Schedulability Analysis of Timed CSP Models Using the PAT Model Checker

Oğuzcan OĞUZ
Jan F. BROENINK
Angelika MADER

Robotics and Mechatronics, University of Twente, The Netherlands
Contents

- Problem Statement & Approach
- Schedulability Analysis Framework
  - Platform Specific Model Construction
  - Analysis of Platform Specific Model
- Example: Analysing the Model of a Robot Control
- Summary & Future Work
Problem Statement

- Two main concerns for reliable embedded system design
  - Concurrency
  - Timeliness

- CSP & Timed CSP for concurrency and timed reasoning
- Tools to model-check CSP and Timed CSP
  - FDR v2.94 & PAT
- CSP-based languages and libraries for implementation
  - Scheduling for real-time applications due to limited resources

- How to check timeliness of a CSP-based implementation?
  - Timed CSP has a ‘maximal parallelism’ assumption
Approach: Schedulability Analysis

- A schedulability analysis framework
  - Schedulability analysis of Timed CSP models
  - Scheduling scheme: Non-preemptive fixed-priority
  - Multiprocessor support
  - Employs PAT model checker for dense-time model checking

- The schedulability analysis workflow
  - Construct a **Platform-Specific Process (PSP)** from a given **Platform-Independent Process (PIP)**
  - Analyse the resulting Platform-Specific Process
Schedulability Analysis Workflow

Platform-Independent Process (PIP) and Platform-Independent Timing

Execution Platform Constraints: Number of CPUs, BCETs & WCETs

Hardware Mapping and Priority Assignments

Construct Platform-Specific Process (PSP)

Verify Specifications

PSP

Deadlines and Liveness Specifications

Satisfied?

No

Revise

LEGEND:

PROCESS
DATA
DECISION
Platform-Independent Process (PIP)

- An **untimed process** for platform-independent behaviour
- A fixed number of **task events**

A simple PIP example:

- \[ P_0 = p0\_in \rightarrow \text{task.0} \rightarrow \text{write\_setpoint} \rightarrow P_0; \]
- \[ P_1 = \text{read\_setpoint} \rightarrow \text{task.1} \rightarrow \text{task.2} \rightarrow p1\_out \rightarrow P_1; \]
- \[ \text{SYSTEM} = P_0 \mid \mid \mid P_1; \]
Construction Steps:

1. Instrument PIP with platform-independent timing
2. Specify hardware mapping, priorities and execution times
3. Add scheduling behaviour
Step 1: Add platform-independent timing

- Instrument PIP with platform-independent timing
  - Cycle times for periodic processes
  - Minimum inter-arrival times for sporadic events
  - Timeout points
  - Urgent events

- Adding timing to the example PIP process:

  \[
  P0 = \text{p0\_in } \rightarrow \text{task.0 } \rightarrow \text{write\_setpoint } \rightarrow P0; \\
  P1 = \text{read\_setpoint } \rightarrow \text{task.1 } \rightarrow \text{task.2 } \rightarrow \text{p1\_out } \rightarrow P1; \\
  \text{SYSTEM } = P0 \ || | \ || P1;
  \]
Step 1: Add platform-independent timing

- Instrument PIP with platform-independent timing
  - Cycle times for periodic processes
  - Minimum inter-arrival times for sporadic events
  - Timeout points
  - Urgent events

- Adding timing to the example PIP process:

```plaintext
P0 = ((p0_in \rightarrow \text{task.0} \rightarrow \text{write_setpoint} \rightarrow \text{Skip}) || \text{Wait[20]}); P0;
P1 = ((\text{read_setpoint} \rightarrow \text{task.1} \rightarrow \text{task.2} \rightarrow \text{p1_out} \rightarrow \text{Skip}) || \text{Wait[10]}); P1;
SYSTEM = P0 || P1;
```
Step 2: Mapping, Priorities & Exec. Times

- Mapping of the Processes
  - PRIORITY: Priority of the mapped process
  - CPU_ID: Id of the CPU that the mapped process is assigned to

Sample Array:
```javascript
var mp_arr[2][2] = [1,0, //mp_id=0: P0
    2,0]; // 1: P1
```

- Task Attributes
  - BCET: Best case execution time
  - WCET: Worst case execution time
  - MP_ID: Id of the mapped process that the task belongs to

Sample Array:
```javascript
var task_arr[3][3] = [4,6,0, //t_id=0: task.0
    1,3,1, // 1: task.1
    1,3,1]; // 2: task.2
```
Step 3: Add Scheduling Behaviour

- Scheduling behaviour is defined by two template processes
  - **TASK** Template Process
    - Represents executional tasks in the system
    - Synchronizes with the assigned CPU process
  - **CPU** Template Process
    - Represents a CPU - Models the scheduling and execution of the tasks
    - Synchronizes with the assigned TASK processes
  - Replace all task events in PIP with TASK process instances
  - Put a number of CPU process instances in parallel with PIP
Before adding TASK & CPU processes:

\[
P0 = ((p0\_in \rightarrow \text{task.0} \rightarrow \text{write\_setpoint} \rightarrow \text{Skip}) ||| \text{Wait}[20]); \ P0;
\]

\[
P1 = ((\text{read\_setpoint} \rightarrow \text{task.1} \rightarrow \text{task.2} \rightarrow p1\_out \rightarrow \text{Skip}) ||| \text{Wait}[10]); \ P1;
\]

\[
\text{SYSTEM} = P0 ||| P1;
\]

The resulting PSP instrumented with TASK & CPU processes:

\[
P0 = ((p0\_in \rightarrow \text{TASK(0)}; \text{write\_setpoint} \rightarrow \text{Skip}) ||| \text{Wait}[20]); \ P0;
\]

\[
P1 = ((\text{read\_setpoint} \rightarrow \text{TASK(1); TASK(2); p1\_out} \rightarrow \text{Skip}) ||| \text{Wait}[10]); \ P1;
\]

\[
\text{PSP\_SYSTEM} = (P0 ||| P1) ||| (\text{CPU(0)} ||| CPU(1));
\]
- Two sets of verifications
  - Schedulability Analysis
  - Verifying liveness properties
Schedulability Analysis

- Specifying deadlines on PSP
  - Mark start and end points for each time constrained process
  - Put DEADLINES process in parallel with PSP

```
P0 = ((d_start.0 ↠ p0_in ↠ TASK(0); write_setpoint ↠ d_end.0 ↠ Skip) ||| Wait[20]);
P0;

P1 = ((d_start.1 ↠ read_setpoint ↠ TASK(1); TASK(2); p1_out ↠ d_end.1 ↠ Skip) ||| Wait[10]);
P1;

PSP_SYSTEM = (P0 ||| P1) || (CPU(0) ||| CPU(1)) || DEADLINES;
```

- Check if any of the deadlines can be missed ever

```
#assert PSP_SYSTEM |= []!(missed.0 || missed.1);
```

- missed.i events denote violations of the specified deadlines
Verification of liveness properties

- PSP is a trace timewise refinement of PIP
  \[ PIP_T \sqsubseteq_{TF} PSP \]

- A finite trace of PSP is also a trace of PIP
- PSP satisfies all the safety properties of PIP
- Verify deadlock freedom and liveness specifications on PSP

```plaintext
#assert PSP_SYSTEM deadlockfree;
```
R2-G2P: A mobile, 2-wheeled robot

- 2 CPUs
- 2 Line sensors
- 2 Distance sensors
- Contact Sensor
- 2 Encoders & 2 Servo Motors
The robot is supposed to
- Drive forward following a black line on the floor
- Keep a predefined distance to any obstacles in the driving direction
- Stop when it goes off the line or bumps into an obstacle

Initial control design results in a two level design
- A sequence controller with a period of 80
- A loop controller with a period of 20
PIP: ROBOT_CONTROL process

ROBOT_CONTROL

- read_distance_sensors
- read_line_sensors
- read_bump_sensor
- read_odometers

SEQUENCE_CONTROL

- motor_speed_setpoints

LOOP_CONTROL

write_pwms

27/08/2012
### Execution times & HW Mapping

**Legend:**
- LOOP_CONTROL process
- SEQUENCE_CONTROL process

**Dependencies:**
- SPEEDOMETER(n) → MOTOR_CONTROL(n)
- OBJECT_DISTANCE → ROBOT_SPEED → MOTOR_SPEED

#### LOOP_CONTROL
**Period/Deadline = 20**

#### SEQUENCE_CONTROL
**Period/Deadline = 80**

<table>
<thead>
<tr>
<th>Process</th>
<th>BCET</th>
<th>WCET</th>
<th>Priority</th>
<th>CPU Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEEDOMETER(0)</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>SPEEDOMETER(1)</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOTOR_CONTROL(0)</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOTOR_CONTROL(1)</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>OBJECT_DISTANCE</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROBOT_SPEED</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MOTOR_SPEED</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Verifying schedulability fails!
  - Witness traces indicate the reason is a multi-processor scheduling anomaly

27/08/2012
A Multi-Processor Scheduling Anomaly

- A good schedule with all tasks taking their WCET:

  - CPU 0: SPEEDOMETER(0) → MOTOR_CONTROL(1)
  - CPU 1: SPEEDOMETER(1) → MOTOR_CONTROL(0) → OBJECT_DISTANCE

- A deadline violation, SPEEDOMETER(1) takes less than its WCET:

  - CPU 0: SPEEDOMETER(0) → MOTOR_CONTROL(1) → ROBOT_SPEED → MOTOR_SPEED
  - CPU 1: SPEEDOMETER(1) → OBJECT_DISTANCE → MOTOR_CONTROL(0)

Legend:
- LOOP_CONTROL process
- SEQUENCE_CONTROL process

Dependencies:
- SPEEDOMETER(n) → MOTOR_CONTROL(n)
- OBJECT_DISTANCE → ROBOT_SPEED → MOTOR_SPEED

27/08/2012  CPA2012
A modified mapping of the processes:

<table>
<thead>
<tr>
<th>Process</th>
<th>Priority</th>
<th>CPU Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEEDOMETER(0)</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>SPEEDOMETER(1)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOTOR_CONTROL(0)</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>MOTOR_CONTROL(1)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>OBJECT_DISTANCE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROBOT_SPEED</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MOTOR_SPEED</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Schedulability query holds!
Summary

- A schedulability framework for Timed CSP
  - Non-preemptive fixed-priority, multiprocessor scheduling

- An associated schedulability workflow
  - PIP → PSP → Analysis

- Non-pessimistic schedulability analysis of CSP-based designs
Future Work

- Investigation of scalability

- Extensions
  - Support more scheduling schemes with
    - Preemption
    - Dynamic priorities
  - Incorporate communication times in the framework