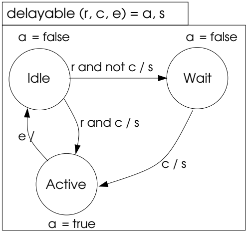
Modelling formalism **and** programming language

- reaction to input flows → 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

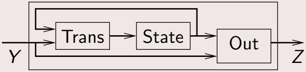
Discrete controller synthesis (DCS): principle

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



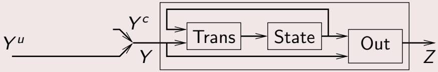
Discrete controller synthesis (DCS): principle

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)

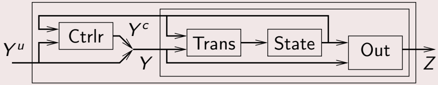
Discrete controller synthesis (DCS): principle

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



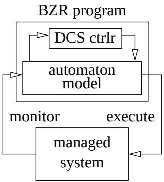
- Partition of variables : controllable (Y^c), uncontrollable (Y^u)
- Computation of a controller such that the controlled system satisfies Φ by control (invariance, reachability, attractivity, ...)

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

Outline

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, ...




```
node delay
  (new_sig: bool; c:bool)
returns (out: bool)
let automaton
  state Idle
   do out=new_sig & c
   until new_sig & not c
                then Waiting
    | new_sig & c then Idle
  state Waiting
   do out=c
   until c then Idle
end tel
```

```
node main
  (signal1, signal2: bool)
returns (d1, d2:bool)
contract
 enforce not (d1 & d2)
 with (c1,c2:bool)
let
 d1 = delay(signal1, c1);
 d2 = delay(signal2, c2);
tel
```

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

Need for modularity

Advantages of DCS approach

- (i) high-level language support
- (ii) correctness of the controller,
- (iii) maximal permissiveness of controllers
- (iv) automated formal synthesis of these controllers
- (v) automated executable code generation in C or Java.

Need for modularity

- scalability: state-space exploration algorithms are exponential
- re-usability of management components

Modularity in BZR

node(...) = ...
assume A **enforce** G
with $c_1, \dots c_q$

subnode$_1$(...) = ...
assume A_1 **enforce** G_1

; . . . ;

subnode$_n$(...) = ...
assume A_n **enforce** G_n

Modular contracts in Heptagon/BZR

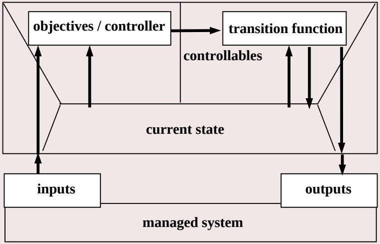
based on the modular compilation of the nodes

- **assume** not only A, but also that the n sub-nodes each do enforce their contract: $\bigwedge_{i=1}^{n}(A_i \implies G_i)$.
- **enforce** G as well as the assumptions of sub-nodes: $\bigwedge_{i=1}^{n} A_i$

Outline

General design method

An interpretation of the MAPE-K loop



Typical modeled features

observability & controllability

resources: levels, on/off,

tasks: activity, start, end, checkpoints, modes

application: task graph, workflow

Granularity levels, depending on decision problem

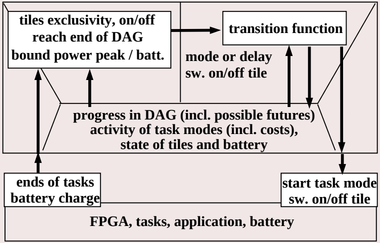
lowest : **MEs** : (relatively) fast, low overhead

level of AM : slower pace, sporadic ; limited by dynamics of system

level of AMs coordination : even slower, can afford synchronizations, distributed decisions e.g. leader election

Reconfiguration control in DPR FPGA-based architectures

Considered class of architectures — ANR Famous

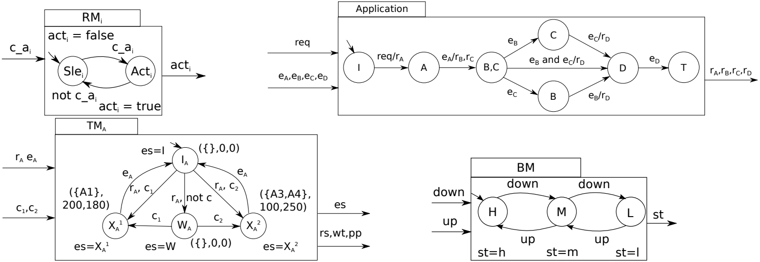


- **architecture** : tiles $A_{1..n}$, (sleep mode) ; battery
- **tasks** : delayable ; modes (tiles, power, WCET, ...)
- **application** : task graph

reconfiguration policy

1. resource usage : exclusive tiles $A1$-$A4$
2. energy : tiles active if and only if needed
3. power peak : bounded w.r.t battery level
4. reachability: application graph end
5. optimizing e.g., global power peak

Modelling for in DPR FPGA control [ICAC13]



Generic models

tiles RM_i, task graph, (two-modes) tasks, battery

global model : composition of instances

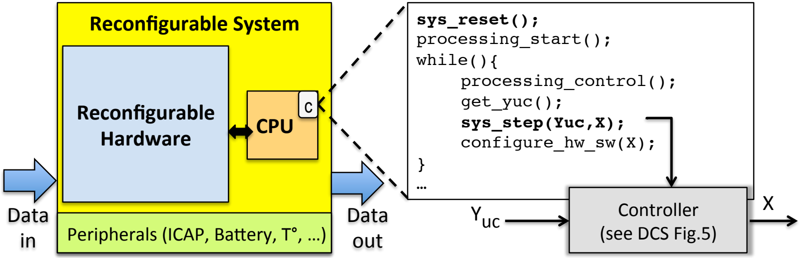
reconfiguration policy

Objective 1 to 3 : *invariance* e.g., for 3 :

$$PP < (v_1 \text{ if } st = h \text{ else } v_2 \text{ if } st = m \text{ else } v_3 \text{ if } st = l)$$

Objective 4 : *reachability* of terminal state T

Implementation of DPR FPGA control

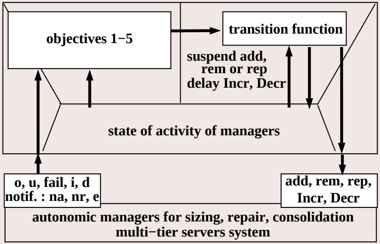


calling executable code generated by BZR

- call **reset** function to initialize sates
- loop for cyclic reaction :
  - acquire sensor input
  - construct automaton input Yuc
  - call **step** function to make transition and decisions
  - transform automaton ouput X into calls to OS API (start, ...)

Coordination of administration loops

Administration loops and their coordination — ANR Ctrl-Green

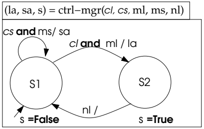


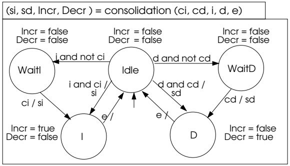
- **multi-tier applications** : replicated web servers ; load balancers
- **autonomic managers** : Self-sizing ; Self-repair ; Consolidation
- **problems** : over-reaction

reconfiguration policy

1. In a replicated tier, avoid size-up when repairing
2. avoid size-down in successor tier when repairing predecessor
3. when consolidating, avoid self-sizing or repairing

Modelling for coordination control [jFGCS14]





Generic models

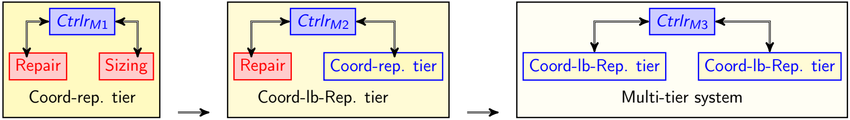
Self-sizing control
Self-repair control
Consolidation control

models instantiated for each AM
composition gives global behavior

reconfiguration policy

1. not (repairing and add)
2. not (repairingL and rem)
3. not (repairing$_{pred}$ and rem$_{succ}$)
4. ...

Modular coordination control [CBSE14]



Bottom-up re-use of nodes

- **replicated servers** tier: `Coord-rep. tier`
  coordinating one Repair and one Sizing
- **load-balanced** tier: `Coord-lb-Rep. tier`
  coordinating one Repair (for LB) and one of the former
- **application**: `Multi-tier system`
  coordinating two of the former

Outline

Conclusions & perspectives

Results

overview on discrete control-based design of autonomic computing

- **tool-supported** method, reactive language & discrete control
- **validation** in domains from software components and smart environments to hardware reconfigurable architectures.
- control-based techniques offer, at the same time, **self-adaptation and predictability**

Perspectives

- **Modeling** : other aspects of computing systems (memory, ...)
- **Expressivity and scalability** : logico-numeric
- **High-level languages** : Domain Specific Languages (DSLs)
- **Adaptive discrete control** : not much theory yet

Outline

Scientific landscape: workshops and conferences

- section spéciale revue TSI
- Staars Grenoble mai & nov. 14 https://persyval-lab.org/en/exploratory-project/staars
- LAAS Toulouse oct. 14 http://projects.laas.fr/autonomique
- sessions WODES 14 , DCDS15(?)
  http://wodes2014.lurpa.ens-cachan.fr/wp-content/uploads/2013/10/SPE_3_E_Rutten.pdf
- ICAC 15 juil., Grenoble http://icac2015.imag.fr
- CAC 15 sept.15 Boston http://autonomic-conference.org
- Dagstuhl: Software Engineering for Self-Adaptive Systems: Assurances, dec. 2013 http://www.dagstuhl.de/13511

  Control Theory meets Software Engineering, sept. 2014 http://www.dagstuhl.de/14382
- Lund (LCCC): Control of Computing Systems, dec. 2011 http://www.lccc.lth.se/index.php?page=workshop-control-computing

  Cloud Control, may 2014 http://www.lccc.lth.se/index.php?page=may-2014-2

Scientific landscape: working groups, in France

- GdRs du CNRS
  - GPL GT COSMAL http://gdr-gpl.cnrs.fr/Groupes/COSMAL
  - GPL et Robotique: conférences CAR
    http://www.lirmm.fr/gtcar/index.php/autres-workshop/workshop-cps-gdr-gpl-robotique
  - ASR: action ADAPT, nouveau groupe en cours,
    2ème Atelier Automatique pour l'Informatique Autonomique Compas 13
    http://compas2013.inrialpes.fr/cfp/ateliers.html#automatique
    Atelier Informatique Autonomique Compas 14 http://compas2014.unine.ch/atelier2
  - SoCSiP: GT Architectures reconfigurables
  - MACS: GT automatique et réseaux,
    nouveau GT commande de systèmes informatiques?
  - autres GdRs?
    IA (Intelligence artificielle),
    ISIS (traitement du signal, image, ...),
    ...
- autres groupes?

$\longrightarrow$ co-existence et coordination