Using timed automata in the design process of the Océ printer datapaths

Georgeta Igna

Radboud Universiteit Nijmegen
IPA Herfstdagen

November 26, 2009
1. Introduction
   - Architecture
   - Octopus project
   - Timed automata model checkers

2. Models
   - Model
   - Scheduling rules

3. Analysis
   - Nondeterminism
   - Scenarios

4. Conclusions and future work
Outline

1. Introduction
   - Architecture
   - Octopus project
   - Timed automata model checkers

2. Models
   - Model
   - Scheduling rules

3. Analysis
   - Nondeterminism
   - Scenarios

4. Conclusions and future work
An Océ printer

Georgeta Igna

Radboud Universiteit Nijmegen

Using timed automata in the design process of the Océ printer datapaths
Octopus project

Goal
New methods for designing systems that can easily adapt to changes during usage

The goal of my subgroup
A tool that can predict the component usage of the adaptive data paths for multiple scenarios
Reasoning about adaptive data paths

1. Reason about component usage and performance of different architectures and choose the most optimal one.
2. Find adaptive schedules that optimize the performance of different scenarios in a balanced way.
Subject of this talk

1. Reason about component usage and performance of a specific Océ architecture
2. Comparison between two timed automata model checkers
   - Uppaal
   - Agent Framework
Architecture

IP - image processing (i.e. zooming, rotation, filter)
Data paths

(a) Scan to Email  
(b) Process from Store  
(c) Simple Print

* Optional
Using timed automata in the design process of the Océ printer datapaths
Uppaal

Full exploration of the state space using one search algorithm
Agent Framework

- Partial exploration of the state space
- One trace obtained by a collection of agents (search algorithms)
Agent Framework (2)

- Partial exploration of the state space
- One trace obtained by a collection of agents (search algorithms)
Agent Framework (3)

- Partial exploration of the state space
- One trace obtained by a collection of agents (search algorithms)
Agent Framework (4)

- Partial exploration of the state space
- One trace obtained by a collection of agents (search algorithms)
**Agent Framework (5)**

**Priced Timed Automata**

Timed Automata + **COST** variable

---

**TRACES**

1. \((l_1, x=y=0) \xRightarrow{(3)}_{12} (l_1, x=y=3) \rightarrow (l_2, x=0, y=3) \rightarrow (l_3, \ldots)\) \(\Sigma c=17\)

2. \((l_1, x=y=0) \xRightarrow{(2.5)}_{10} (l_1, x=y=2.5) \rightarrow (l_2, x=0, y=2.5) \xRightarrow{(5)}_{1} (l_2, x=0.5, y=3) \rightarrow (l_3, \ldots)\) \(\Sigma c=16\)

3. \((l_1, x=y=0) \rightarrow (l_2, x=0, y=0) \xRightarrow{(3)}_{6} (l_2, x=3, y=3) \rightarrow (l_2, x=0, y=3) \rightarrow (l_3, \ldots)\) \(\Sigma c=11\)
Outline

1 Introduction
   - Architecture
   - Octopus project
   - Timed automata model checkers

2 Models
   - Model
   - Scheduling rules

3 Analysis
   - Nondeterminism
   - Scenarios

4 Conclusions and future work
Resource Automaton

start_component?
work=memory_used,
addTask(component_id, work),
recomputeWork(),
old_rate=crt_rate[component_id],
x := 0

(crt_rate[res_id]!=old_rate) ||
      (crt_rate[res_id]>0 imply
       x <= divide(work, crt_rate[res_id]))
end_resource!
removeTask(res_id),
work=0

old_rate!=crt_rate[res_id]
urg!

i:int[0,max_exec_time]
i<=x && (i+1)>x
update_work(i),
old_rate=crt_rate[res_id],
x=0

(crt_rate[res_id]!=old_rate) &&
  (crt_rate[res_id]>0 &&
   x >= divide(work, crt_rate[res_id]))
end_resource!
removeTask(res_id),
work=0

(crt_rate[res_id]!=old_rate) ||
  (crt_rate[res_id]>0 imply
   x <= divide(work, crt_rate[res_id]))
end_resource!
removeTask(res_id),
work=0

Georgeta Igna
Radboud Universiteit Nijmegen IPA Herfstdagen

Using timed automata in the design process of the Océ printer datapaths
Job Automaton - Process from Store

```
prev(download_prev_page[job_id]) && pages_started<pages_required
start_new_page!
pages_started++
start_download!
print_memory=>page_memory_req() && prev(download_prev_page[job_id])
allocate_page_memory()
end_download?
download_prev_page[job_id]=page_id
zoom_factor==100 & chan_skip_ip2!
start_ip2!
zoom_factor=100 && prev(ip2_prev_page[job_id])
start_ip2!
ip2_prev_page[job_id]=i
end_ip2?
print_memory+=download_memory,
x_clock<=delay_histogram
x_clock>=delay_histogram
chan_delay_histogram!
page_done!
update()
prev(ip4_prev_page[job_id])
start_ip4!
prev(ip4_prev_page[job_id])
start_ip4!
ip4_prev_page[job_id]=page_id
end_ip4?
release_memory_after_ip4(),
add_page_to_upload_array()
end_upload?
download_prev_page[job_id]=page_id
print_memory>=page_memory_req() && prev(download_prev_page[job_id])
start_download!
allocate_page_memory()
end_download?
download_prev_page[job_id]=page_id
print_memory+=ip4_memory
```
Scheduling rules

- Non-overtaking of file pages
- Upload in order
- Bus throttling
- Print jobs higher priority
Non-overtaking of file pages

Example

One job: 1 file with 3 pages, datapath: Simple print
Non-overtaking of file pages

Example

One job: 1 file with 3 pages, datapath: Simple print
Non-overtaking of file pages

Example

One job: 1 file with 3 pages, datapath: Simple print
Upload in order

- USB Upload is the slowest component of the system
- Many jobs wait for upload
- Waiting jobs should be taken in order
- Waiting list:
  - Scan jobs added after Scanning
  - Print jobs added after IP4
Print jobs - higher priority

Example:
Two job types:
1. Scan to Email
2. Process from Store
Print jobs - higher priority

Example:

Two job types:
1. Scan to Email
2. Process from Store

Print jobs higher priority over scan jobs
Bus throttling

\[ \text{Speed}(\text{component}) = f(\text{bandwidth}(\text{component}), \text{component\_id}) \]

Example:
Bus bandwidth allocation

![Diagram showing bandwidth allocation for different components with specific percentages.]
Bus throttling (2)

Bus bandwidth allocation:

![Diagram showing bus bandwidth allocation]

- IP4: 40%
- IP2: 40%
- UP: 10%
- 10%
- Scanner: 25%
Bus throttling (3)

Priority list:

1. **Scanner**
2. Printer
3. IP4
4. IP1
5. IP2
6. IP3
7. USB Download
8. USB Upload
Bus throttling (4)

Bus bandwidth allocation

<table>
<thead>
<tr>
<th></th>
<th>IP4</th>
<th>IP2</th>
<th>Scanner</th>
<th>Upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>40%</td>
<td>35%</td>
<td>25%</td>
<td>0%</td>
</tr>
</tbody>
</table>

The speed of IP2 and Upload changed
Outline

1 Introduction
   - Architecture
   - Octopus project
   - Timed automata model checkers

2 Models
   - Model
   - Scheduling rules

3 Analysis
   - Nondeterminism
   - Scenarios

4 Conclusions and future work

Using timed automata in the design process of the Océ printer datapaths
Scenario description

- Two concurrent jobs
- Each job contains one large file (memory full)
- End-to-end latency
  - path where all the jobs are complete (fastest path)
Nondeterminism (2)

- Océ case study - fully deterministic
- Timed automata
  - nondeterminism left
  - no partial order reduction
    i.e. two independent transitions are both explored
  - state space explosion
Avoidance of state space explosion
- Uppaal: priorities on channels and processes
  - No guarantee that we obtain the optimal schedule
- Agent Framework: cost function for each *job automaton*
Scenario 1

Simple Print job || Process from Store job

Datapaths:
- Process from Store: *USB Download, IP2, IP4, USB Upload*
- Simple Print: *USB Download, Print*

Total number of pages processed: 40 - Process from Store job, 30 - Simple Print job

Uppaal - priorities on processes - not useful
Agent framework - cost functions - useful
Scenario 2

Scan to Email job || Process from Store job

Datapaths:

- Scan to Email: Scanner, IP2, IP3, IP4, USB Upload
- Process from Store: USB Download, IP2, IP4, USB Upload

Total number of pages processed: 80 - scan job, 60 - print job

Schedule - Agent framework better then schedule Uppaal (7%)
Scenario 3

Scan to Email job || Process from Store job

Datapaths:

- Scan to Email: *Scanner, IP2, IP3, IP4, USB Upload*
- Process from Store: *USB Download, IP2, IP4, USB Upload*

Total number of pages processed: 80 - scan job, 60 - print job

*No bus throttling used*: a component waits until it gets all the bandwidth needed

New Uppaal Schedule 17% better then the old schedule (11% better then the old agent framework results)

Agent framework: - no results
Outline

1. Introduction
   - Architecture
   - Octopus project
   - Timed automata model checkers

2. Models
   - Model
   - Scheduling rules

3. Analysis
   - Nondeterminism
   - Scenarios

4. Conclusions and future work
Conclusions and future work

Conclusions:
- agent framework - promising timed automata model checker
- the architecture allows some optimization (i.e. no bus throttling)

Future work:
- validate the model
- separate scheduling rules from job automata
Thank you!
Questions?