An Executable Semantics for Synchronous Task Graphs: From SDRT to Ada

Morteza Mohaqeqi

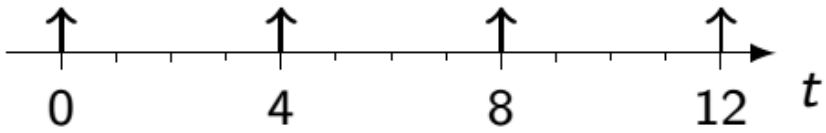
Jakaria Abdullah

Wang Yi

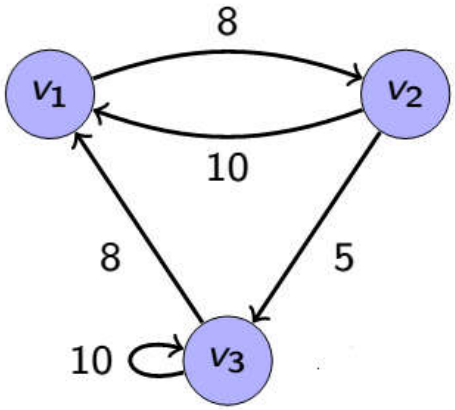
Uppsala University, Sweden

Real-Time Task Models

Release Pattern



Periodic/Sporadic



Graph-based

UPPSALA UNIVERSITET

Introduction

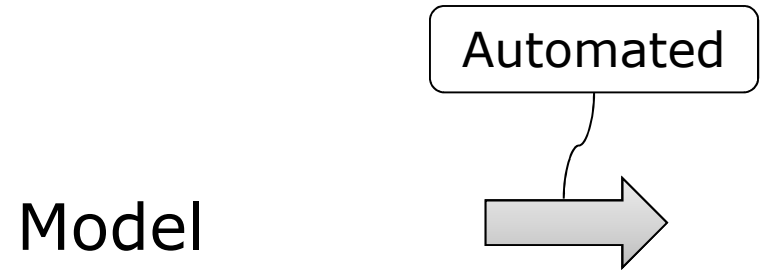
Real-Time Task Models

Release Pattern

Tasks' Dependency

Priorities

Model-based Approach



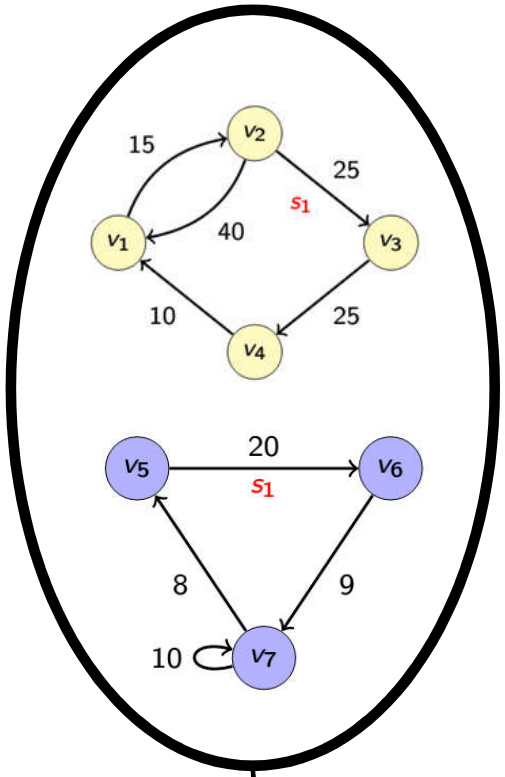
Release Pattern

Tasks' Dependency

Priorities

An executable program conforming the model

Overview



SDRT Task Set

```
    when g1 =>
       accept A;
       next_state := u;
       goto loop_start;
or
       delay until p2;
end select;
if g2 then
       Task_2.B;          -- Entry call to Task_2
       next_state := v;
       goto loop_start;
end if;
select
       -- Repetition of the code appeared in Lines 3 to 6
or
       delay until p3;
end select;
select     -- A selective accept
       -- Repetition of the code appeared in  Lines 3 to
or
       when g3 =>
       accept C;
       next_state := w;
       goto loop_start;
```

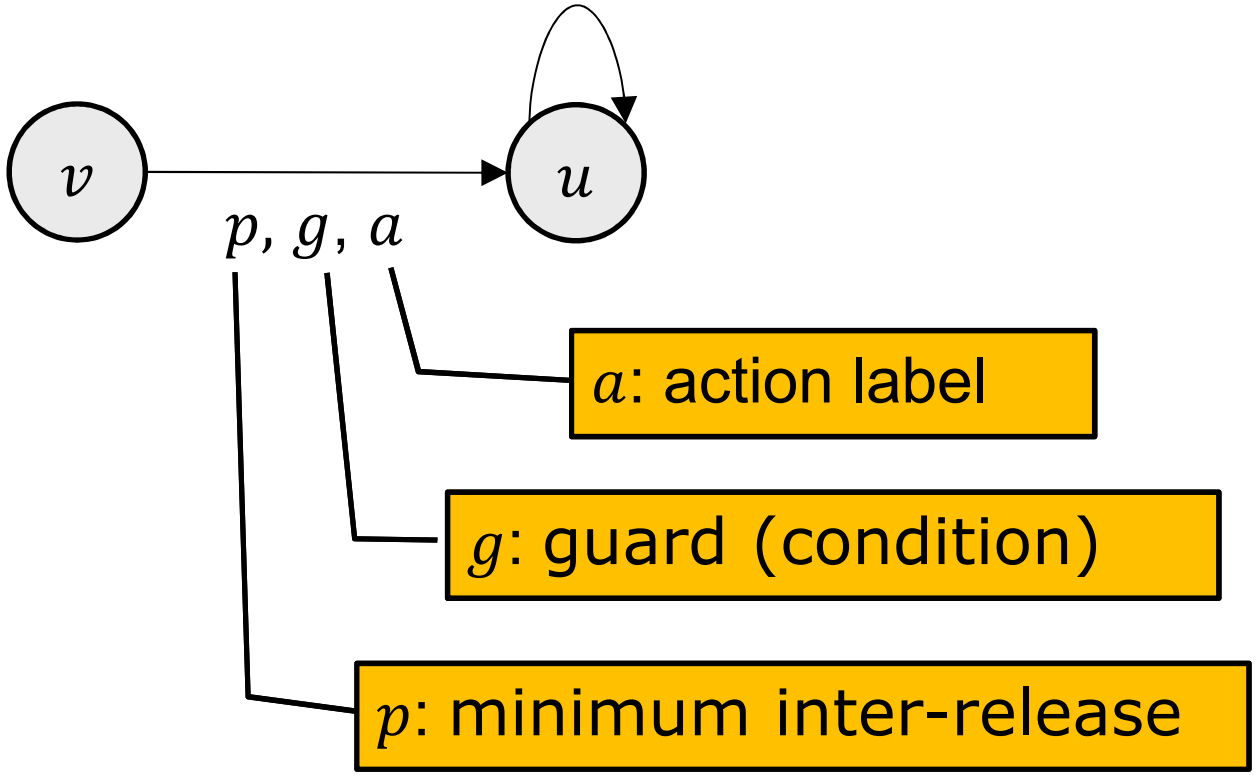
Ada Source Code

UPPSALA
UNIVERSITET

The SDRT Task Model

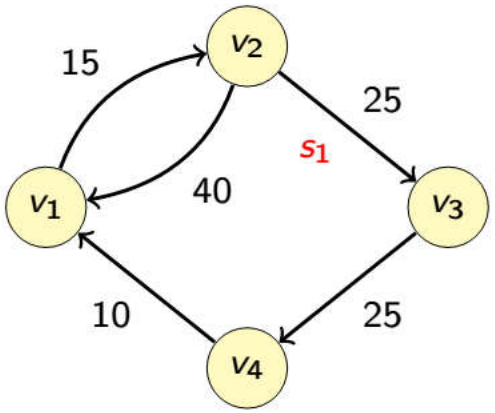
The SDRT Task Model

- Synchronous Digraph Real-Time Tasks

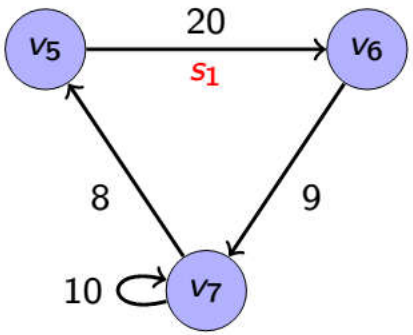


SDRT Example

Task T_1:

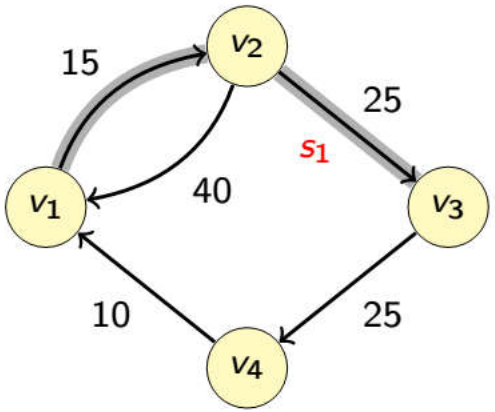


Task T_2:

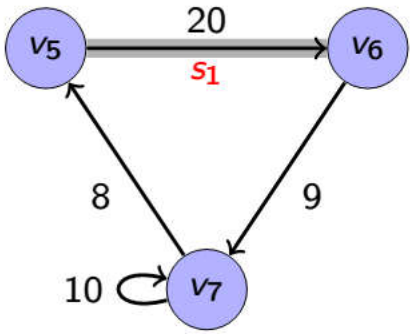


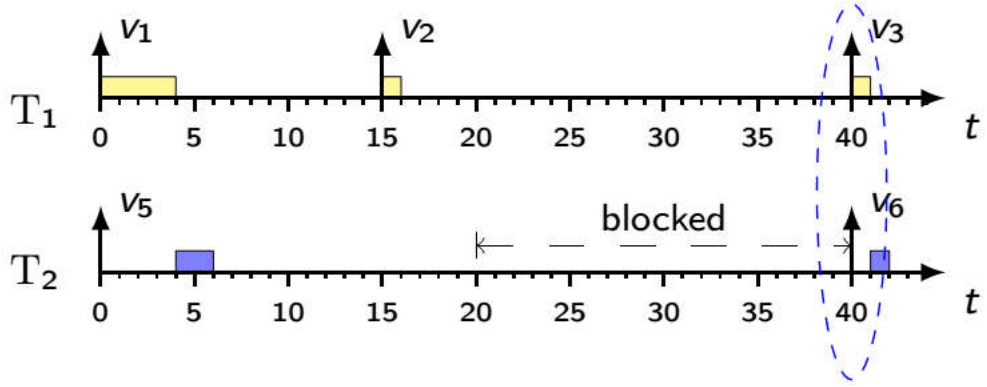
SDRT Example

Task T_1:



Task T_2:





UPPSALA
UNIVERSITET

Operational Semantics

UPPSALA UNIVERSITET

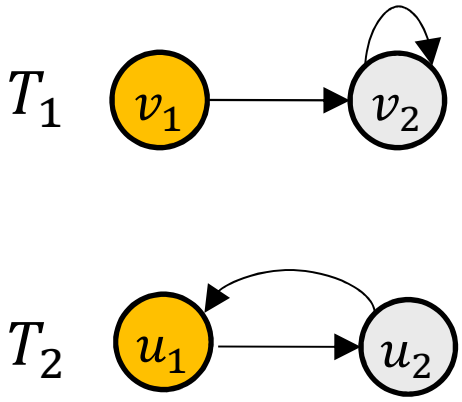
Operational Semantics

Labeled Transition System

- State: $(\bar{v}, \bar{\sigma}, \bar{c})$
  - $\bar{v} = \langle v_1, \ldots, v_n \rangle$: type of the last released jobs
  - $\bar{\sigma} = \langle \sigma_1, \ldots, \sigma_n \rangle$: the current valuation of the variables
  - $\bar{c} = \langle c_1, \ldots, c_n \rangle$: clocks →
    - the time passed after the release of the last jobs

Labeled Transition System

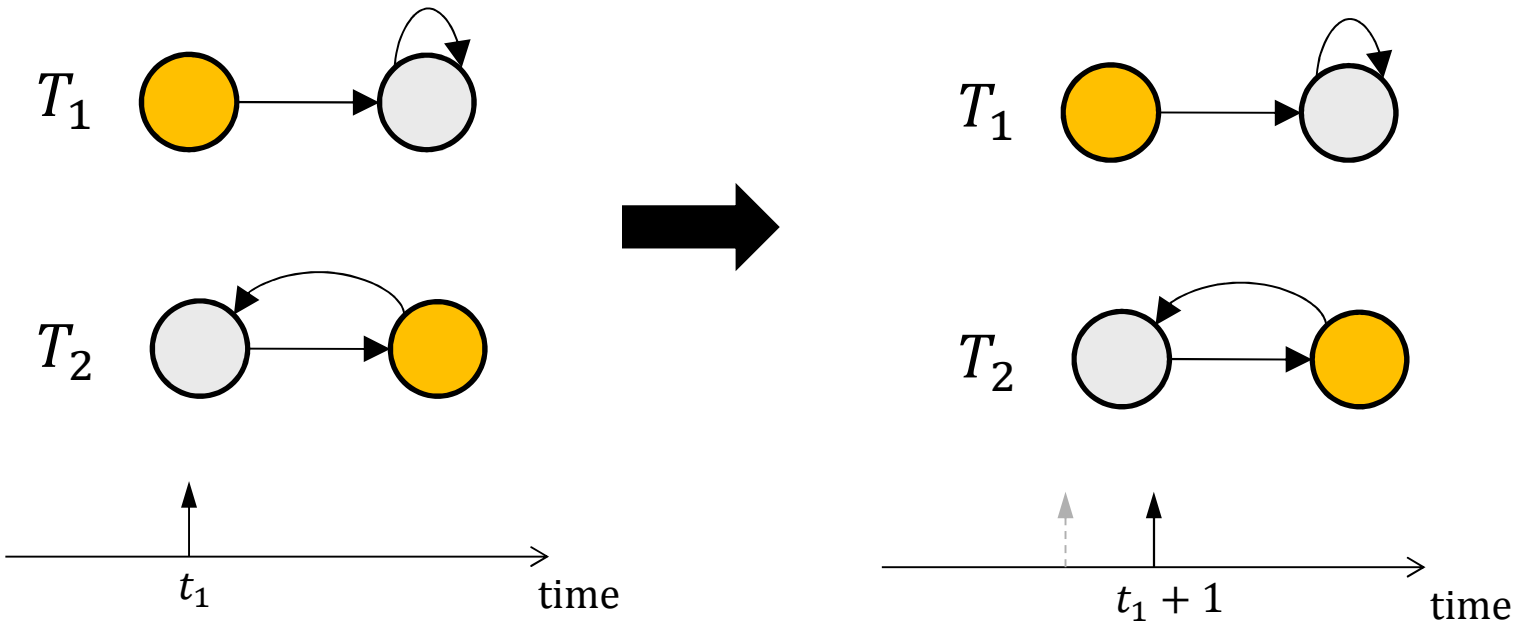
- Initial State: $(\bar{v}_0, \bar{\sigma}_0, \bar{c}_0)$
- Example
  - $\bar{v}_0 = \langle v_1, u_1 \rangle$: the starting jobs
  - $\bar{\sigma} = \langle \sigma_{1,0}, \sigma_{2,0} \rangle$: the initial valuation
  - $\bar{c} = \langle 0,0 \rangle$: clocks



Labeled Transition System

- Transition rules:
  - Delay transitions:

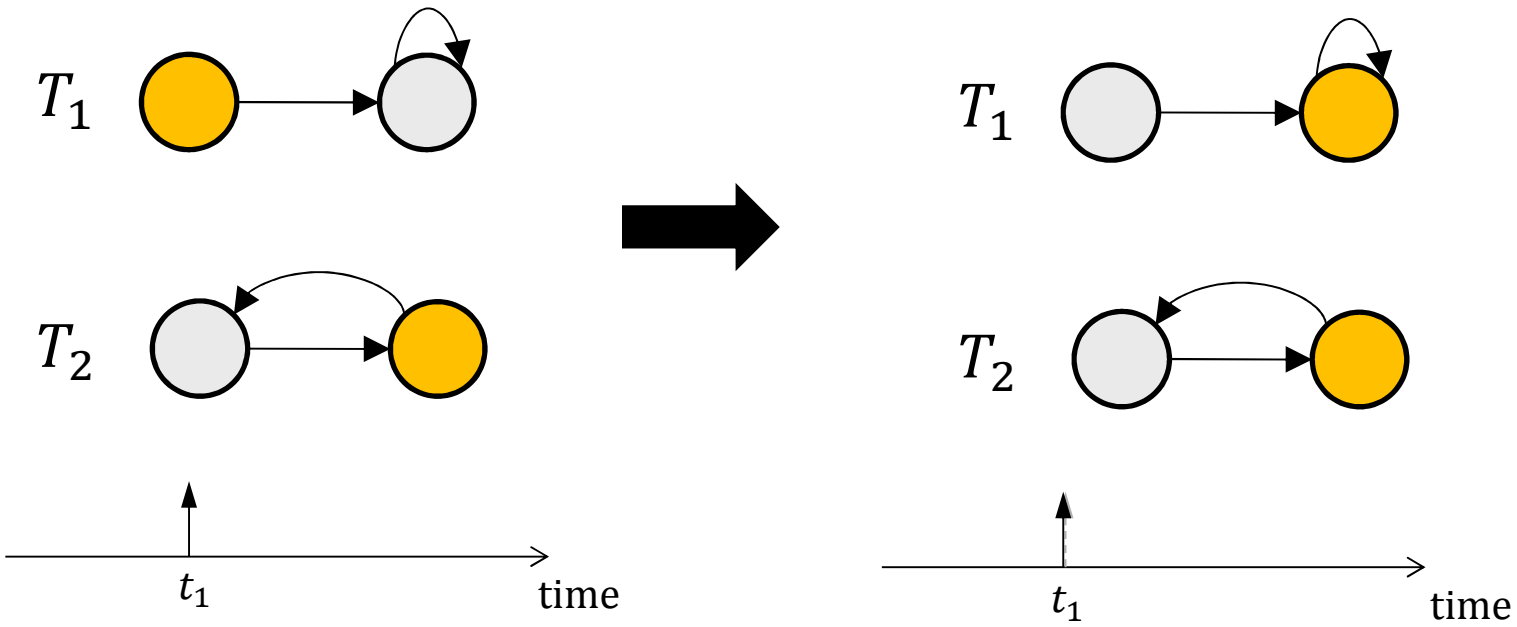
$$\langle c_1, \ldots, c_n \rangle \rightarrow \langle c_1 + 1, \ldots, c_n + 1 \rangle$$



Labeled Transition System

- Transition rules:
  - Release transitions:

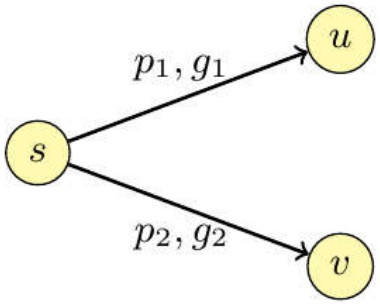
$$\langle v_1, \ldots, v_i, \ldots, v_n \rangle \rightarrow \langle v_1, \ldots, v_i', \ldots, v_n \rangle$$



UPPSALA UNIVERSITET

Code Generation

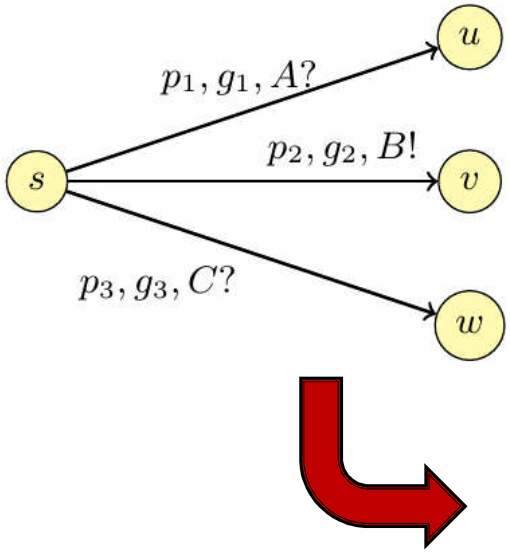
Code Generation: Branching



- Assumption: $p_1 \leq p_2$

```
1     -- After completion of the last released job
2     delay until p1;
3     if g1 then
4         next_state := u;
5         goto loop_start; -- Skipping the rest
6     end if;
7     delay until p2 − p1;
8     if g2 then
9         next_state := v;
10        goto loop_start;
11    end if;
```

Code Generation: Synchronization



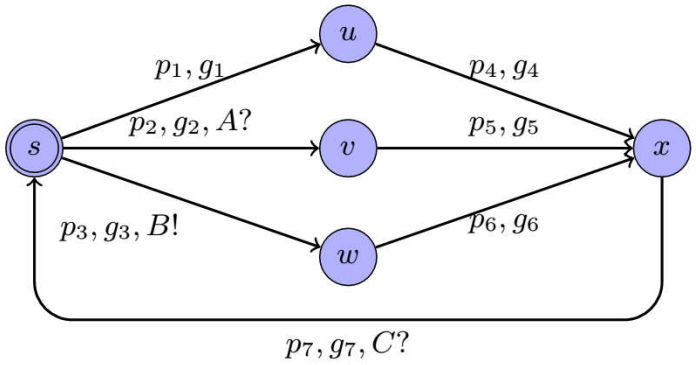
Assumption:

$$p_1 \leq p_2 \leq p_3$$

```
1   delay until p1;
2   select
3       when g1 =>
4       accept A;
5       next_state := u;
6       goto loop_start;
7   or
8       delay until p2 − p1;
9   end select;
10  if g2 then
11      Task_2.B;          -- Entry call to Task_2
12      next_state := v;
13      goto loop_start;
14  end if;
15  select
16      -- Repetition of the code appeared in Lines 3 to 6
17  or
18      delay until p3 − p2;
19  end select;
20  select     -- A selective accept
21      -- Repetition of the code appeared in  Lines 3 to 6
22  or
23      when g3 =>
24      accept C;
25      next_state := w;
26      goto loop_start;
27  end select;
```

Code Generation: Tasks

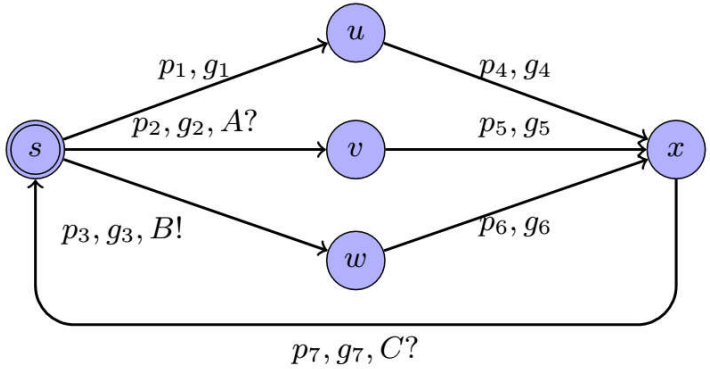
- Declaration




```
1  -- Context clauses and pragmas omitted
2  procedure Taskset_1 is
3      ----  Task declaration ----
4      task T1 is        -- A singleton task
5          pragma Priority(System.Priority'Last);
6          entry A;
7          entry C;
8      end T1;
```

Code Generation: Tasks

- Job Types

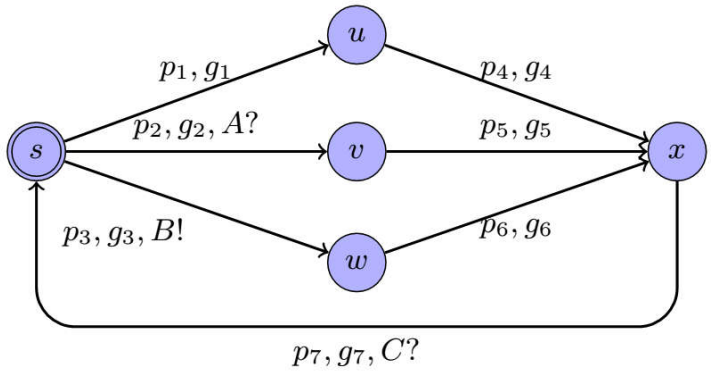


Running the job with the task priority

```
-- Procedures for the job types of T1:
procedure s_code is
begin
    Ada.Dynamic_Priorities.Set_Priority(T1_prio);
    -- The code for job type s goes here
    Ada.Dynamic_Priorities.Set_Priority(System.Priority'Last);
end s_code;
```

Code Generation: Tasks

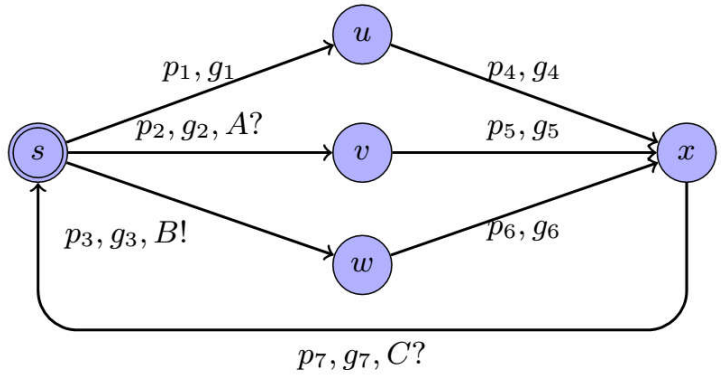
- Task body




```
begin
    Last_Release := Clock;
    loop
        <<T1_loop>>
        case Current_State is
            when s =>
                s_code;
                -- Determine the next job type
                -- Wait until the next release
            when u =>
                u_code;
                 ...
            when v =>
                u_code;
                 ...
            when w =>
                u_code;
                 ...
            when x =>
                x_code;
                ...
        end case;
    end loop;
```

- Infinite loop
- Implementing the graph structure

Task Body



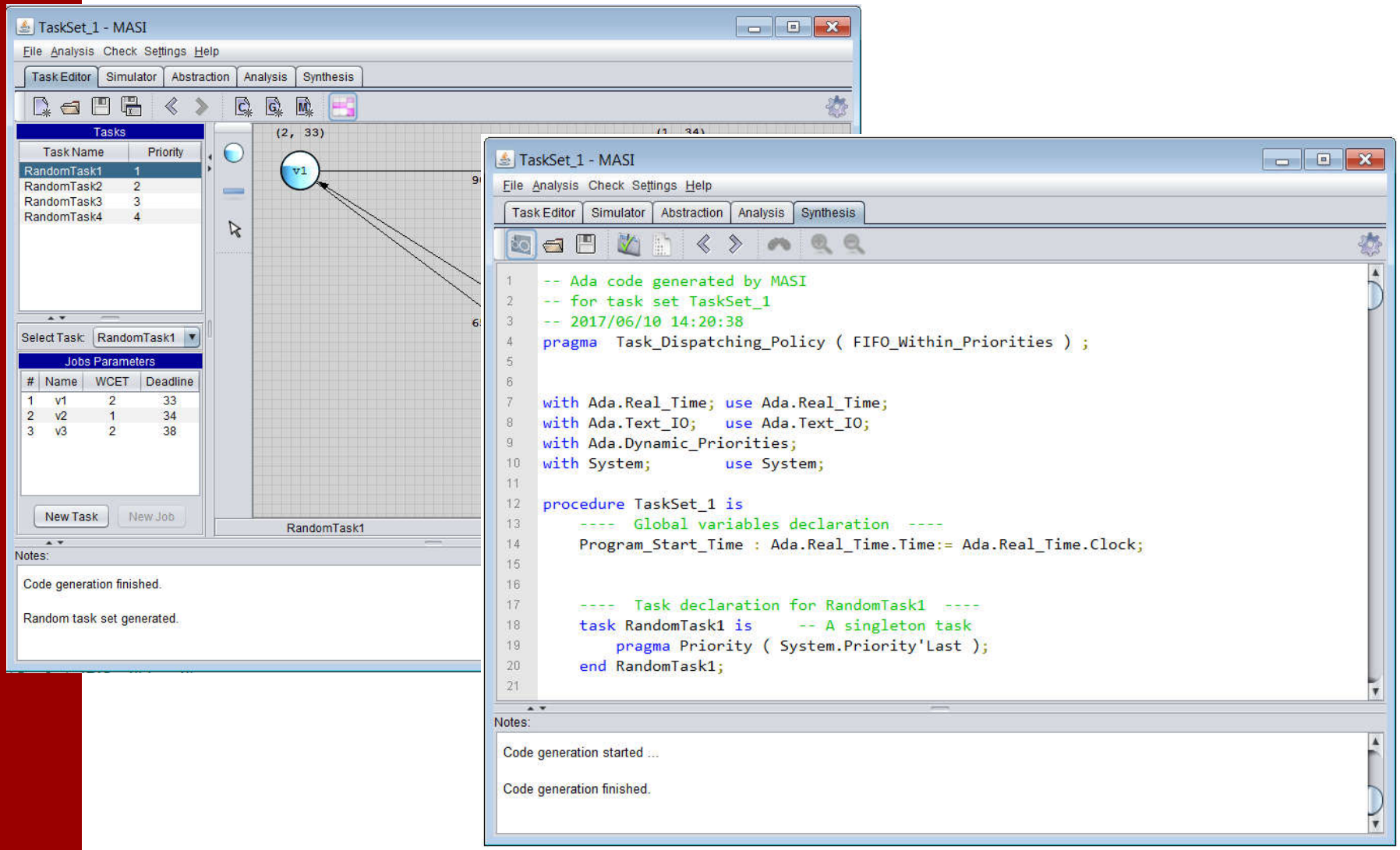

```
begin
    Last_Release := Clock;
    loop
        <<T1_loop>>
        case Current_State is
            when s =>
                s_code;
                delay until Last_Release + p1;
                if g1 then
                    Current_State := u;
                    Last_Release := Last_Release + p1;
                    goto T1_loop;
                end if;
                delay until Last_Release + p2 - p1;
                select
                    when g2 =>
                    accept A;
                    Last_Release := Clock;
                    Current_State := v;
                    goto T1_loop;
                or
                    delay until Last_Release + p3 - p2;
                end select;
                if g3 then
                    T2.B;           -- Entry call to task T2
                    Last_Release := Clock;
                    Current_State := w;
                    goto T1_loop;
                end if;
```

Edge $s \rightarrow u$

Edge $s \rightarrow v$

Edge $s \rightarrow w$

Tool Implementation



UPPSALA UNIVERSITET

An Executable Semantics for Synchronous Task Graphs: From SDRT to Ada

Morteza Mohaqeqi,
Jakaria Abdullah,
Wang Yi

Uppsala University, Sweden

morteza.mohaqeqi@it.uu.se

Ada europe 2017