Automata-based control for reconfigurable computing systems

Eric Rutten & a.

G. Delaval, H. Marchand, T. Bouhadina, Q. Sabah, N. de Palma, S. Gueye

LIG / INRIA, Grenoble

December 5, 2011
Control of computation adaptation as a closed control loop.
Use of **Discrete Event Systems** and supervisory control:
- Petri nets, language theory (R&W), automata (synchronous)
  - e.g. [Lafortune e.a.]

bullet Control of **computation** adaptation as a closed control loop

![Diagram showing the process of control adaptation](image)
Use of **Discrete Event Systems** and supervisory control:
Petri nets, language theory (R&W), automata (synchronous)
e.g. [Lafortune e.a.]

- Control of computation adaptation as a closed control loop
- BZR programming language, and Discrete Controller Synthesis to compute the decision component (controller)
A reactive language for the control of computing systems

- mixed imperative/declarative programming language BZR
- declarative invariance contracts are enforced upon
  imperatively described behaviors (automata)
A reactive language for the control of computing systems

- mixed imperative/declarative programming language BZR
  [http://bzr.inria.fr](http://bzr.inria.fr)
  declarative invariance contracts are enforced upon
  imperatively described behaviors (automata)

- compilation based upon Discrete Controller Synthesis (DCS)
  formal technique from supervisory control theory of DES
  events, states, control modes $\leftrightarrow$ automata (e.g., StateFlow)
Motivation

BZR

Controlling computing

Perspectives

A reactive language for the control of computing systems

- mixed imperative/declarative programming language BZR
  
  [http://bzr.inria.fr](http://bzr.inria.fr)

  declarative invariance contracts are enforced upon
  imperatively described behaviors (automata)

- compilation based upon Discrete Controller Synthesis (DCS)
  formal technique from supervisory control theory of DES
  events, states, control modes \(\leftrightarrow\) automata (e.g., StateFlow)

- application to adaptive and reconfigurable computing systems
  closed-loop adaptation controllers: flexible execution of
  functionalities w.r.t. changing resource and environment
A reactive language for the control of computing systems

- mixed imperative/declarative programming language BZR

declarative invariance contracts are enforced upon
imperatively described behaviors (automata)

- compilation based upon Discrete Controller Synthesis (DCS)
  formal technique from supervisory control theory of DES
events, states, control modes $\leftrightarrow$ automata (e.g., StateFlow)

- application to adaptive and reconfigurable computing systems
  closed-loop adaptation controllers: flexible execution of
  functionalities w.r.t. changing resource and environment
  1. component-based software with Fractal
  2. coordination of autonomic loops in data-centers
Examples of discrete computing modes

state $\leftrightarrow$ configuration

resource access, level of consumption/quality, ...
Examples of discrete computing modes

state $\leftrightarrow$ configuration

resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)

```
\begin{align*}
delayable(r,c,e) &= \text{act} \\
act &= \text{false} \quad \text{act} = \text{false} \\
\text{Idle} \quad r \land \neg c \quad \text{Wait} \\
\text{Active} \quad e \\
c \\
\text{act} &= \text{true}
\end{align*}
```
Examples of discrete computing modes

- state ↔ configuration
  - resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)

- modes: algorithm variants for a functionality (resource, QoS)

\[
delayable(r,c,e) = \begin{cases} 
  \text{false} & \text{if } r \land \neg c \\
  \text{true} & \text{if } r \land c \\
\end{cases}
\]

- fault tolerance: migration/rollback upon processor failure

- architecture control: frequency, DVS, stand-by in MPSoC
Examples of discrete computing modes

state $\leftrightarrow$ configuration

resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)
- modes: algorithm variants for a functionality (resource, QoS)
- placement and migration: task $T_i$ on processor/core $P_j$

\[
delayable(r,c,e) = \text{act}
\]

\[
\begin{align*}
\text{act} &= \text{false} \\
\text{act} &= \text{false} \\
\text{act} &= \text{true}
\end{align*}
\]

\[
\begin{array}{c}
\text{Idle} \quad r \land \neg c \\
\text{Wait} \\
\text{Active} \quad e \\
\text{Active} \quad r \land c \\
\text{Idle} \quad c \\
\text{Idle} \quad \text{act} = \text{true}
\end{array}
\]
Examples of discrete computing modes

- **state ↔ configuration**
- resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)
- modes: algorithm variants for a functionality (resource, QoS)
- placement and migration: task $T_i$ on processor/core $P_j$
- resource budgeting: proc./core taken for other application

```
act = false
r \land \neg c

act = false
e

r \land c

act = true
c
```

```
delayable(r,c,e) = act
```
Examples of discrete computing modes

state $\leftrightarrow$ configuration
resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)
- modes: algorithm variants for a functionality (resource, QoS)
- placement and migration: task $T_i$ on processor/core $P_j$
- resource budgeting: proc./core taken for other application
- fault tolerance: migration/rollback upon processor failure
Examples of discrete computing modes

state ↔ configuration

resource access, level of consumption/quality, ...

- computation task control
  (example of Heptagon node)
- modes: algorithm variants for a functionality (resource, QoS)
- placement and migration: task $T_i$ on processor/core $P_j$
- resource budgeting: proc./core taken for other application
- fault tolerance: migration/rollback upon processor failure
- architecture control: frequency, DVS, stand-by in MPSoC

```
delayable(r,c,e) = act
```

```
act = false
```

```
Idle
r \land \neg c
```

```
Wait
```

```
e
```

```
r \land c
```

```
c
```

```
Active
```

```
act = true
```

```
act = false
```
Discrete controller synthesis: principle

Goal

**Enforcing a temporal property \( \Phi \) on a system on which \( \Phi \) does not yet hold a priori**
Discrete controller synthesis: principle

**Goal**

Enforcing a temporal property \( \Phi \) on a system on which \( \Phi \) does not yet hold a priori

**Principle (on implicit equational representation)**

- **State**: memory
- **Trans**: transition function
- **Out**: output function

[Diagram of the system with labels Trans, State, and Out connected by arrows labeled Y and Z]
Discrete controller synthesis: principle

Goal

Enforcing a temporal property $\Phi$ on a system on which $\Phi$ 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 ($Y^u$)

![Diagram of discrete controller synthesis]

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

**Goal**

Enforcing a temporal property $\Phi$ on a system on which $\Phi$ 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 $\Phi$ ([invariance], reachability, attractivity, ...)

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

Controlling computing

BZR

Perspectives

BZR contracts and DCS

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

\[ e_A \implies e_G \]

with \( c_1, \ldots, c_q \)

\[ y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

\[ \ldots \]

\[ y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
BZR contracts and DCS

<table>
<thead>
<tr>
<th>$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_A \implies e_G$</td>
</tr>
<tr>
<td>with $c_1, \ldots, c_q$</td>
</tr>
</tbody>
</table>

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$
...
$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct:
  - assuming $e_A$ (on the environment), enforce objective $e_G$
  - by constraining the additional controllable variables $c_1, \ldots, c_q$ local to the component (with)
Motivation

BZR contracts and DCS

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

\[ e_A \implies e_G \]

with \( c_1, \ldots, c_q \)

\[ y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

\[ \ldots \]

\[ y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct:
  - assuming \( e_A \) (on the environment), enforce objective \( e_G \)
  - by constraining the additional controllable variables \( c_1, \ldots, c_q \) local to the component (with)
- encoded as a DCS problem (invariance)
  computes a local controller for each component
**Motivation**

**BZR**

**Controlling computing**

**Perspectives**

---

**BZR contracts and DCS**

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

\[ e_A \implies e_G \]

with \( c_1, \ldots, c_q \)

\[ y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

\[ \ldots \]

\[ y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct:
  - assuming \( e_A \) (on the environment), enforce objective \( e_G \)
  - by constraining the additional controllable variables \( c_1, \ldots, c_q \) local to the component (with)
- encoded as a DCS problem (invariance)
  - computes a local controller for each component

& G. Delaval & H. Marchand [ACM LCTES’10]
BZR hierarchy and modularity

**BZR composite contract node:**
re-use contracts of sub-nodes for the controller of the composite

assuming $e_A$, as well as $(e_{A1} \Rightarrow e_{G1})$ and $(e_{A2} \Rightarrow e_{G2})$

enforce $e_G$, as well as $e_{A1}$ et $e_{A2}$

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

**assume** $e_A$

**enforce** $e_G$

**with** $c_1, \ldots, c_q$

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

**assume** $e_{A1}$

**enforce** $e_{G1}$

\[
\ldots
\]

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

**assume** $e_{Ap}$

**enforce** $e_{Gp}$
BZR programming methodology

classical programming: write the solution, then verify
here: specify the problem, then the solution is derived

- **write nodes** describing the possible behaviors
  in the absence of control
  identify possible choice and control points
BZR programming methodology

classical programming: write the solution, then verify
here: specify the problem, then the solution is derived

- **write nodes** describing the possible behaviors
  in the absence of control
  identify possible choice and control points

- **write contracts** for the control objectives
  different objectives can be possible
  controllability for a specific objective is not always given
BZR programming methodology

classical programming: write the solution, then verify here: specify the problem, then the solution is derived

- **write nodes** describing the possible behaviors in the absence of control
  identify possible choice and control points

- **write contracts** for the control objectives
  different objectives can be possible controllability for a specific objective is not always given

- **compile** the program to obtain the controller using DCS can be seen as completion of partially specified program
BZR compilation & implementation

BZR specification
- contracts
- automata

Extension with contracts
- (synchronous) compiler

Objectives
- transition system

DCS tool
- (synchronous)

Controller (constraint)

Triangularize
- transl. to eq.

Controller (function)

Compose

Controlled automata

(synchronous)

Compiler

C, Java, ... sequential code

Development process:
- Integration in computing system (here: Orccad [IFAC11]):
  - Constraint generated C code (with constraint resolution)
  - Real-time Xenomai (C, Linux/RT & RP automata & contract BZR compiler spec.)
  - Orccad extract synchronous compiler DCS seq. C code Bool. eq. & obj.
Development process:
integration in computing system
(here: Orccad [IFAC11]):
Discrete control of tasks sequencings and mode changes

- Discrete and continuous **layers**

![Diagram of control systems](image-url)
Discrete control handlers of continuous control tasks

Discrete control of tasks sequencings and mode changes

- Discrete and continuous **layers**

  - Local task automata, coordinated by application automata with **discrete supervisor**, enforcing logical objective
Controlling computing systems: range of targets

in cooperation with field experts
different abstraction levels, different criteria for adaptation

- **Middleware and components**: Fractal/MIND
  HTTP server application (& ST Micro, P2012)
  & G. delaval, T. Bouhadiba, Q. Sabah [CBSE10, EMSOFT11]
Controlling computing systems: range of targets

in cooperation with field experts
different abstraction levels, different criteria for adaptation

- **Middleware and components**: Fractal/MIND
  HTTP server application (& ST Micro, P2012)
  & G. delaval, T. Bouhadiba, Q. Sabah [CBSE10, EMSOFT11]

- **Operating systems and virtual machines**: e.g. Jade,
  green computing, administration loops and their coordination
  & N. de Palma [GCM10] ; & O. Gruber & F. Boyer [FeBID11]
Controlling computing systems: range of targets

in cooperation with field experts
different abstraction levels, different criteria for adaptation

- **Middleware and components**: Fractal/MIND
  HTTP server application (& ST Micro, P2012)
  & G. delaval, T. Bouhadiba, Q. Sabah [CBSE10, EMSOFT11]

- **Operating systems and virtual machines**: e.g. Jade,
  green computing, administration loops and their coordination
  & N. de Palma [GCM10] ; & O. Gruber & F. Boyer [FeBID11]

- **Hardware and architectures**: partially reconfigurable FPGAs
  ANR Famous (INRIA Lille, LabSticc Lorient, U. Dijon, Sodius)
  & S. Guillet, Xin An (doct.) [Reconfig10, MBed11]
Controlling computing systems: range of targets

in cooperation with field experts
different abstraction levels, different criteria for adaptation

- **Middleware and components**: Fractal/MIND
  HTTP server application (& ST Micro, P2012)
  & G. delaval, T. Bouhadiba, Q. Sabah [CBSE10, EMSOFT11]

- **Operating systems and virtual machines**: e.g. Jade,
  green computing, administration loops and their coordination
  & N. de Palma [GCM10] ; & O. Gruber & F. Boyer [FeBID11]

- **Hardware and architectures**: partially reconfigurable FPGAs
  ANR Famous (INRIA Lille, LabSticc Lorient, U. Dijon, Sodius)
  & S. Guillet, Xin An (doct.) [Reconfig10, MBed11]

- **Control applications**: e.g. Orccad (NeCS/GIPSA-Lab)
  reconfiguration between continuous feedback schedulers
  & D. Simon & R. Pissard-Gibollet [IFAC11]
Middleware and components

- **Fractal** component-based framework
Middleware and components

- **Fractal** component-based framework
- **HTTP server** case study
  
  & G. Delaval, T. Bouhadiba, Q. Sabah  [CBSE10,EMSOFT11]

  Minalogic project MIND
A Running Example (Comanche HTTP server)

Comanche Server

frontend \rightarrow analyzer \rightarrow dispatcher \rightarrow file server1
A Running Example (Comanche HTTP server)
A Running Example (Comanche HTTP server)

** Components are deployed over 4 Processing Elements using Comete middleware

** Bindings $\Rightarrow$ Asynchronous communications with FIFOs

Comanche Server

Deployment

Execution Platform (4 processing elements)
A Running Example (Comanche HTTP server)

** Components are deployed over 4 Processing Elements using Comete middleware

** Bindings → Asynchronous communications with FIFOs

Possible Reconfigurations:
- Start/Stop Components
- Connect/Disconnect Components
A Running Example (Comanche HTTP server)

** Components are deployed over 4 Processing Elements using Comete middleware
** Bindings \(\Rightarrow\) Asynchronous communications with FIFOs

Possible Reconfigurations:
- Start/Stop Components
- Connect/Disconnect Components
- Migrate Components
A Running Example (Comanche HTTP server)

** Components are deployed over
4 Processing Elements using Comete middleware

** Bindings  ➔ Asynchronous communications with FIFOs

Possible Reconfigurations:
- Start/Stop Components
- Connect/Disconnect Components
- Migrate Components
Modeling Components

- Modular approach
- Distinguish between hardware and software models
- The global model == the synchronous composition of instances
Lifecycle model enable controlling the state of the component. It fires reconfiguration commands (start, stop, connect, disconnect).

FIFO size model receives its input from the application. It states on the size of a FIFO by means of its output (full).
Execution Platform Models

- Position enables controlling the position of a component in the execution platform.
- Proc() models the availability of a processing element.
The Design of the Manager

Instantiating models:
- Availability of processing elements.
- Component lifecycles and positions.
- FIFO sizes.
- Workload computation of processing elements

Describing objective properties (by means of Boolean formula) e.g.,:
- When file server1 is overloaded start the file server2.
- Keep the workload of processing elements lower than X unit.
- No component can run on an unavailable processing element.
An Example of Enforced Property

The property "When file server1 is overloaded start the file server2" is written as follows:

\[ \text{enforce}(\text{full} \implies \text{run} \land \neg\text{disc}) \]
Tools for the Design of the Manager

- Heptagon (Synchronous Moore machines)
- We use BZR (Heptagon + contracts & DCS)
- BZR program compilation results in two functions:
  - Init() initializes the memory
  - Step() computes the outputs and updates the global state depending on the inputs and the current state
- The synchronous program reads input events and outputs reconfiguration commands.
Manager Generation and Integration

Contents

1 Introduction & Motivation
2 Synchronous Manager for the Control of Reconfigurations
3 Modeling/Describing the Control State Space
4 Manager Generation and Integration
5 Conclusion & Perspectives
Manager Component-ization and Integration

prepare_events()  step()  generate_commands()
Manager Component-ization and Integration

prepare_events()  step()  generate_commands()

Manager

Interface evt

Interface cmd

interface SynchronousManager.api.Events{
  setEvent(in event_name, in event_value)
}

interface Comete.generic.api.ComponentManager {
  bind(...) ...
  start(...) ...
  migrate(in instance_id, in target_pe_id)
}
One reaction consists of:
- Components register events through `evt`
- Call to `prepare_events(); step(); generate_commands();`
- generated commands are function calls to Comete
Operating system administration

- system administration automation
- virtual machines
Operating system administration

- system administration automation
  virtual machines

- green computing
  managing power/energy

& N. de Palma & S. Gueye

and more to come! (ANR project CtrlGreen)
Techniques for administration loops coordination

Coordination Challenges
- Synchronizing AMs’ execution
- Logical control of AMs’ operations

Synchronous Approach
- Parallelism
- Synchronization
- Determinism
Ensures good performance while optimizing resources usage.

Dynamically adapts the number of replicated servers to the load on the system.
Dvfs

- Ensures good performance while optimizing the energy consumption
- Dynamically adapts the CPU frequency/voltage level of server to the load that server receives
Use of both Sizing and Dvfs administration loops

Objectives

- Local energy optimization: Dvfs
- Global energy optimization: Sizing

Efficient use of both managers

- Global optimization before Local optimization
  - May not be achieved without coordination
Coordination controller design

- Modeling system composed of managers sizing and Dvfs
- Synthesis of Discrete controller
Describes the control of Sizing operations:
- Sizing operations can be allowed/denied according to the value of *delay*

\[
\text{Delay\_control}(c) = \text{delay}
\]
- Describes Sizing execution modes
  - control of upsizing operations
- controllable variable: delay
Modeling Manager Dvfs

- Describes the different states in which the set of Dvfs managers could be
  - **Max**: All CPUs are in highest frequency
  - **Min**: All CPUs are in lowest frequency
  - **Normal**: Any other case

- no controllable variables

Dvfs_control(minimum, maximum) = min_freq, max_freq

![Diagram showing the states and transitions of Dvfs control](image)
Motivation

Our approach

Coordinating two energy-aware managers

Experimental evaluation

Conclusion

Synchronous control of Sizing and Dvfs

Coordination policies

- allow upsizing operations
  - when all processors are in their lowest frequency level

Control with

**Contract:** \((\text{freq\_max AND not delay}) \ OR \ (\text{not freq\_max AND delay})\)

\[
\text{Main (...) = ...}
\]

\[
\text{enforce (max\_freq and not delay) or (not max\_freq and delay)}
\]

With \(c\):

- **Active**
  - delay = false
  - not \(c\) /
  - \(c\) /

- **Idle**
  - delay = true

- **Down**
  - UpDown
  - \(... \ and \ not \ delay/\)
  - \(... \ and \ not \ delay/\)

- **Up**
  - \(... \ and \ not \ delay/\)
  - \(... \ and \ not \ delay/\)

- **Max**
  - Normal
  - Min
step 1: overload is false and max_freq is false
step 5: overload is true but max_freq is false: no upsizing operation
step 11: overload is true and max_freq is true: upsizing operation
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Experimental platform

3 Machines: Ubuntu Os
- **node 0**: 2.17 Ghz, 2.0Go RAM, Ubuntu-10.4
- **node 1**: 1.20 Ghz, 1.50Go RAM, Ubuntu-10.4
- **node 2**: 1.20 Ghz, 992.3Mo RAM, Ubuntu-10.4

Network
- Switch 3Com 4300 48PORT

Managed system: 2-tiers architecture
- **One Apache server**: Load balancer
  - Receives all requests from clients
  - Forwards them to Tomcat servers
- **Replicated Tomcat servers**
  - Treat client’s requests
  - Degree of replication may vary according to the system load

Use of Jmeter for the simulation of clients’ requests
- **Step 1**: increasing load during 2 minutes
- **Step 2**: constant load
Sizing operation disabled

Injection of load that is supported at maximum CPU frequency
Without coordination, the same load leads to upsizing operation and the increase of CPU frequency.
With coordination, the same load does not lead to upsizing operation.
Execution with coordination: Injecting higher load

With coordination, upsizing operation is performed only when it is necessary.
Conclusions

- **Discrete control integrated in programming language**
  Integration of DCS tool in compiler

- **Application of DCS to computing systems**
  - component-based software with Fractal
  - coordination of autonomic loops in data-centers

---

get BZR free! : bzr.inria.fr
Conclusions

- **Discrete control integrated in programming language**
  Integration of DCS tool in compiler
- **Application of DCS to computing systems**
  - component-based software with Fractal
  - coordination of autonomic loops in data-centers

Perspectives

- **more DCS** : efficiency, expressivity
  reachability, dynamical controllers, costs on paths [WODES10]
- **more elaborate models** of adaptive systems
  finer grain, e.g. fault tolerance [FMSD09]
- **more integration** in existing frameworks
  e.g. component-based Fractal [EMSOFT11]
- **more adaptive computing systems**
  reconfigurable FPGA architectures (ANR Famous)
  administration loops in data-centers (ANR CtrlGreen)