A Hierarchical Architecture for Time- and Event-Triggered Real-Time Systems

Jorge Real, Sergio Sáez, Alfons Crespo
Universitat Politècnica de València
Spain

23rd International Conference on Reliable Software Technologies – Ada-Europe 2019
11-14 June, Warsaw, Poland

Presentation outline

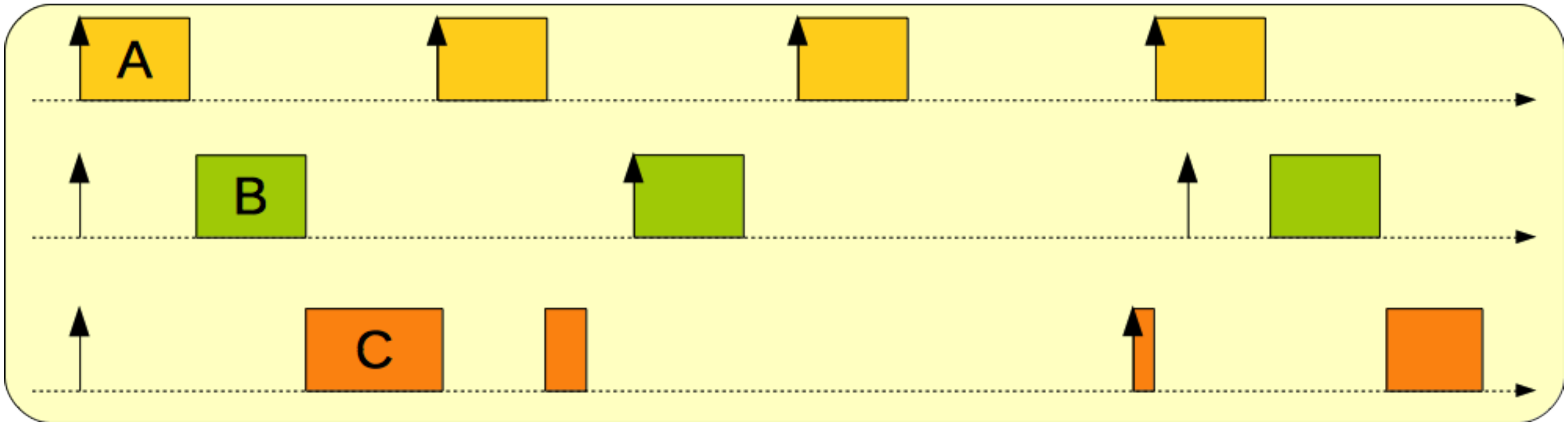
- Introduction
- System model
- Use examples – TT patterns
- Implementation
- A library of TT utilities
- Conclusion

Introduction

- Two major approaches for real-time scheduling
  - Time-Triggered (TT) – Static, table-driven plans
  - Event-Triggered (ET) – Priority schedulers, static/dynamic priorities

A TT plan with three tasks

Three fixed-priority, pre-emptively scheduled tasks

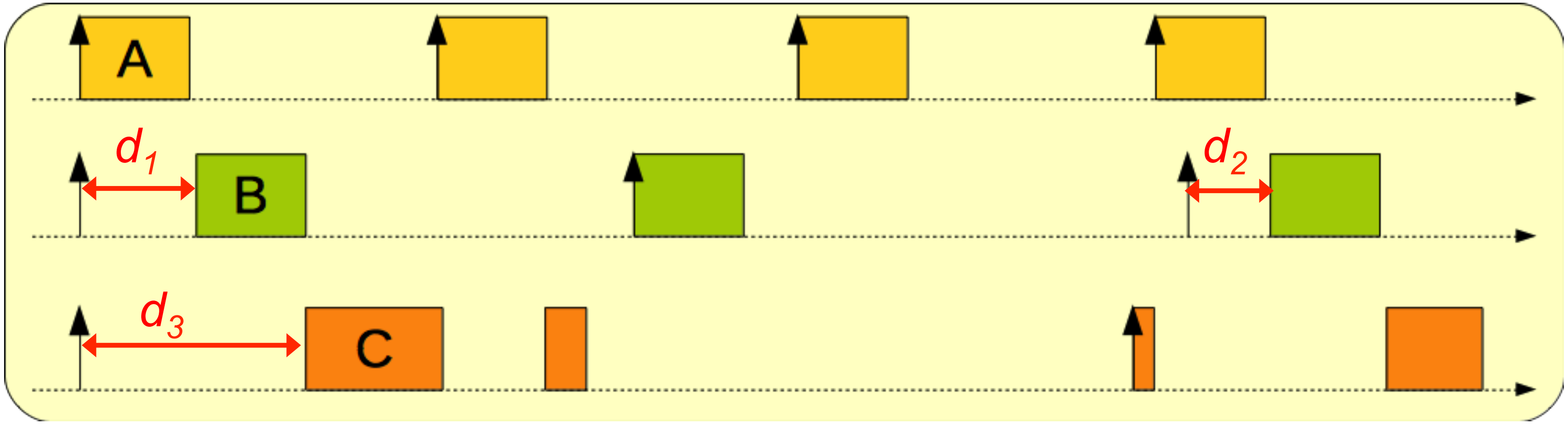


Introduction

- Two major approaches for real-time scheduling
  - Time-Triggered (TT) – Static, table-driven plans
  - Event-Triggered (ET) – Priority schedulers, static/dynamic priorities

A TT plan with three tasks

Three fixed-priority, pre-emptively scheduled tasks



Introduction

- Two major approaches for real-time scheduling
  - Time-Triggered (TT) – Static, table-driven plans
  - Event-Triggered (ET) – Priority schedulers, static/dynamic priorities

Table-driven	Priority-based
✦ All events need be known in advance – Building a schedule is complex ✦ Low responsiveness to events unrelated with time, difficult to accommodate long tasks	✓ Decoupling of functional and timing aspects – Eases system design ✓ Naturally accommodates sporadic tasks and long tasks
✓ Predictable – Tasks start at predetermined points in time ✓ Ideal for delay-sensitive systems	✦ Tasks can be variably delayed due to interference and resource sharing ✦ Major issue in control systems

System model

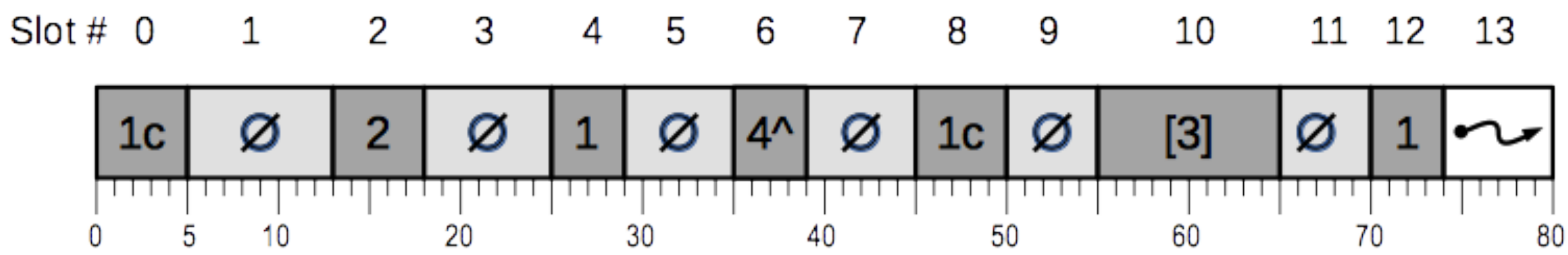
- On top of a priority scheduler
  - Use a highest-priority TT plan for subset of delay-sensitive tasks
    - Control and communication tasks
  - All other priorities for rest of tasks
    - HMI, logging, sporadics, *long* tasks...
- Best of TT and ET scheduling
  - Fewer tasks require simpler TT plans
  - Benefits of priority scheduler for rest of tasks

System model

- TT Plan is an ordered sequence of time slots
  - Each slot has a given duration
    - may have additional attributes
  - Each slot starts right after the end of previous slot
  - Cyclic plan
- TT scheduler
  - Acts at slot boundaries, depending on type of slot (next slide):
    - Check state of running task
    - Start one TT task
    - Other actions...
- Schedulability analysis:
  - TT plan – by construction
  - ET tasks – RTA, with TT plan regarded as flow of tasks with offsets

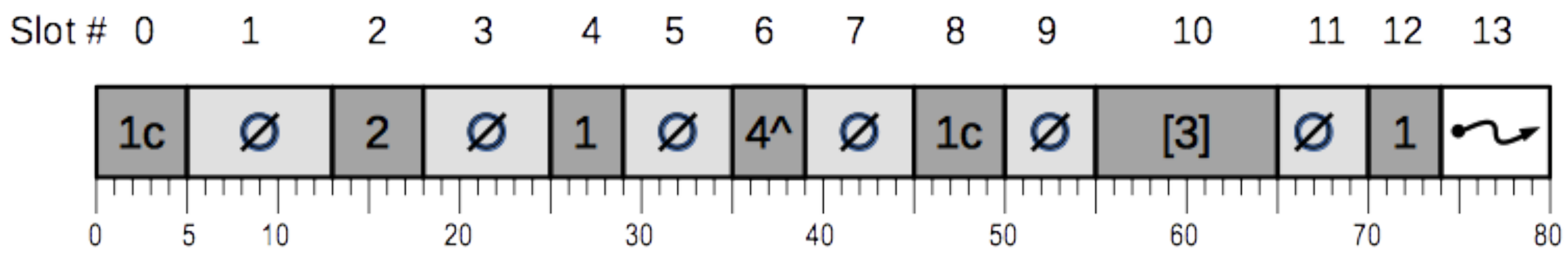
System model: Types of slots

- Empty slots ⊘
  - Define intervals available for ET tasks (no TT activity)
- Mode change slots ↝
  - Define times when it is possible to switch TT plan
- TT Work slots *j*, *jc*, *[j]*, *j^*
  - Several sub-types (next slide)
  - Involve (potential) execution of TT work, or sync points with the TT plan. Hence they all use a Work_ID, *j*



System model: TT work slots

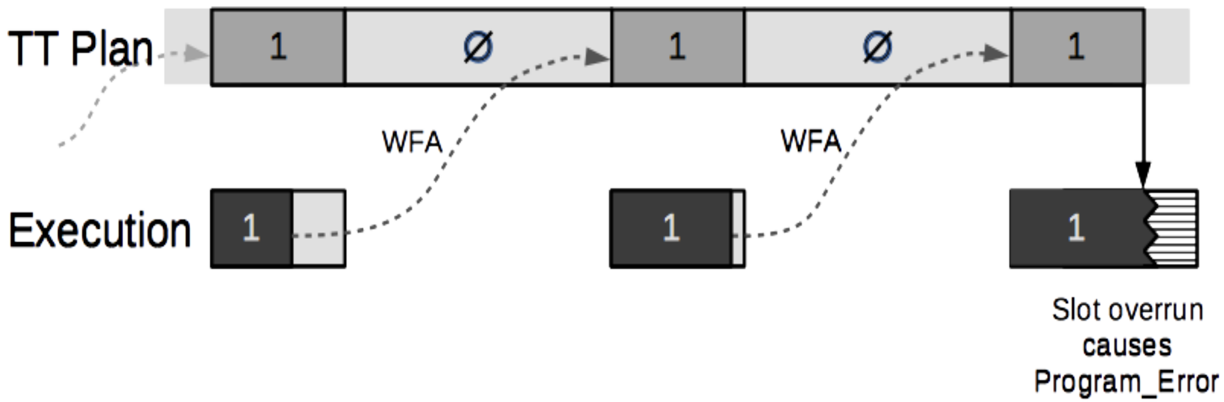
- Regular – 1, 2 – *Run TT task with id = j*
  - TT task must be waiting. Overrun check at end – PE if check fails
- Regular with Continuation – 1c – *Run TT task j, sliced*
  - Hold/Continue mechanism for long TT tasks
- Optional – [3] – *Run TT task n, optionally*
  - With overrun check if slot is taken
- Optional with Continuation – *Optionally, take sliced seq.*
- Sync slot – 4^ – Sync slots for non-TT tasks



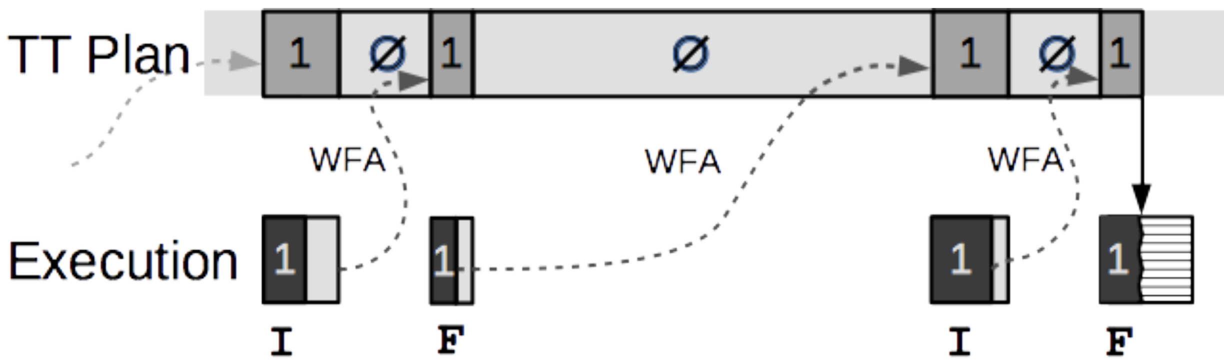
Use examples – TT Patterns

- With regular slots

```
-- Simple TT task
loop
  Wait_For_Activation (1);
  Do_My_Work;
end loop;
```



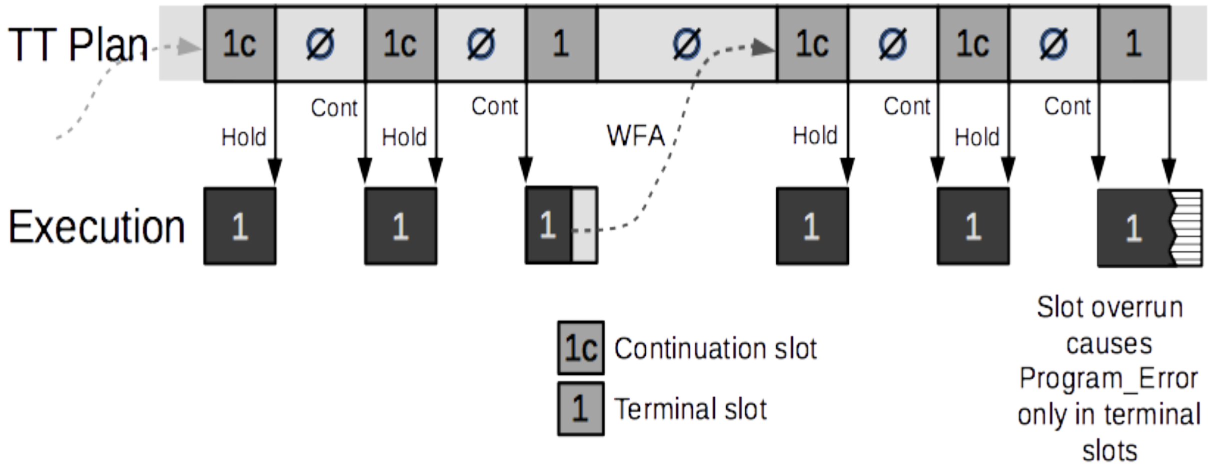

```
-- Initial-Final TT task I-F
loop
  Wait_For_Activation (1);
  Do_Initial;
  Wait_For_Activation (1);
  Do_Final;
end loop;
```



Use examples – TT Patterns

- With continuation slots

```
-- Sliced TT task
loop
  Wait_For_Activation (1);
  Do_My_Work_Sliced;
end loop;
```

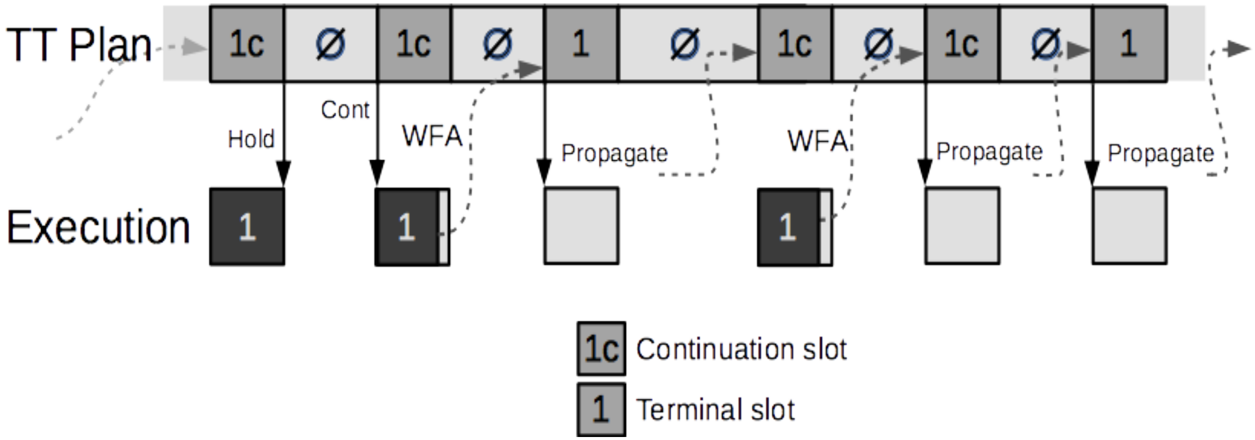


- Pattern looks like a Simple TT Task, but plan is different
  - One or more continuation slots, ending with a *terminal*, regular slot

Use examples – TT Patterns

- With continuation slots

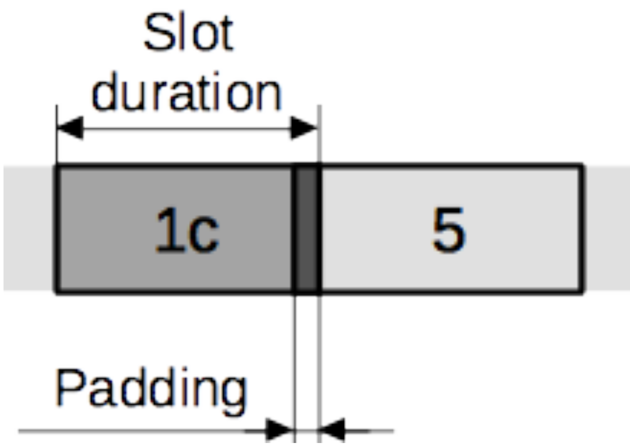
```
-- Sliced TT task
loop
  Wait_For_Activation (1);
  Do_My_Work_Sliced;
end loop;
```



- Early completion of sliced TT task

Use examples – TT Patterns

- With continuation slots
  - Hold/Continue take care of ongoing protected actions
    - A Padding time may be specified for continuation slots, to absorb the time of closing an ongoing PA without delaying the next slot
    - Hold is applied at slot_end – padding



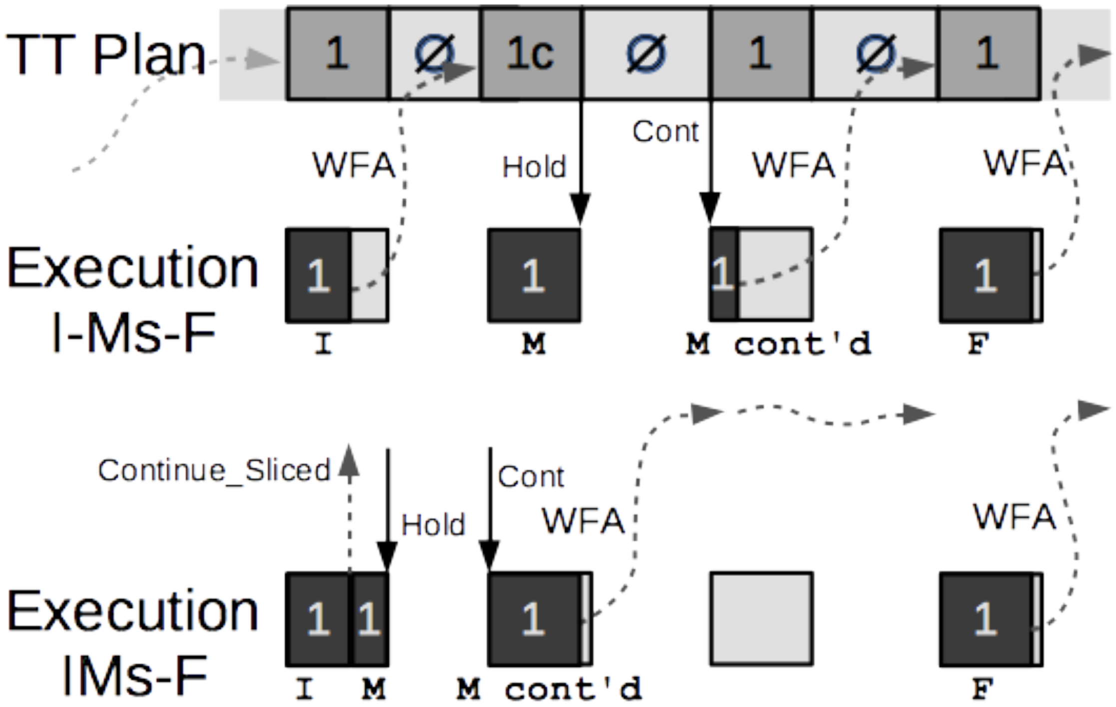
Use examples – TT Patterns

- More sliced patterns – motivation for `Continue_Sliced`

I-Ms-F and IMs-F

```
loop   -- I-Ms-F Pattern
   Wait_For_Activation (1);
   Do_Initial_Part;
   Wait_For_Activation (1);
   Do_Mandatory_Sliced;
   Wait_For_Activation (1);
   Do_Final_Part;
end loop;
```

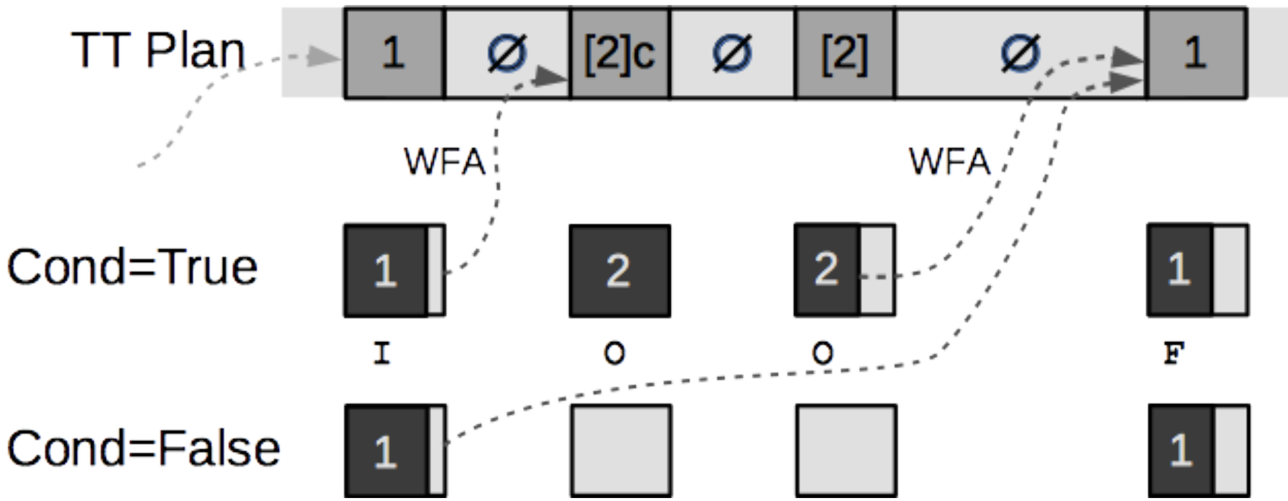
```
loop   -- IMs-F Pattern
   Wait_For_Activation (1);
   Do_Initial_Part;
   Continue_Sliced;
   Do_Mandatory_Sliced;
   Wait_For_Activation (1);
   Do_Final_Part;
end loop;
```



Use examples – TT Patterns

- With optional slots – no-show is not an error
- Optional sequence
  - Take entire sequence or no slot of the sequence at all
- For example
  - Initial - Optional Sliced – Final

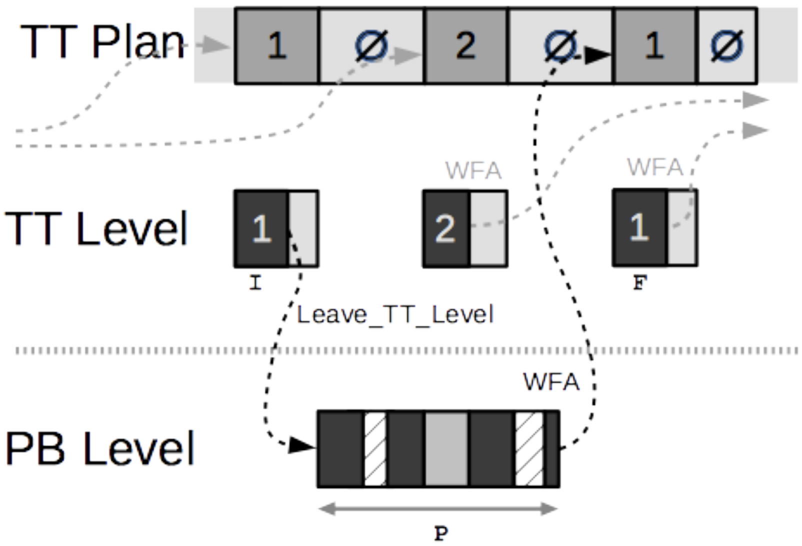
```
loop
  Wait_For_Activation (1);
  Do_Initial_Part;
  if Condition then
    Wait_For_Activation (2);
    Do_Optional_Sliced;
  end if;
end loop;
```



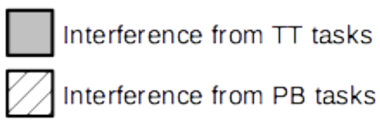
Use examples – TT Patterns

- Using non-TT parts – motivation for `Leave_TT_Level`

Initial-Priority_Based-Final




```
loop
   Wait_For_Activation (1);
   Do_Initial_Part;
   Leave_TT_Level;
   Do_Priority_Based_Part;
   Wait_For_Activation (1);
   Do_Final_Part;
end loop;
```

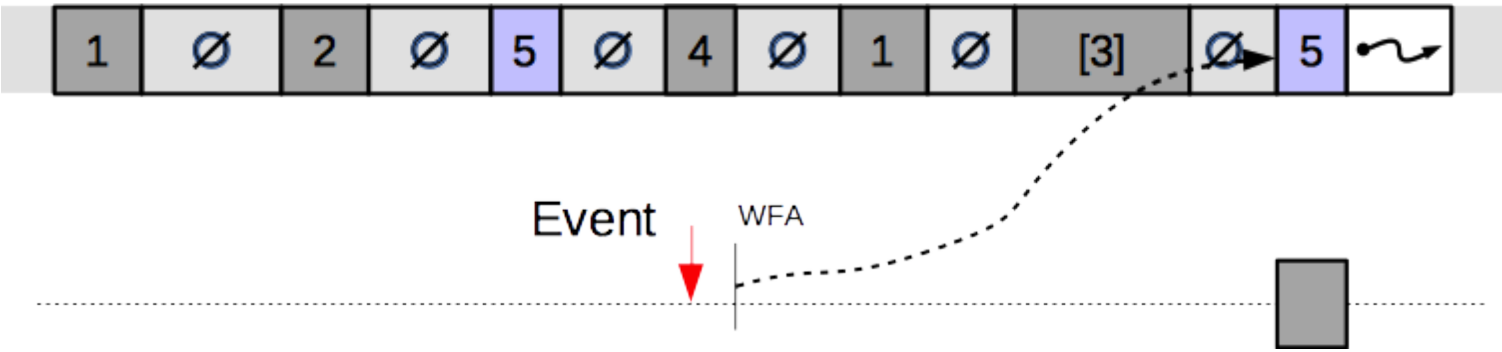


Dynamic priorities, although in a restricted manner

- Changes only from self task
- Changes only between base and TT priorities
- Conceptually, like a ceiling inherited when running the TT parts

Use examples – TT Patterns

- ET task using optional slots and `Leave_TT_Level`




```
loop
  Wait_For_Transmission_Phase_Ready;
  Wait_For_Activation (5);
  Transmit;
  Leave_TT_Level;
end loop;
```

- Lower priority transmissions: use Sync slots from ET task

Implementation

- TT Scheduler provided as an *extension* Ada library
  - XAda.Dispatching.TTS
  - Generic with two parameters
    - Number_Of _Work_Ids
    - Number_Of_Sync_Ids
  - Subprograms
    - Set_Plan (TTP: Time_Triggered_Plan_Access);
    - Wait_For_Activation (Work_Id : TT_Work_Id; When: out Time);
    - Wait_For_Sync (Sync_Id: TT_Sync_Id; When : out Time);
    - Continue_Sliced;
    - Leave_TT_Level;
    - Get_Current_Slot_Info return Any_Slot_Type_Access
    - Get_Last_Plan_Release return Time;

Implementation

- Types
  - Time_Slot is abstract, tagged record: Slot_Duration
  - Empty_Slot/Mode_Change_Slot is new Time_Slot with null record
  - Work_Slot is new Time_Slot with Work_Id, ...
  - Optional_Slot is new Work_Slot
  - Time_Triggered_Plan is array (Natural range <>) of Time_Slot_Access
- Scheduler
  - In Ravenscar
  - TT plan events handled as timing events
    - New_Slot timing event at slot boundaries
    - Specific timing event for Hold when Padding > 0

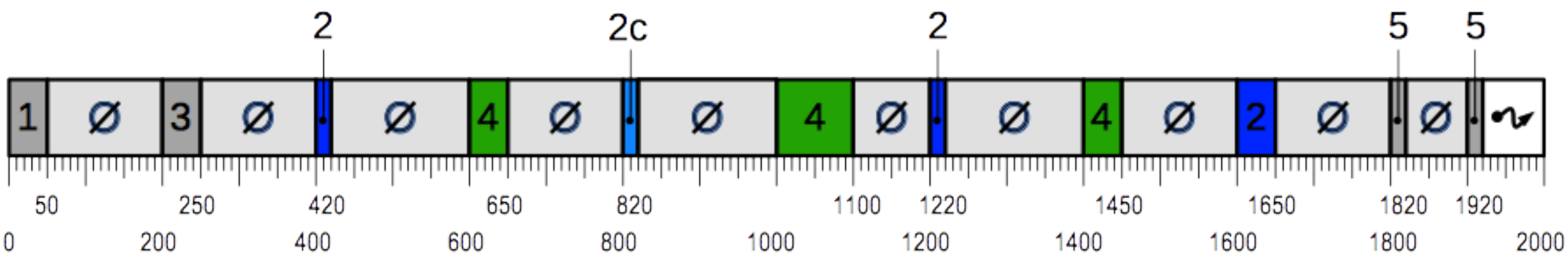
Utilities library

- Hides use of XAda.Dispatching.TTS
- Generic package with same generic parameters
- Slot constructor to ease plan definition
- Abstract types for patterns, extensible with task states
    - Including task's local variables and subprograms

- Code available on GitHub project TTS-Ravenscar-Runtime
    - Latest additions still to be merged to master branch – July
- Test board: STM32F4 Discovery

Utilities library

```
TT_Plan : aliased Time_Triggered_Plan :=
  ( A_TT_Slot (Regular, 50, 1),         A_TT_Slot (Empty, 150),           -- Single slot for 1st seq. start
    A_TT_Slot (Regular, 50, 3),         A_TT_Slot (Empty, 150),           -- Single slot for 2nd seq. start
    A_TT_Slot (Regular, 20, 2),         A_TT_Slot (Empty, 180),           -- Seq. 1, IMs part
    A_TT_Slot (Regular, 50, 4),         A_TT_Slot (Empty, 150),           -- Seq. 2, IMs part
    A_TT_Slot (Continuation, 20, 2),    A_TT_Slot (Empty, 180),           -- Seq. 1, continuation of Ms part
    A_TT_Slot (Terminal, 100, 4),       A_TT_Slot (Empty, 100),           -- Seq. 2, terminal of Ms part
    A_TT_Slot (Terminal, 20, 2),        A_TT_Slot (Empty, 180),           -- Seq. 1, terminal of Ms part
    A_TT_Slot (Regular, 50, 4),         A_TT_Slot (Empty, 150),           -- Seq. 2, F part
    A_TT_Slot (Regular, 50, 2),         A_TT_Slot (Empty, 150),           -- Seq. 1, F part
    A_TT_Slot (Regular, 20, 5),         A_TT_Slot (Empty, 80),            -- I part of end of plan
    A_TT_Slot (Regular, 20, 5),         A_TT_Slot (Mode_Change, 80) );    -- F part of end of plan
```



Utilities library

- TT task state

```
type Initial_Mandatory_Final_Task_State is abstract tagged null record;
procedure Initialize (S : in out Initial_Mandatory_Final_Task_State) is abstract;
procedure Initial_Code (S : in out Initial_Mandatory_Final_Task_State) is abstract;
procedure Mandatory_Code (S : in out Initial_Mandatory_Final_Task_State) is abstract;
procedure Final_Code (S : in out Initial_Mandatory_Final_Task_State) is abstract;

type Any_Initial_Mandatory_Final_Task_State is access all Initial_Mandatory_Final_Task_State'Class;
```

- TT task pattern

```
task type InitialMandatorySliced_Final_TT_Task
  (Work_Id      : TT_Work_Id;
   Task_State   : Any_Initial_Mandatory_Final_Task_State;
   Synced_Init : Boolean)
with Priority => Priority'Last – 1;
```

Utilities library

- Instantiation of I-M-F pattern

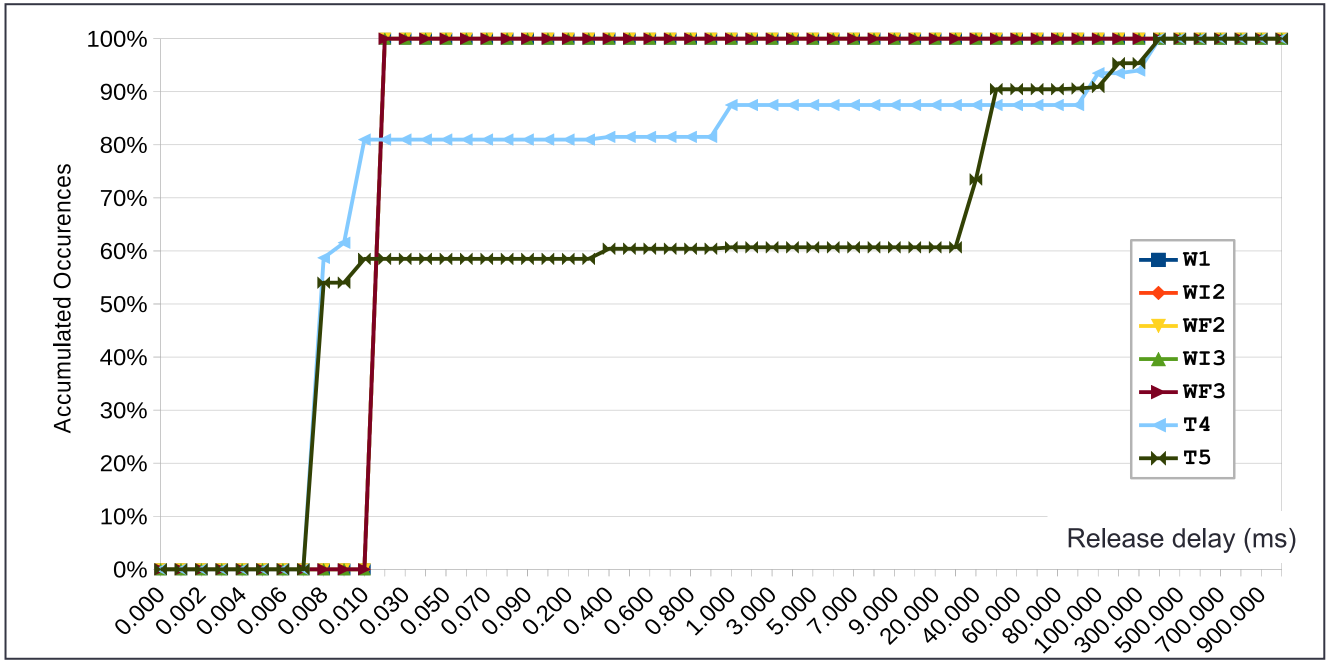
```
type First_IMF_Task is new Initial_Mandatory_Final_Task_State with
   record
      Counter : Natural;
   end record;
procedure Initialize (S : in out First_IMF_Task) is null;
procedure Initial_Code (S : in out First_IMF_Task);
procedure Mandatory_Code (S : in out First_IMF_Task);
procedure Final_Code (S : in out First_IMF_Task);

procedure Initial_Code (S : in out First_IMF_Task) is
begin
   S.Counter := 0;
end Initial_Code;

--  Bodies of Mandatory and Final parts
...
```

Conclusion

- Additions to the 2019 version
  - A well-defined model for sliced TT tasks and its implementation at the runtime level
    - PO-safe Hold and Continue operations
  - Synchronisation between TT and ET workloads with Sync slots
    - An ET task can be delayed until a particular point in the TT plan
  - Facilities for obtaining information of running plan
    - Plan start
    - Cycle start
  - Facilities for obtaining information about the current slot
    - E.g., when will my next slot occur? (in the cooking)
  - An OO re-implementation of
    - TT plans – extensible slots
    - TT patterns – extensible patterns

Conclusion

- With additions included, similar timing results (-O3)
  - TT scheduler overhead in the region of 20 μs
  - This is the only release delay incurred by TT tasks
  - So consistent that it can be removed by slightly advancing slot switch events



Conclusion

- With additions included, similar timing results (-O3)
  - TT scheduler overhead in the region of 20 μs
  - This is the only release delay incurred by TT tasks
  - So consistent that it can be removed by slightly advancing slot switch events