Designing Robust Self-Adaptation Managers for DPR FPGA Architectures Using Discrete Control

Soguy Mak-karé Gueye, Éric Rutten and Jean-Philippe Diguet

1Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble France
2CNRS UMR 6285, LAB-STICC, Université de Bretagne Sud F-56321 Lorient France

4th Grenoble Workshop on Autonomic Computing and Control
October 17, 2017
Plan

1. Introduction
2. Design approach
3. Design Methodology
   - Conditional objectives
   - Priority
4. Case Study
5. Conclusion
HPeC Project

1. Self-adaptive embedded systems (e.g., *Unmanned Vehicles*)
   1. Robust behaviors
   2. Manage themselves uncertainties

FPGA architectures

1. High performance computations
2. High flexibility
3. *Dynamic Partial Reconfiguration (DPR)*
DPR for supporting more Hardware \textit{than statically available}

Embedding all tasks necessary for achieving the missions
- Tracking Learning Detection (TLD)
- Obstacle detection, Path planning
- Search area landing, Forced landing
- Stabilization, Texture Analysis ...

Selecting the subset of tasks that must be active
- All tasks not needed to fulfill a mission
- Activate only the needed subset

Environment, system Health and processing quality
Achieving execution requirements

Task reconfigurable (versions)
- Compatibility in terms of resources
- Performance, Quality of Service (QoS)

Resources, peripheral reconfigurable
- Save power
Hierarchical control architecture

1. Selecting the subset of tasks that must be active
   - Environment & System health

2. Choosing the versions of the selected tasks to execute
   - Performance, QoS
   - Resource sharing

3. Scheduling the functions of the selected versions
   - Processing the sequences of reconfiguration
Self-adaptation manager

- Tasks (state, metrics)
- Tile state, CPUs state
- Peripheral devices state

Adaptation manager

- Tasks version
- Tile, CPU allocation
- Peripheral devices modes

Requests
- Activation
- Requirements

Internal state

Role

1. Enforces requests from upper level manager
   - Start/stop tasks
   - Execution requirements of the tasks

2. Dynamically re-organize the resources allocation
   - Execution requirements of the tasks
   - Tasks (metrics, state), resources state
Plan

1. Introduction

2. Design approach
   - Reactive languages
   - Discrete controller Synthesis (DCS)
   - Heptagon/BZR

3. Design Methodology
   - Conditional objectives
   - Priority

4. Case Study

5. Conclusion
Automata-based approach

We propose a design approach based on discrete control

1. **Automata-based modeling**
   - States (configurations)
   - Behaviors
   - Controllability

Reactive programming

1. **Reactive language**
   - Provides formal semantics
   - Formal verification (*model-checking*)

2. **Discrete Controller Synthesis**
   - Enabling declarative objectives
   - Generation of control logic enforcing objectives
Reactive languages

Programming language

1. Data flow / equations: \( \text{Input flows} \rightarrow \text{output flows} \)
2. Mode automata (FSM) 
   \((F. \text{Maraninchi, Mode-automata (2003))}\)
3. Parallel & hierarchical composition
4. Synchronization

Tools

1. Compilers (e.g., Heptagon)
2. Code generation (e.g., C, Java)

Delayable task

\[
(\text{active, dem}) = \text{task} (c, \text{req, fin})
\]

Two delayable tasks

\[
(\text{active1, active1, dem1,dem2}) = \text{twotasks} (c1, c2, \text{req1, c2, fin1, fin2})
\]
Model-checking

Verify properties

\[(\text{active1, active1, dem1, dem2}) = \text{twotasks} (c1, c2, req1, req2, fin1, fin2)\]

Tasks

not active at the same time?

Programming a solution for enforcing properties

1. **Complex, error-prone**
2. **Debug, repeat**
Discrete controller Synthesis (DCS)

Goal

Construct a *controller* that restrains a system to behaviors satisfying a property $\Phi$
Discrete controller Synthesis (DCS)

Goal
Construct a *controller* that restrains a system to behaviors satisfying a property $\Phi$.

Principle
- **State**: Memory
- **Trans**: Transition function
- **Out**: Outputs
Discrete controller Synthesis (DCS)

Goal
Construct a \textit{controller} that restrains a system to behaviors satisfying a property $\Phi$.

Principle

- \textit{State} Memory
- \textit{Trans} Transition function
- \textit{Out} Outputs

- Inputs partitioning
  - Uncontrollables $Y^u$ \textit{(e.g., failure, overload, ...)}
  - Controllables $Y^c$ \textit{(e.g., configuration parameters)}
Discrete controller Synthesis (DCS)

Goal

Construct a *controller* that restrains a system to behaviors satisfying a property $\Phi$

Principle

- **State** Memory
- **Trans** Transition function
- **Out** Outputs

- Inputs partitioning
  - Uncontrollables $Y^u$ (e.g., failure, overload, ...)
  - Controllables $Y^c$ (e.g., configuration parameters)

- Controller synthesis such that the system to be controlled satisfies $\Phi$ (*invariance, accessibility, ...*)
Discrete controller Synthesis (DCS)

Goal

Construct a **controller** that restrains a system to behaviors satisfying a property $\Phi$

Principle

**State** Memory

**Trans** Transition function

**Out** Outputs

- Inputs partitioning
  - Uncontrollables $\mathcal{Y}^u$ (*e.g.*, failure, overload, ...)
  - Controllables $\mathcal{Y}^c$ (*e.g.*, configuration parameters)

- Controller synthesis such that the system to be controlled satisfies $\Phi$ (*invariance*, *accessibility*, ...)

- Expressing **numerical** properties

SIGALI (H. Marchand e.a.)

REA$X$ (N. Berthier e.a.)
Heptagon/BZR

- Synchronous language
- Integrate Discrete Controller Synthesis
- Behavioral contract

(Compilation time)

(Compilation time)

(User-friendly)

Declaring mutual exclusion

\[
\text{node task}(c, \text{req}, \text{fin} : \text{bool}) \]

\[
\text{returns (active, dem} : \text{bool}) \]

let automaton

\[
\text{state Inactive do} \]

\[
\text{active} = \text{false}; \text{dem} = \text{r and c} ; \]

\[
\text{until req and c then Active} \]

\[
| \text{req and not c then Attente} \]

\[
\text{state Attente do} \text{active} = \text{false}; \text{dem} = \text{c} ; \]

\[
\text{until c then Active} \]

\[
\text{state Active do} \text{active} = \text{true}; \text{dem} = \text{false} ; \]

\[
\text{until fin then Inactive} \]

end tel

\[
(\text{active}_1, \text{dem}_1, \text{active}_2, \text{dem}_2) = \text{twotasks}(\text{req}_1, \text{req}_2, \text{fin}_1, \text{fin}_2) \]

\[
\text{assume not} (\text{req}_1 \text{and req}_2) \]

\[
\text{enforce not} (\text{active}_1 \text{and active}_2) \]

\[
\text{with} \ C_1, C_2 \]

\[
(\text{active}_1, \text{dem}_1) = \text{task}(C_1, \text{req}_1, \text{fin}_1); \]

\[
(\text{active}_2, \text{dem}_2) = \text{task}(C_2, \text{req}_2, \text{fin}_2) \]

No with stat. ⇒ model-checking

(Verification)
Plan

1. Introduction
2. Design approach
3. Design Methodology
   - Principle
   - Conditional objectives and Priority
     - Conditional objectives
     - Priority
4. Case Study
5. Conclusion
Behavioral models

- Architecture elements (e.g., processing resources)
- Available tasks and their different execution configurations

**Generic modeling patterns**
- Domain Specific Language (DSL)
- Framework: automatically constructs the automata

**Modeling in a systematic way**

1. State transition diagram for Proc and Stor:
   - State = tile (r, c1, c2, e)
   - State = Processing
   - State = storage

2. Task diagram for Vers1 and Vers2:
   - Task = \{res1, wcet1\}
   - Task = \{res2, wcet2\}
   - r & c1/
   - e /
   - r & c2/
   - not c1 & c2/
   - not r /
   - e & r & c1/

S. M. K GUEYE (INRIA)
Beharioval contract

- Declaring the control objectives

\[
\text{reconfig}\_\text{manager}(r_1, e_1, ...) = (\text{res}_1, \text{wcet}_1, ...)
\]
\[
\text{assume Env. properties}
\]
\[
\text{enforce Objectives}
\]
\[
\text{with } c_1, c_2, ...
\]
\[
(\text{res}_1, \text{wcet}_1) = \text{task}(r_1, c_1, c_2, e_1)
\]
\[
... = \text{tile}(...)
\]

- Discrete Controller Synthesis (DCS)
  - Construct logic enforcing the objectives
  - Maximally-permissive

Deterministic behavior
Behavorial (models + contract)

- Controller/autonomic manager

- Generated implementation
  - Functions

*The generated manager can be executed in an ARM processor*
Conditional objectives for Robustness

DCS tries to find a solution which enforces declared objectives all time

Enforcing execution requirements

1. Requirements defined at runtime
2. Sometimes not possible to achieve
3. Sometimes conflicting (in terms of resources)

Programming

1. Declaring **assume**
2. Programming

*must be guaranteed all situations must be considered*
Knowing the achievability of an objective

Hept/BZR

Use of control variables

1. As for controlling behaviors
2. Implementing policy $\Rightarrow$ actions $\Rightarrow$ behaviors

Possible to control policy implementation

We exploit the characteristics of H/BZR

1. Assigns value to the control variables with respect to their declaration order
2. First tries $\text{true}$ for boolean control variables.
   - It assigns $\text{false}$ to a control variable when $\text{true}$ leads to problem.
Conditional Objectives

Certain objectives are context-based achievable

- Maintain the execution time of a task lower than a max (max_thres)
  Switching to faster version
  \[ \text{obj} : (\text{time}_t \geq \text{max}_th) \Rightarrow (0 \ fby \ \text{wcet}) > \text{wcet} \]
  - When the fastest version achieved?

Knowing the achievability of the objective

\[ c_{poss} \Rightarrow ((\text{time}_t \geq \text{max}_th) \Rightarrow (0 \ fby \ \text{wcet}) > \text{wcet}) \]

- \( c_{poss} \) is true when \( \text{obj} \) can be satisfied
- Otherwise \( c_{poss} \) is false

The generated controller always switches to an implementation with lower wcet all time when it is possible
Defining priority

- Maintain the execution time of a task lower than a max (max_thres), and QoS greater than a min (min_qos)
  - $obj_1: (time_t \geq max_th) \Rightarrow ((0 \text{ fby wcet}) > wcet)$
  - $obj_2: (qos_t < min_qos) \Rightarrow ((0 \text{ fby qos}) < qos)$
  - When a state that satisfies $obj_1$ does not satisfy $obj_2$?

**Priority**: control variables/declaration order

- $obj_1': c_{ect} \Rightarrow ((time_t \geq max_th) \Rightarrow ((0 \text{ fby wcet}) > wcet))$
- $obj_2': c_{qos} \Rightarrow ((qos_t < min_qos) \Rightarrow ((0 \text{ fby qos}) < qos))$
- if $obj_1'$ prior then declare $c_{ect}$, $c_{qos}$
- else declare $c_{qos}$, $c_{ect}$

However, with this solution, the priority is fixed at design-time. One can want to **change the priority at run-time**.
Defining Dynamic Priority

**Priority**: control variables/declaration order

- \[\text{obj}_1' : c_{\text{ect}} \Rightarrow (\text{m_time} \geq \text{max_th}) \Rightarrow (0 \text{ fby wcet}) > \text{wcet})\]
- \[\text{obj}_2' : c_{\text{qos}} \Rightarrow (\text{m_qos} < \text{min_qos}) \Rightarrow (0 \text{ fby qos}) < \text{qos})\]

Implementing dynamic priority

- **Input variables**: \( i_1, i_2 \)
- **Declaring control variables**: \( c_1 \text{ then } c_2 \)
- **Declaring** \( c_{\text{ect}} \) and \( c_{\text{qos}} \) as local variables
  - \( c_{\text{ect}} = \text{if } i_1 \geq i_2 \text{ then } c_1 \text{ else } c_2 \)
  - \( c_{\text{qos}} = \text{if } i_2 > i_1 \text{ then } c_1 \text{ else } c_2 \)
Plan

1. Introduction

2. Design approach

3. Design Methodology
   - Conditional objectives
   - Priority

4. Case Study
   - Managing Search landing area task
   - Improving the model

5. Conclusion
Search landing area task

1. Rgb2gray
2. Median Filter
   - Reducing noise
3. Canny (Gaussian, Sobel)
   - Edge Detection
4. Morphological operator
   - dilate -> erode

Two Versions
- Software \((wcet = 1500ms)\)
- Hardware \((wcet = 60ms)\)
Control

DE1-Soc FPGA system

- HPS : Dual-core ARM (ROS/Linux OS)
  - Self-adaptation manager
  - Software version of the task
- FPGA : one tile
  - Hardware version of the task

The control objective consists in maintaining the execution time of the task between a minimum threshold and a maximum threshold.
Choose faster version when the executime time is greater then the maximum threshold

Choose lower version when the executime time is lower than the minimum threshold

```c
main(r, e, time_t, min_thres, max_thres, f, rp)
    = act, res, wcet, used, err, objective

assume true
enforce objective
with cp1, cp2, c1, c2, c, ...

(act, res, wcet) = task(r, c1, c2, e);
(used, err) = tile(c, f, rp);

objective = (cp1 ⇒ ((time_t > max_thres) ⇒ ((0 fby wcet) > wcet))) and
(cp2 ⇒ ((time_t < min_thres) ⇒ ((0 fby wcet) < wcet)))
...
```

S. M.K GUEYE (INRIA)
Execution of the generated manager

1. The manager starts the software version of the task
   - max_thres = 70:
     - (time_t > max_thres)
     - The manager switches to the hardware version

2. max_thres = 1500:
   - min_thres = 100
   - (time_t < min_thres)
   - The manager switches to the software version
Oscillations

1. The manager starts the software version of the task
2. \( \text{max\_thres} = 70 \):
   - \( \text{time}_t > \text{max\_thres} \)
   - The manager switches the hardware version
3. \( \text{max\_thres} = 1000 \):
   - \( \text{min\_thres} = 100 \)
   - \( \text{time}_t < \text{min\_thres} \)
   - Oscillations
Improving the model

- Choose faster version when the executime time is greater than the maximum threshold
- Choose lower version when the executime time is lower than the minimum threshold
- Avoid choosing a version with $wcet$ greater than the maximum threshold

```plaintext
main(r, e, time_t, min_thres, max_thres, f, rp)
    = act, res, wcet, used, err, objective

assume true
enforce objective

with cp1, cp2, cp3, c1, c2, c, ...

(act, res, wcet) = task(r, c1, c2, e);
(used, err) = tile(c, f, rp);

objective = (cp1 ⇒ (((0 fby wcet) < wcet) ⇒ (wcet < max_thres))) and
            (cp2 ⇒ ((time_t > max_thres) ⇒ ((0 fby wcet) > wcet))) and
            (cp3 ⇒ ((time_t < min_thres) ⇒ ((0 fby wcet) < wcet)))
            ...);
```
Case Study

Improving the model

After improvement

1. The manager starts the software version of the task
2. $\text{max\_thres} = 70$ :
   - $(\text{time}_t > \text{max\_thres})$
   - The manager switches the hardware version
3. $\text{max\_thres} = 1000$ :
   - $\text{min\_thres} = 100$
   - $(\text{time}_t < \text{min\_thres})$
   - Nothing
Plan

1. Introduction
2. Design approach
3. Design Methodology
   - Conditional objectives
   - Priority
4. Case Study
5. Conclusion
Conclusion

1. Methodology for designing self-adaptation manager
   1. FPGA architectures
   2. Discrete control

2. Case study on a task
   Search landing area

Perspective

1. DSL for automatic Automata generation
2. Modular design to break complexity
Hierarchical control

\[(\ldots) = \text{upperlevel\_ctrl} (\ldots)\]

\[\text{assume } e_A\]
\[\text{enforce } e_G\]
\[\text{with } c_{1i}, \ldots, c_{ni}\]

\[\ldots = \text{sub\_ctrl}_1 (c_{1i}, \ldots)\]

\[\text{assume } e_{A1}\]
\[\text{enforce } e_{G1}\]
\[\text{with } c_{1i}\]

\[\ldots = \text{sub\_ctrl}_n (c_{ni}, \ldots)\]

\[\text{assume } e_{An}\]
\[\text{enforce } e_{Gn}\]
\[\text{with } c_{ni}\]

- \(\forall \Box ((e_{A1} \Rightarrow e_{G1}) \land \ldots \land (e_{An} \Rightarrow e_{Gn}) \land e_A) \Rightarrow (e_G \land e_{A1} \land \ldots \land e_{An})\)

- For each contract a controller is generated

- Hierarchy of controllers
DCS: monolithic vs modular

- Previous experience:
  - Coordinating multiple autonomic managers
  - 1 consolidation manager
  - 6 managers per multi-tiers application

**Synthesis costs**

![Graph showing synthesis time and memory usage for monolithic and modular approaches.](image)

<table>
<thead>
<tr>
<th>nb. app.</th>
<th>Synthesis time monolithic</th>
<th>Synthesis time modular</th>
<th>Memory monolithic</th>
<th>Memory modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0s</td>
<td>5s</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>49s</td>
<td>11s</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>42m24s</td>
<td>24s</td>
<td>34.81MB</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>&gt; 2j. –</td>
<td>1m22s</td>
<td>&gt;149,56MB</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>4m30s</td>
<td>–</td>
<td>20.37MB</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>13m24s</td>
<td>–</td>
<td>53.31MB</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>25m57s</td>
<td>–</td>
<td>77.50MB</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>50m36s</td>
<td>–</td>
<td>115.59MB</td>
<td>–</td>
</tr>
<tr>
<td>9</td>
<td>2h11m</td>
<td>–</td>
<td>236.59MB</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>9h4m</td>
<td>–</td>
<td>479.15MB</td>
<td>–</td>
</tr>
</tbody>
</table>

0s → insignificant value
→ Synthesis completed