Modular Coordination of multiple autonomic managers
using Modular Discrete Controller Synthesis

Soguy Mak-karé Gueye\textsuperscript{1}  Gwenaël Delaval\textsuperscript{1}
Noël de Palma\textsuperscript{1}  Eric Rutten\textsuperscript{2}

\textsuperscript{1}Laboratoire Informatique de Grenoble (LIG)
\textsuperscript{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

1 Need for coordination in Self-managing systems
   - Autonomic Management of large-scale systems
   - Autonomic manager

2 Approach

3 Modular coordination of autonomic managers

4 Application
**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

⇒ 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
Synchronous programming

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
```

```
node delayable(r,c,e) = a,s

a = false
Idle
r and not c
Wait
a = false
r and c/s
e
Active
c/s
a = true
```

Discrete controller synthesis

Purpose

Constrain system behaviours to satisfy a property $\Phi$

compute a controller
Discrete controller synthesis

**Purpose**

Constrain system behaviours to satisfy a property $\Phi$

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 $\Phi$ (invariance, reachability, attractivity, ...)

DCS tool: Sigali (H. Marchande et al.)
Discrete controller synthesis

**Purpose**

Constrain system behaviours to satisfy a property $\Phi$

*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 $\Phi$

*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 $\Phi$ (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)

delayable(r,c,e) = a,s

- a = false
  - Idle
  - Wait
  - a = false
- a = true
  - Active
  - c/s
  - r and not c
  - e
  - r and c/s

\[
\begin{align*}
twotasks(r_1,e_1,r_2,e_2) &= a_1,s_1,a_2,s_2 \\
assume \quad &\text{true} \\
enforce \quad &\text{not } (a_1 \text{ and } a_2) \\
with \quad &c_1,c_2 \\
\quad \quad (a_1,s_1) &= \text{delayable}(r_1,c_1,e_1); \\
\quad \quad (a_2,s_2) &= \text{delayable}(r_2,c_2,e_2)
\end{align*}
\]
Use of contracts of each subnode

\[
f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)
\]

\begin{align*}
\text{assume } e_A \\
\text{enforce } e_G
\end{align*}
with \(c_1, \ldots, c_q\)

\[
f_1(x_{11}, \ldots, x_{1n}, c_1, \ldots, c_q) = (y_{11}, \ldots, y_{1p})
\]

\begin{align*}
\text{assume } e_{A1} \\
\text{enforce } e_{G1}
\end{align*}

\[
\ldots
\]

\[
f_p(x_{p1}, \ldots, x_{pn}, c_1, \ldots, c_q) = (y_{p1}, \ldots, y_{pp})
\]

\begin{align*}
\text{assume } e_{Ap} \\
\text{enforce } e_{Gp}
\end{align*}

Synthesis objective:

\[
\forall \Box \left( (e_{A1} \Rightarrow e_{G1}) \land \ldots \land (e_{Ap} \Rightarrow e_{Gp}) \land e_A \right) \Rightarrow \left( e_G \land e_{A1} \land \ldots \land e_{Ap} \right)
\]

- Assuming \(e_A\) and each subnode \(e_{Ai}\) 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

1. Need for coordination in Self-managing systems

2. Approach

3. Modular coordination of autonomic managers
   - Need for modular design of the coordination control
   - Modular Coordination: principle

4. Application
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
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
**Self-sizing control**

\[(\text{add}, \text{rem}, \text{adding}) = \text{self-sizing} (\text{ca}, \text{crm}, o, u, na)\]

- **States:**
  - UpDown: awaits
  - Adding: adding
- **Actions:**
  - add: triggers adding
  - rem: triggers removal
- **Controllables:**
  - ca: control adding
  -crm: control removal

**Diagram:**

- **States:** UpDown, Adding
- **Inputs:** o: overload, u: underload, na: adding completed
- **Outputs:**
  - add: triggers adding
  - rem: triggers removal
- **Controllables:**
  - ca: control adding
  - crm: control removal

**Self-repair control**

\[(\text{rep}, \text{repairing}) = \text{self-repair} (\text{cr}, \text{fail}, \text{nr})\]

- **States:**
  - Wait: awaits failure
  - Repair: repairing
- **Actions:**
  - rep: triggers repair
- **Controllables:**
  - cr: control repair

**Diagram:**

- **States:** Wait, Repair
- **Inputs:** fail: failure, nr: repair completed
- **Outputs:**
  - rep: triggers repair
- **Controllables:**
  - cr: control repair
Modular control model: managers

Consolidation control

\[ \text{si, sd, Incr, Decr} = \text{consolidation}(\text{ci, cd, i, d, e}) \]

- **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 (repairing_{pred} and 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

\[
\begin{align*}
    \text{(...)} &= \text{Main\_node (...)} \\
    \text{enforce all contracts} \\
    \text{with all controllable variables} \\
    (\text{rep}_1, \text{repairing}_1) &= \text{self-repair} \left( c'_1, \text{fail}_1, \text{nr}_1 \right); \\
    \ldots \\
    (\text{rep}_N, \text{repairing}_N) &= \text{self-repair} \left( c'_N, \text{fail}_N, \text{nr}_N \right); \\
    (\text{add}_1, \text{rem}_1, \text{adding}_1) &= \text{self-sizing} \left( c\alpha_1, \ldots \right); \\
    \ldots \\
    (\text{add}_M, \text{rem}_M, \text{adding}_M) &= \text{self-sizing} \left( c\alpha_M, \ldots \right); \\
    (\text{si}, \text{sd}, \text{Incr}, \text{Decr}) &= \text{consolidation} \left( c\iota, c\delta, i, d, e \right);
\end{align*}
\]
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: \((c_i' \text{ or not } a_i)\)
    - long action: \(\text{longActCtrl}(c_i', a_i, s_i) \overset{\text{def}}{=} ((c_i' \text{ or not } a_i) \text{ and ((not (false fby } s_i) \text{ and not } a_i)) \Rightarrow \text{not } s_i)\)
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
Multi-tier application node

\[(...) = \text{coord\_appli}(cL'_{1}, \text{failL}_{1}, \text{nrL}_{1}, c'_{1}, \text{fail}_{1}, \text{nr}_{1}, ca'_{1}, \text{crm}'_{1}, o_{1}, u_{1}, na_{1},
\hspace{1em}cL'_{2}, \text{failL}_{2}, \text{nrL}_{2}, c'_{2}, \text{fail}_{2}, \text{nr}_{2}, ca'_{2}, \text{crm}'_{2}, o_{2}, u_{2}, na_{2})\]

enforce \((\text{not} (\text{repairingL}_{1} \text{ or repairing}_{1}) \text{ and rem}_{2}))

\hspace{1em}and \ longActCtrl(cL'_{i}, \text{repL}_{i}, \text{repairingL}_{i})
\hspace{1em}and \ longActCtrl(c'_{i}, \text{rep}_{i}, \text{repairing}_{i})
\hspace{1em}and \ longActCtrl(ca'_{i}, \text{add}_{i}, \text{adding}_{i}) \text{ and } (\text{crm}'_{i} \text{ or not rem}_{i})

\hspace{1em}i = 1, 2

with \(cL_{1}, c_{1}, ca_{1}, \text{crm}_{1}, cL_{2}, c_{2}, ca_{2}, \text{crm}_{2}\)

\((\text{repL}_{1}, \text{repairingL}_{1}, \text{rep}_{1}, \text{repairing}_{1}, \text{add}_{1}, \text{rem}_{1}, \text{adding}_{1})\)
\hspace{1em}= \text{coord\_lb-repl-tier}(cL_{1}, \text{failL}_{1}, \text{nrL}_{1}, c_{1}, \text{fail}_{1}, \text{nr}_{1}, ca_{1}, \text{crm}_{1}, o_{1}, u_{1}, na_{1});
\((\text{repL}_{2}, \text{repairingL}_{2}, \text{rep}_{2}, \text{repairing}_{2}, \text{add}_{2}, \text{rem}_{2}, \text{adding}_{2})\)
\hspace{1em}= \text{coord\_lb-repl-tier}(cL_{2}, \text{failL}_{2}, \text{nrL}_{2}, c_{2}, \text{fail}_{2}, \text{nr}_{2}, ca_{2}, \text{crm}_{2}, o_{2}, u_{2}, 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 \(\text{crm}'_{i}\) with associated objectives
Exploiting models with DCS: modularly (iv)

Two-application data-center

$\ldots = \text{two-data-center} (\ldots)$

\[
\begin{align*}
\text{enforce } ( & \text{(not ((Incr or Decr) and (repairing}_{ij} \text{ or adding}_{ij} \text{ or rem}_{ij})))} \\
& \text{and (not ((repairing}_{ij} \text{ or adding}_{ij}) \text{ and sd})} \\
& \text{and } \text{longActCtrl}(cL'_{ij}, repL_{ij}, repairingL_{ij}) \\
& \text{and } \text{longActCtrl}(c'_{ij}, rep_{ij}, repairing_{ij}) \\
& \text{and } \text{longActCtrl}(ca'_{ij}, add_{ij}, adding_{ij}) \text{ and (crm'}_{ij} \text{ or not rem}_{ij})) \\
i = 1, 2; j = 1, 2
\end{align*}
\]

with \(cL_{11}, c_{11}, \ldots,crm_{22}, ci, cd\)

\[
\ldots = \text{coord_apply} (cL_{11}, c_{11}, ca_{11}, crm_{11}, \ldots, cL_{21}, c_{21}, ca_{21}, crm_{21}, \ldots) \\
\ldots = \text{coord_apply} (cL_{12}, c_{12}, ca_{12}, crm_{12}, \ldots, cL_{22}, c_{22}, ca_{22}, crm_{22}, \ldots) \\
(si, sd, \text{Incr, Decr}) = \text{consolidation} (ci, cd, i, d, e);
\]

Two application & consolidation

1. Control of two coord-apply 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

<table>
<thead>
<tr>
<th>nb. app.</th>
<th>Synthesis time</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>monolithic</td>
<td>modular</td>
</tr>
<tr>
<td>1</td>
<td>0s</td>
<td>5s</td>
</tr>
<tr>
<td>2</td>
<td>49s</td>
<td>11s</td>
</tr>
<tr>
<td>3</td>
<td>42m24s</td>
<td>24s</td>
</tr>
<tr>
<td>4</td>
<td>&gt; 2 days</td>
<td>1m22s</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>4m30s</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>13m24s</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>25m57s</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>50m36s</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>2h11m</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>9h4m</td>
</tr>
</tbody>
</table>
Implementation and Evaluation

Uncoordinated execution

![Graphs showing uncoordinated execution for Application 1 and Application 2.](image)

Coordinated execution

![Graphs showing coordinated execution for Application 1 and Application 2.](image)
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.