Modular Coordination of multiple autonomic managers
using Modular Discrete Controller Synthesis

Soguy Mak-karé Gueye[1] Gwenaël Delaval[1]
Noël de Palma[1] Eric Rutten[2]

[1]Laboratoire Informatique de Grenoble (LIG)

[2]INRIA Grenoble - Rhône-Alpes

Ctrl-a / STAARS, January 13, 2014

Outline

1. Need for coordination in Self-managing systems
   - Autonomic Management of large-scale systems
   - Autonomic manager
2. Approach
   - Synchronous programming
   - Discrete controller synthesis
   - BZR programming language
3. Modular coordination of autonomic managers
   - Need for modular design of the coordination control
   - Modular Coordination: principle
4. Application
   - Multi-tier systems in a Datacenter
     - Self-sizing, self-repair and server consolidation manager
     - Coordination Problem
     - Coordination policy
   - Modelling the managers behaviours
   - Coordinating the managers

Outline

Autonomic Management of large-scale systems

Difficult to handle manually

- Complexity in size
- Degree of heterogeneity
- Distributed architecture

Autonomic Computing

Self-managing System

- Manages itself autonomously
- Aware of its context
- React and adapt to changes
- functional areas

Self-Configuration, Self-Healing, Self-Optimization, Self-Protection ...

Autonomic manager

Software (or Hardware) component

- Implement management decisions
- Behaviours
  - Monitor system execution status
    - cpu load, response time, etc.
  - Analyse of the measures
  - Plan & execute reconfigurations

Ensuring global system management

- Multiple autonomic managers
  - achieve overall objectives
- Preventing
  - conflicting decisions, redundant operations

Coordination to get coherent and efficient management

$\Rightarrow$ *Problem of synchronization and logical control AMs actions*

Outline

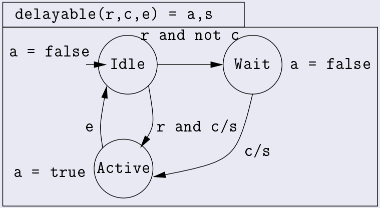
1. Need for coordination in Self-managing systems
2. Approach
   - Synchronous programming
   - Discrete controller synthesis
   - BZR programming language
3. Modular coordination of autonomic managers
4. Application

Modelling formalism and programming language

- data-flow nodes / equations : input flows → output flows
- mode automata (FSM)
- parallel and hierarchical composition

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

example: computing task control, delayable




```
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

Purpose

Constrain system behaviours to satisfy a property Φ

compute a controller

Discrete controller synthesis

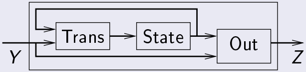
Purpose

Constrain system behaviours to satisfy a property Φ

compute a controller

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function



Discrete controller synthesis

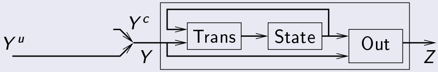
Purpose

Constrain system behaviours to satisfy a property Φ

compute a *controller*

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

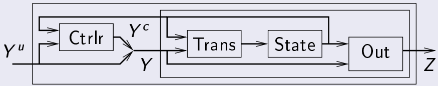
Purpose

Constrain system behaviours to satisfy a property Φ

compute a controller

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, ...)

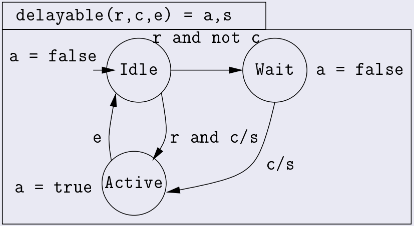
DCS tool: Sigali (H. Marchand e.a.)

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

Built on top of nodes in Heptagon

- to each contract, associate controllable variables, local to the component
- at compile-time (user-friendly DCS),
  Compute a controller for each component
- When no controllable inputs : verification by model-checking

Example (cont'd)



twotasks$(r_1, e_1, r_2, e_2) = a_1, s_1, a_2, s_2$

assume true

enforce not $(a_1$ and $a_2)$

with c_1, c_2

$(a_1, s_1) =$ delayable(r_1, c_1, e_1);

$(a_2, s_2) =$ delayable(r_2, c_2, e_2)

Modular DCS with Heptagon/BZR

Use of contracts of each subnode

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$
`assume` e_A
`enforce` e_G
`with` $c_1, \ldots, c_q$

$f_1(x_{1\,1}, \ldots, x_{1\,n}, c_1, \ldots, c_q) = (y_{1\,1}, \ldots, y_{1\,p})$
`assume` $e_{A\,1}$
`enforce` $e_{G\,1}$

…

$f_p(x_{p\,1}, \ldots, x_{p\,n}, c_1, \ldots, c_q) = (y_{p\,1}, \ldots, y_{p\,p})$
`assume` $e_{A\,p}$
`enforce` $e_{G\,p}$

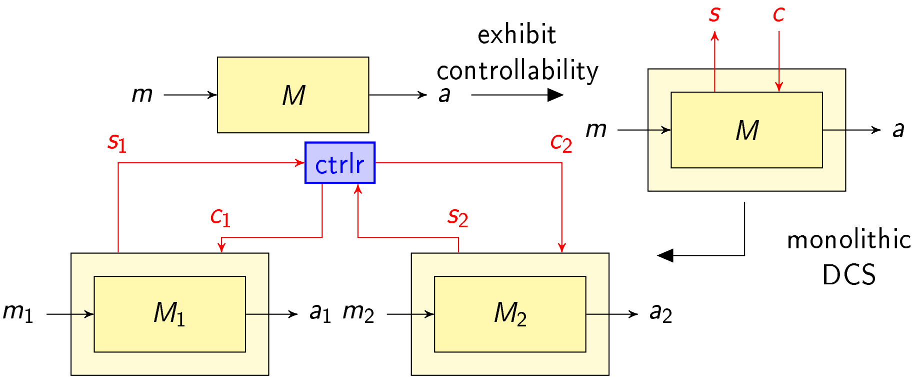
Synthesis objective:

$$\forall\Box\Big(\big(e_{A1} \Rightarrow e_{G1}\big) \wedge \ldots \wedge \big(e_{Ap} \Rightarrow e_{Gp}\big) \wedge e_A\Big) \Rightarrow \Big(e_G \wedge e_{A1} \wedge \ldots \wedge e_{Ap}\Big)$$

- Assuming e_A and each *subnode$_i$* enforces $(e_{Ai} \Rightarrow e_{Gi})$
    - Compute controller enforcing e_G and *each* e_{Ai}
    - Contracts of subnodes abstract their body
- Modular code generation

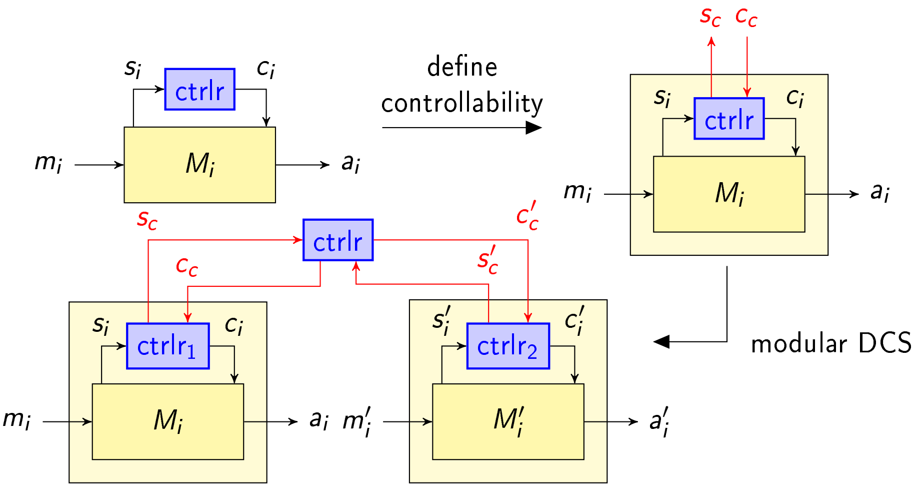
Outline

Monolithic Coordination: principle



- Scalability ?
  - State-space explosion problem
- Re-usability ?
  - Contracts in the main node – recompilation when modification

Modular Coordination: principle



- Scalable: contracts decomposition – break down complexity
- Re-usable: not recompiled

Outline

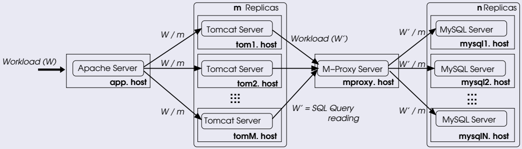
1. Need for coordination in Self-managing systems
2. Approach
3. Modular coordination of autonomic managers
4. Application
   - Multi-tier systems in a Datacenter
     - Self-sizing, self-repair and server consolidation manager
     - Coordination Problem
     - Coordination policy
   - Modelling the managers behaviours
   - Coordinating the managers

Multi-tier systems in a Datacenter

Datacenter

- Provide virtualized execution platform
  - Supported by several physical servers
  - Shared among multiple applications (*e.g.*, *Multi-tier JEE App*)
  - Virtual machines host application servers

Multi-tier JEE application



- Inter-connected tiers
  - Apache dispatches incoming requests to Tomcats
  - Tomcats access Database by Mysql-proxy
  - Mysql-proxy dispatches SQL queries reading to MySQLs
- Replicated tiers
  - Tomcats and MySQLs

Autonomic managers

Self-Sizing – Management of the Degree of replication

- Optimize resources usage and improve performance
  - Monitors cpu load of servers Hosts
  - Sizes up when overload
  - Sizes down when underload

Self-Repair – Management of service availability

- Improve service availability
  - Monitors servers hosts HeartBeat
  - Repairs when failure (fail-stop)

Server Consolidation –

- Preserve efficient usage of servers and applications performance
  - Resize the number of active servers based on the workload

Coordination Problem

Server failure leading to Workload misinterpretation

- Failure in a replicated tier ⇒ temporary overload
  - Tomcats and MySQLs
- Failure in a tier ⇒ temporary underload in its replicated successors
  - Apache ⇒ Tomcats and MySQLs
  - Tomcat ⇒ MySQLs
  - Mysql-Proxy ⇒ and MySQLs

Conflicting management actions

- Consolidation decrease execution ⇒ size-up (repair) failure
  - no enough resource
- Size-down ⇒ invalidate consolidation increase plan
  - additional resource not needed
- Size-up (repair) ⇒ invalidate consolidation decrease plan
  - resource requested

Coordination policy

Multi-tier level

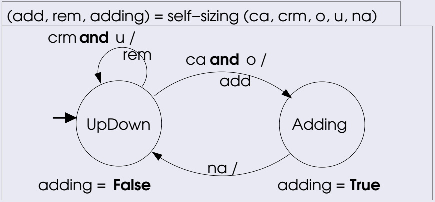
1. Within a replicated tier, avoid size-up when repairing.
2. Within a load-balanced replicated tier, avoid size-down when repairing the load-balancer.
3. In multi-tiers, more generally, avoid size-down in a successor replicated tier when repairing in a predecessor.

Datacenter level

1. When consolidating, avoid sizing or repairing.
2. When repairing or sizing up, delay consolidation decreasing
3. When sizing down, delay consolidation increasing

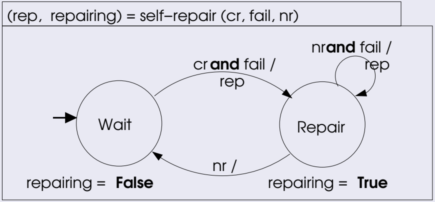
Modular control model : managers

Self-sizing control



- States:
  - UpDown: awaits
  - Adding: adding
- inputs:
  - o: overload
  - u: underload
  - na: adding completed
- actions:
  - add: triggers adding
  - rem: triggers removal
- controllables:
  - ca: control adding
  - crm: control removal

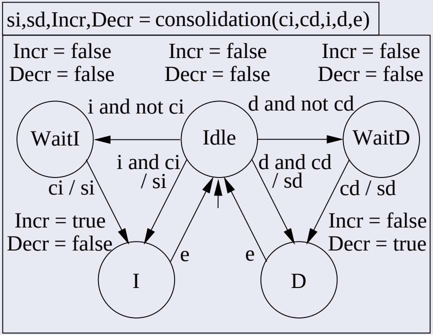
Self-repair control



- States:
  - Wait: awaits failure
  - Repair: repairing
- inputs:
  - fail: failure
  - nr: repair completed
- actions:
  - rep: triggers repair
- controllables:
  - cr: control repair

Modular control model : managers

Consolidation control



- States:
  - Idle: awaits
  - WaitI: awaits authorisation to increase
  - I: increasing (Incr)
  - WaitD: awaits authorisation to decrease
  - D: decreasing (Decr)
- inputs:
  - i: increase notification
  - d: decrease notification
  - e: completion notification
- actions:
  - si: triggers increase
  - sd: triggers decrease
- controllables:
  - ci: control increase
  - cd: control decrease

Coordination objectives

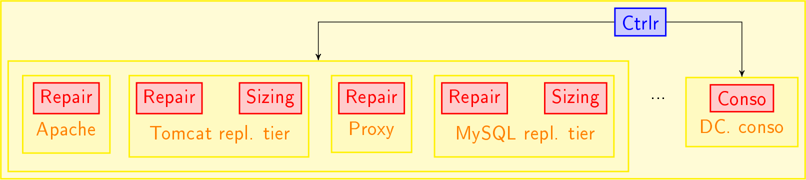
Multi-tier

1. Replicated tier: not (repairing and add)
2. Load-balanced replicated tier: not (repairingL and rem)
3. In multi-tiers: not $(\text{repairing}_{pred}$ and $\text{rem}_{succ})$

Datacenter

1. not ((Incr or Decr) and (repairing* or adding* or rem*))
2. not ((repairing* or adding*) and sd)
3. not (rem* and si)

Exploiting models with DCS : monolithically



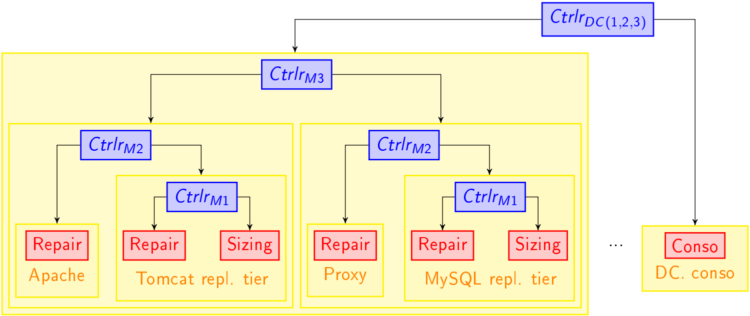
$(\dots) =$ `Main_node` $(\dots)$

`enforce` *all contracts*

`with` *all controllable variables*

```
(rep_1, repairing_1)   = self-repair (c'_1, fail_1, nr_1);
...
(rep_N, repairing_N)   = self-repair (c'_N, fail_N, nr_N);
(add_1, rem_1, adding_1) = self-sizing (ca_1, ...);
...
(add_M, rem_M, adding_M) = self-sizing (ca_M, ...);
(si, sd, Incr, Decr) = consolidation (ci, cd, i, d, e);
```

Modular DCS: Modelling architecture



Beside local control objectives

- Enforce outside control objectives
  - control triggering of actions: $(\neg c_i' \Rightarrow \neg a_i)$
    - short action: $(c_i'$ `or not` $a_i)$
    - long action: $longActCtrl(c_i', a_i, s_i) \stackrel{\text{def}}{=}$
      $((c_i'$ `or not` $a_i)$ `and` $(($`not` $($`false fby` $s_i)$ `and not` $a_i) \Rightarrow$ `not` $s_i)$

Exploiting models with DCS : modularly

Replicated tier node

```
(...) = coord_repl-tier (cr', fail, nr, ca', crm', o, u, na)
enforce ((not (repairing and add))
          and longActCtrl(cr', rep, repairing))
          and longActCtrl(ca', add, adding)
          and (crm' or not rem))
with cr, ca, crm
   (rep, repairing)    = self-repair (cr, fail, nr);
   (add, remove, adding) = self-sizing (ca, crm, o, u, na);
```

Replicated tier

1. Control of a self-sizing and a self-repair
2. Enforcement
   1. local control: not (repairing and add)
   2. outside control: cr', ca' and crm' with associated objectives

Exploiting models with DCS : modularly (ii)

Load-balanced replicated tier node

```
(...) = coord_lb-repl-tier (cL', failL, nrL, c', fail, nr, ca', crm', o, u, na)
enforce (not (repairingL and rem))
        and longActCtrl(cL', repL, repairingL))
        and longActCtrl(c', rep, repairing))
        and longActCtrl(ca', add, adding)
        and (crm' or not rem))
with cL, c, ca, crm

  (repL, repairingL)  = self-repair (cL, failL, nrL);
  (rep, repairing, add, rem, adding) = coord_repl-tier (c, fail, nr, ca,
                                                        crm, o, u, na);
```

Load balanced replicated tier

1. Control of a self-repair and a coord_repl-tier
2. Enforcement
   1. local control: not (repairingL and rem)
   2. outside control: cL', c', ca' and crm' with associated objectives

Exploiting models with DCS : modularly (iii)

Multi-tier application node

$$(\ldots) = \texttt{coord\_appli}\ (\texttt{cL}'_1,\ \texttt{failL}_1,\ \texttt{nrL}_1,\ \texttt{c}'_1,\ \texttt{fail}_1,\ \texttt{nr}_1,\ \texttt{ca}'_1,\ \texttt{crm}'_1,\ \texttt{o}_1,\ \texttt{u}_1,\ \texttt{na}_1,\ \texttt{cL}'_2,\ \texttt{failL}_2,\ \texttt{nrL}_2,\ \texttt{c}'_2,\ \texttt{fail}_2,\ \texttt{nr}_2,\ \texttt{ca}'_2,\ \texttt{crm}'_2,\ \texttt{o}_2,\ \texttt{u}_2,\ \texttt{na}_2)$$

$$\begin{aligned}\texttt{enforce}\ ((&\texttt{not}\ ((\texttt{repairingL}_1\ \texttt{or}\ \texttt{repairing}_1)\ \texttt{and}\ \texttt{rem}_2)))\\ &\texttt{and}\ \mathit{longActCtrl}(\texttt{cL}'_i, \texttt{repL}_i, \texttt{repairingL}_i))\\ &\texttt{and}\ \mathit{longActCtrl}(\texttt{c}'_i, \texttt{rep}_i, \texttt{repairing}_i))\\ &\texttt{and}\ \mathit{longActCtrl}(\texttt{ca}'_i, \texttt{add}_i, \texttt{adding}_i)\ \texttt{and}\ (\texttt{crm}'_i\ \texttt{or}\ \texttt{not}\ \texttt{rem}_i))\\ &\quad i = 1, 2\end{aligned}$$

$\texttt{with}\ cL_1,\ c_1,\ ca_1,\ crm_1,\ cL_2,\ c_2,\ ca_2,\ crm_2$

$$(\texttt{repL}_1,\ \texttt{repairingL}_1,\ \texttt{rep}_1,\ \texttt{repairing}_1,\ \texttt{add}_1,\ \texttt{rem}_1,\ \texttt{adding}_1) = \texttt{coord\_lb-repl-tier}\ (cL_1,\ \texttt{failL}_1,\ \texttt{nrL}_1, c_1,\ \texttt{fail}_1,\ \texttt{nr}_1,\ ca_1,\ crm_1,\ \texttt{o}_1,\ \texttt{u}_1,\ \texttt{na}_1);$$

$$(\texttt{repL}_2,\ \texttt{repairingL}_2,\ \texttt{rep}_2,\ \texttt{repairing}_2,\ \texttt{add}_2,\ \texttt{rem}_2,\ \texttt{adding}_2) = \texttt{coord\_lb-repl-tier}\ (cL_2,\ \texttt{failL}_2,\ \texttt{nrL}_2, c_2,\ \texttt{fail}_2,\ \texttt{nr}_2,\ ca_2,\ crm_2,\ \texttt{o}_2,\ \texttt{u}_2,\ \texttt{na}_2);$$

Multitier

1. Control of two coord_Lb-repl-tier
2. Enforcement
   1. local control: `not (repairingL and rem)`
   2. outside control: cL'_i, c'_i, ca'_i and crm'_i with associated objectives

Exploiting models with DCS : modularly (iv)

Two-application data-center

```
(...) = two-data-center (...)
enforce (  (not ((Incr or Decr) and (repairing_ij or adding_ij or rem_ij)))
            and (not ((repairing_ij or adding_ij) and sd)
           and longActCtrl(cL'_ij, repL_ij, repairingL_ij))
           and longActCtrl(c'_ij, rep_ij, repairing_ij))
           and longActCtrl(ca'_ij, add_ij, adding_ij) and (crm'_ij or not rem_ij))
      i = 1,2; j = 1,2
with cL11, c11, ..., crm22, ci, cd

  (...) = coord_appli (cL11, c11, ca11, crm11, ..., cL21, c21, ca21, crm21, ...)
  (...) = coord_appli (cL12, c12, ca12, crm12, ..., cL22, c22, ca22, crm22, ...)
  (si, sd, Incr, Decr) = consolidation (ci, cd, i, d, e);
```

Two application & consolidation

1. Control of two coord-appli and a consolidation
2. Enforcement
   1. local control: not (repairingL and rem)
   2. outside control: cL'_{ij}, c'_{ij}, ca'_{ij} and crm'_{ij} with associated objectives

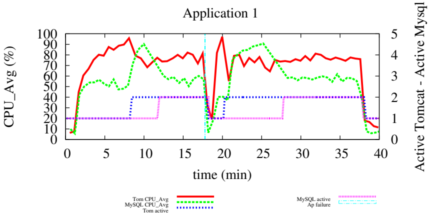
Exploiting models with DCS : comparisons

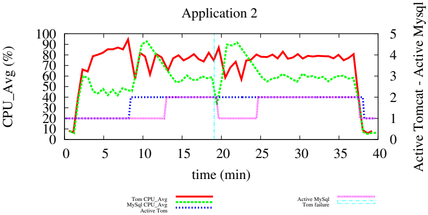
monolithic vs modular

nb. app.	Synthesis time		Memory usage	
	monolithic	modular	monolithic	modular
1	0s	5s	-	-
2	49s	11s	-	-
3	42m24s	24s	34.81MB	-
4	$>$ 2 days	1m22s	>149,56MB	-
5	-	4m30s	-	20,37MB
6	-	13m24s	-	53,31MB
7	-	25m57s	-	77,50MB
8	-	50m36s	-	115,59MB
9	-	2h11m	-	236,59MB
10	-	9h4m	-	479,15MB

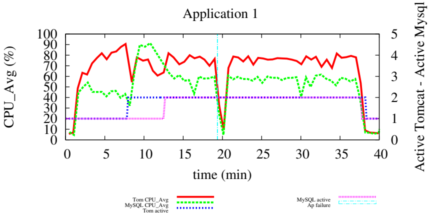
Implementation and Evaluation

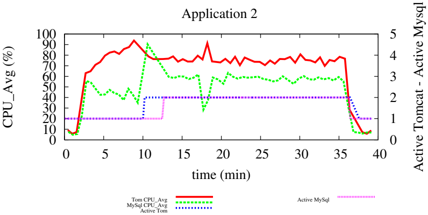
Uncoordinated execution





Coordinated execution





Conclusion

- major challenge : consistent, efficient and flexible coexistence between autonomic managers in the same system

Conclusion

- major challenge : consistent, efficient and flexible coexistence between autonomic managers in the same system
- approach: synchronous programming and DCS
  - automatic generation of the controller for cooperation of multiple autonomic managers from high-level policy,
  - correctness by construction of the generated controller

Conclusion

- major challenge : consistent, efficient and flexible coexistence between autonomic managers in the same system
- approach: synchronous programming and DCS
  - automatic generation of the controller for cooperation of multiple autonomic managers from high-level policy,
  - correctness by construction of the generated controller
- perspectives
  - large scale coordination : several managers and multi-tier architectures
  - control beyond mutual exclusion : sequential aspects