# Project R5: Eclipse view for task executions

Samy Dafir
Dominik Baumgartner
Sophie Reischl

25. Januar 2017

# Content

# Aufgabenstellung

## Aufgabe:

- Eclipse Plugin
- Daten aus Files einlesen, diese dann grafisch darstellen
- Mittels Nebula XY Graph

# Data Files

## Binary File:

- Data von Typ:
  typedef struct monRec {
  double timeStamp;
  double value;
  double ID;
  } MON_RECORD;
- Jedes file: *Tasks_* < *name_of_core* > *.vdt*

# Data Files

## XML File:

```
<actor name="Core_A1" type="CPUSCHEDULER"' ID="2">
   <itemlist type="Task" nrOfItems="2">
      <item name="Task_LET_DRV_TASK_A1">
         <itemAttribute name="ID" value="1" />
         <property name="Priority" value="10" />
      </item>
      <item name="Task_A1_10msT1_LET01">
         <itemAttribute name="ID" value="2" />
         <property name="Priority" value="5" />
      </item>
   </itemlist>
</actor>
```

# User Eingaben:

## XML File:

- Benutzer wählt binary files und xml files
- Danach werden die gewünschten Tasks ausgewählt
- Graph mit gewählten Tasks wird erstellt

# Display Output:

## Graph:

- Graph in Cores eingeteilt
- Jeder Task nach Priorität in Cores eingeteilt
- Benutzer kann zoomen auf der Zeitachse

# Implementation

## What do we start with?

xml file: task name, id, priority binary files: id, states, tiestamps

## What is there to do?

1. Parse xml file
2. Select processes from list
3. Parse binary files $\rightarrow$ extract state info
4. Map process to all its states
5. Insert all processes into xy Graph

# Parse XML

- Parse xml with simple DOM parser.
- Create HashMap of all processes.

| Taskname | TaskInfo |
|---|---|
| Taskname 1 | id = 4, priority = 8 |
| Taskname 2 | id = 2, priority = 4 |
| Taskname 3 | id = 3, priority = 9 |
| ... | ... |

# Parse binary files

Only get relevant info: Processes the user selected

| Task Name | Core | ID | Priority |
|---|---|---|---|
| ☑ TaskMainacrnkout_c0_30cam | Core_c0 | 0.0 | 7.0 |
| ☐ TaskMainaigtout_Act_CndEvt... | Core_c0 | 1.0 | 16.0 |
| ☐ TaskMainaigtout_Act_CndEvt... | Core_c0 | 2.0 | 16.0 |
| ☑ TaskMainasp1_sp1tsk | Core_c0 | 3.0 | 19.0 |
| ☐ TaskMainacrnkout_c0_10catask | Core_c0 | 4.0 | 8.0 |
| ☑ TaskMainaigtout_Act_CndEvt... | Core_c0 | 5.0 | 16.0 |
| ☑ TaskMainspp_sttrevt_c0_8msl | Core_c0 | 6.0 | 2.0 |
| ☐ TaskMainspp_sttrevt_c0_2msm | Core_c0 | 7.0 | 8.0 |
| ☐ TaskMainacrnkout_c0_crnkinit | Core_c0 | 8.0 | 5.0 |
| ☐ TaskMainspp_sttrevt_c0_4msm | Core_c0 | 9.0 | 5.0 |

# Parse binary files

- Get selected ids from HashMap
- Go through complete binary file
- Read state and timestamp info
- Only record states if ID selected
- All states for ID collected in HashMap

# Parse binary files

**Resulting HashMap:**

| ID | StateInfo |
|---|---|
| 1 | List(states, timestamps) |
| 2 | List(states, timestamps) |
| ... | ... |

# Combine process and state info

## Combine State and TaskInfo

- contains all relevant info
- TreeMap provides instrinsic sorting
- Prefill set with TaskInfo
- For each entry: get States from HashMap

| **TraceInfo** |
| --- |
| name1, core1, priority1, stateList1 |
| name2, core2, priority2, stateList2 |
| ... |

# Build graph

- Traces sorted by core and priority
- Traces contain states (1-4)
- Iterate over tree
- Calculate offset for each task
- Add to graph

# Beispiel: