Workload Models (for Timing Analysis)

Can P finish its execution within D sec’s?
Embedded Systems

Timing Analysis

- What is the maximal delay at each component?
- What is the maximal end-to-end delay?
Timing Analysis

Sequential Case (WCET Analysis)

Concurrent Case (Response Time Analysis)

Non-deterministic releases

WCET
Timing Analysis

Sequential Case (WCET Analysis)

- Assume the WCET of each task is given (resource budget)
- How to estimate the Worst-Case Response Time of a task?

Concurrent Case (Response Time Analysis)

Non-deterministic releases

Assume the WCET of each task is given (resource budget) How to estimate the Worst-Case Response Time of a task?

Wilhelm et al
Precision >> 95%

[aiT tool from AbsInt]
Modeling for (System-Level) Timing Analysis

- The event arrival patterns (e.g. using Layland & Liu task model)
- Synchronization between components
- Resource arbitration, protocols and scheduling algorithms
- Resource demands or budget e.g. the WCET
- Timing constraints e.g. deadlines
Expressiveness vs. Analysis Efficiency

Feasibility test

Expressiveness

Efficient

difficult

Precise

Approximate

Strongly (co)NP-hard
Pseudo-Polynomial

(intractable)

(tractable)
Modeling a system for analysis

• System = a set of Tasks

  Task 1  Task 2  Task n

• Tasks releasing jobs, $J = (e, d)$ -- basic unit of workload
  – Release time $r$
  – Worst-case execution time $e$
  – Deadline $d$

• Job release patterns:
  – periodicity, branching structures, loop ... ...

Feasibility/Schedulability: Can we check s.t. all jobs meet their deadlines?
Typical task code structure:

... 

loop

// Execute some function for, e.g.,
// up to 11ms
// (obtained via WCET analysis)

delay until Previous_Period + 50ms;
end loop;
...

**Periodic Task**

- Execution time $e = 11ms$ of each *job*
- Period $p = 50ms$
- Implicit deadline $d = p$ for each job
The Liu and Layland Task Model

- Each periodic task defined via 2 numbers:
  - Job WCET $e$
  - Minimum inter-release delay $p$ (implicit deadline)

![Diagram showing periodic tasks]

- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, p_i)$
- How to schedule these tasks?
  - Static/fixed priority scheduling (e.g. RM)
  - Dynamic priority scheduling (e.g. EDF)
- How to analyze schedulability?
The Liu and Layland Task Model: Schedulability

- **Static priority scheduling**
  - Response time analysis for all tasks
    \[ R_i = e_i + \sum_{j \in hp(i)} \left\lfloor \frac{R_i}{p_j} \right\rfloor \cdot e_j \]
    
    - Compute iteratively; compare with \( p_i \)
    - Precise test (sufficient and necessary)
  - Utilization test
    - Define \( U_i := \frac{e_i}{p_i} \) as utilization of task \( \tau_i \)
    - \( \tau \) schedulable if \( \sum_i U_i \leq n(2^{1/n} - 1) \)
    - Only sufficient test

- **Dynamic priority scheduling**
  - Just focus on EDF, it’s optimal
  - Simple and precise test: \( \sum_i U_i \leq 1 \)
Hierarchy of Models

Feasibility test

difficult

expressiveness

L&L

efficient

Strongly (co)NP-hard
Pseudo-Polynomial
The Sporadic Task Model

- What if deadlines ≠ periods?
- Each *sporadic* task defined via 3 numbers:
  - Job WCET $e$
  - Relative deadline $d$
  - Minimum inter-release delay $p$

![Diagram showing sporadic tasks](Diagram showing sporadic tasks)

- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, d_i, p_i)$
- Scheduling?
  - Static/fixed priority scheduling: DM *optimal*
  - Dynamic priority scheduling: EDF *optimal*
  - But how to *analyze* schedulability?
The Sporadic Task Model: Schedulability

- **Static priority scheduling**
  - As before, compute response times
  - (They are independent of deadlines!)

- **Dynamic priority scheduling**, i.e., EDF
  - Simple test $\sum_i U_i \leq 1$ doesn’t work anymore (why?)
  - Introduce *demand bound functions*
Hierarchy of Models

Feasibility test

difficult

three integers
sporadic

explicit deadline

L&L

implicit deadline
two integers

efficient

low

high

Expressiveness
The General Multiframe (GMF) Task Model

- Code is not always periodic:

```plaintext
loop
    // Execute some function
    delay until Period_Start + 50ms;
    // Execute another function
    delay until Prev_Function + 30ms;
    // Execute yet another function
    delay until Prev_Function + 70ms;
end loop;
```

- Task is split in *frames*, each with own
  - Execution time $e^{(i)}$
  - Inter-release separation $p^{(i)}$
  - Deadline $d^{(i)}$ for the job
The General Multiframe (GMF) Task Model (cont.)

- Tasks *cycle* through job types
  - Vector for WCET \((e^{(1)}, \ldots, e^{(n)})\)
  - Vector for deadlines \((d^{(1)}, \ldots, d^{(n)})\)
  - Vector for minimum inter-release delays \((p^{(1)}, \ldots, p^{(n)})\)
Hierarchy of Models

Feasibility test

Efficient

difficult

Expressiveness

High

Low

cycle graph

gmf

different job types

sporadic

explicit deadline

two integers

l&l

implicit deadline

three integers
The Recurring Branching (RB) Task Model

What about *branches*?

```plaintext
loop
    // Execute some function
    delay until Period_Start + 50ms;
    if (condition) then {
        // Execute another function
        delay until Prev_Function + 30ms;
    } else {
        // Execute yet another function
        delay until Prev_Function + 70ms;
    }
end loop;
```

Task is a tree
- Vertices represent jobs
- Edges represent control flow and delays
- Restarted once a leaf is reached
The Recurring Branching (RB) Task Model (cont.)

- Introduces branching structures
- A tree for each task
  - Vertices $v$: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges $(u, v)$: minimum inter-release delays $p(u, v)$
  - General period parameter $P$

![Diagram of a tree structure with labels on edges and vertices, indicating job types and delays. The period $P = 57$.]
Restrictions of DAG/RRT model

- Tasks are still \textit{recurrent}
  - Always revisit source $J_1$
  - \textit{No cycles allowed!}

- Consequences:
  - No \textit{local loops}
  - Not \textit{compositionnal} (for modes etc.)
Restrictions of DAG/RRT model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)
Restrictions of DAG/RRT model

- Tasks are still *recurrent*
  - Always revisit source \( J_1 \)
  - *No cycles allowed!*

- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)
Hierarchy of Models

difficult

Feasibility test

efficient

high

Expressiveness

tree

branching

cycle graph

different job types

three integers

sporadic

explicit deadline

two integers

L&L

implicit deadline
Generalize Further?

- What we can model now:
  - Periodic behavior
  - Arbitrary deadlines
  - Multiple frames
  - Branching behavior

- Still not possible:
  - Local loops
  - Local modes
  - ...

```plaintext
loop
  f1();
  delay ...
  if (cond) then {
    while (cond2) {
      f2();
      delay ...
    }
  } else {
    f3();
    delay ...
  }
end loop;
```
The Digraph Real-Time Model (DRT)

- Pairs on nodes are the WCET and deadline on the task code e.g. A has WCET 2 and relative deadline 4
- Numbers on edges are the minimum inter-release delays

In Ada Tasking:

- Procedure PA "release A"
  - Delay(2)
  - PC

- Procedure PB "release B"
  - Delay(25)
  - PA

- Procedure PC "release C"
  - If "condition"
    - then Delay(10); PA
    - else Delay(11); PB

The WCET, deadlines and release delays should be ensured by the Ada run-time system

[Stigge et al, RTAS 2011]
Hierarchy of Models

Feasibility test

difficult

efficient

low

Expressiveness

high

arbitrary graph

tree

cycle graph

branching, loops, ...

branching

different job types

explicit deadline

implicit deadline

three integers

two integers

sporadic

L&L
Path $\pi = (v_4)$
DRT: Semantics

Path $\pi = (v_4, v_2)$
DRT: Semantics

Path $\pi = (v_4, v_2, v_3)$
DRT: Semantics

Path $\pi = (v_4)$
DRT: Semantics

Path $\pi = (v_4, v_2)$
DRT: Semantics

Path $\pi = (v_4, v_2, v_3)$
A system model = a set of DRT’s modeling the components

- How to check the feasibility?
- How to check the schedulability for a given scheduler?
DBF (workload) of a DRT

Demand Bounds Function (dbf)

Time window
A system model = a set of DRT’s modeling the components

The system workload:
A system model = a set of DRT’s modeling the components

The system workload:

If the “blue” workload curve is always below the “red” supply Up to a bound D, the system is “feasible”
Ideas for feasibility analysis

- Characterize the system workload ...
- If the worst-case workload is over 100%, it is over-loaded, implying deadline miss

Units of work a CPU can compute over time (100%)

Workload

dbf

Time
Of course, if the **BLUE line** is always below the **RED**, the system should work well without deadline miss!

How to check this?

Units of work a CPU can compute over time (100 %)

Workload

---

**dbf**

**Time**
Here is the intuition why “Pseudo-P”

If the utilization (long-term rates of DRT’s) of a system is bounded by a constant $U < 1$, any deadline miss, if exists, must appear before a pseudo-polynomial upper bound:
How to characterize the workload of a system?

Demand Bound Function
The Demand Bound Function

- General tool/technique for EDF schedulability analysis: \( \text{dbf}(t) \)
- Intuition:
  - Given a time interval length \( t \)
  - \( \text{dbf}(t) \) bounds the demand for processor time within any \( t \) interval

![Diagram showing the demand bound function with intervals and demand values]

\[ \text{dbf}(t) = 8 + 8 \]
Example: L&L tasks \((e_i, p_i)\)

\[
\text{dbf}_{\tau_i}(t) = \left[ \frac{t}{p_i} \right] \cdot e_i
\]
Example: Sporadic tasks: \((e_i, d_i, p_i)\)

\[
dbf(t) = \sum_{T_i \in \tau} e_i \cdot \max \left\{ 0, \left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1 \right\}
\]
DRT: Demand bound over a Path

Path $\pi = (v_4)$

Demand bound: (5, 10)
DRT: Demand bound over a Path

Path $\pi = (v_4, v_2)$
DRT: Demand bound over a Path

Path $\pi = (v_4, v_2, v_3)$

Demand bound: (5,10)  Demand bound: (6,28)  Demand bound: (9,43)
Feasibility Test Using \( \text{dbf}(t) \)

**Theorem**

A task system \( \tau \) is preemptive uniprocessor feasible iff:

\[
\forall t \geq 0 : \sum_{T \in \tau} \text{dbf}_T(t) \leq t
\]
Feasibility Test Using dbf()
Calculating the Bound

dbf(t) is bounded by $C_{\text{max}} + t \cdot U$

- **Linear bound for dbf(t)**
  - Slope: Less than 1
  - Intersection with $t$ gives bound $D$
  - Check only up to $D$

$C_{\text{max}}$ = sum of WCETs for all jobs
$U$ = “the long-term utilization”
(note: $U < 1$)
Calculating the Bound

\[ \sum \text{dbf}_T(t) \]

- Linear bound for \( \text{dbf}(t) \)
  - Slope: Less than 1
- Intersection with \( t \) gives bound \( D \)
- Check only up to \( D \)

\[ D = \frac{C_{\text{max}}}{1 - U} \]

“Most dense” cycle

\[ \text{dbf}(t) \leq t \ast U + C_{\text{max}} \]
Calculating the workload: demand pairs

Recall Task Model:

For each path, we have:
1. An execution demand $e$
2. A deadline $d$ for this demand

Call $\langle e, d \rangle$ a demand pair

Exec. demand:
$$e = 1 + 5 + 2 = 8$$

Deadline:
$$d = 11 + 10 + 10 = 31$$

Demand pair: $\langle 8, 31 \rangle$
Calculating the workload: demand triples (path abstraction)

- Extend $\langle e, d \rangle$ with end vertex $v$
- Call $\langle e, d, v \rangle$ a demand triple
- Allows extensions to create new triples

Demand triple: $\langle 8, 31, J_5 \rangle$
Calculating the workload: demand triples (path abstraction)

- Extend \( \langle e, d \rangle \) with end vertex \( v \)
- Call \( \langle e, d, v \rangle \) a **demand triple**
- Allows **extensions** to create new triples

![Graph with demand triples and extensions](image)

Demand triple: \( \langle 8, 31, J_5 \rangle \)
New demand triple: \( \langle 10, 41, J_1 \rangle \)
Iterative Procedure

- Start with just the vertices (0-paths)
- Then, at each iteration:
  1. **Extend** current demand triples
  2. Optimization: Discard non-critical triples on the way
- Finally, $dBf$ is maximum
Iterative Procedure

- Start with just the vertices (0-paths)
- Then, at each iteration:
  1. **Extend** current demand triples
  2. Optimization: Discard non-critical triples on the way
- Finally: $dbf_T$ is maximum

The number of demand triples is bounded by $D^2 * D^2 * \#\text{vertices}$
Hierarchy of Models

[Stigge et al, RTAS 2011]
Iterative Procedure

- Create all demand triples up to $D$:
  1. Start with all 0-paths, i.e., $\langle e(v), d(v), v \rangle$ for all vertices $v$
  2. Pick some stored demand triple $\langle e, d, u \rangle$
  3. **Create new demand triple**:
     - Choose successor vertex $v$ of $u$
     - $e' = e + e(v)$
     - $d' = d - d(u) + p(u, v) + d(v)$
     - $\langle e', d', v \rangle$ is new demand triple!
  4. Store $\langle e', d', v \rangle$ if
     - not stored yet, and
     - $d' \leq D$
  5. Repeat from 2 until no change

- **Efficient** procedure!
  - Note: Actual paths never stored
  - Optimizations: Discard non-critical triples along the way
DRT Schedulability: Summary

- Schedulability test for EDF, based on dbf
- First, compute utilization for all tasks
  - Based on most dense cycles in graphs
- Derive bound $D$
- Compute dbf($t$) for all $t \leq D$
  - Uses iterative procedure with demand triples
  - Path abstraction to reduce complexity
- If $t \leq D$ with dbf($t$) $> t$ found: $\tau$ unschedulable
- Else: $\tau$ schedulable
Evaluation: Runtime vs. Utilization

Setting:
- Randomly generated task sets
- 1-30 tasks, 5-10 vertices per task, branching degree 1-3, ...
• How about “synchronization among tasks”?
• How about “general constraints on job releases”?
• How about “static priority scheduling”?
Hierarchy of Models

Stigge/Wang, ECRTS 2012
## Summary

<table>
<thead>
<tr>
<th>Models</th>
<th>Analysis Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility i.e. EDF-Schedulability</td>
</tr>
<tr>
<td></td>
<td>Static-priority Schedulability</td>
</tr>
<tr>
<td>General graphs (Di-graph)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td></td>
<td>Strongly coNP-complete</td>
</tr>
<tr>
<td>Trees/DAGs</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td></td>
<td>Strongly coNP-complete</td>
</tr>
<tr>
<td>Cyclic graphs (GMF)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td></td>
<td>Strongly coNP-complete</td>
</tr>
<tr>
<td>Sporadic (L&amp;L, deadline≠period)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td></td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>L&amp;L(sporadic, deadline=period)</td>
<td>Linear</td>
</tr>
<tr>
<td></td>
<td>Pseudo-P</td>
</tr>
</tbody>
</table>

For systems with utilization bounded by a constant less than 1 (or below 100%) **RTSS 2015 Best Paper**

Otherwise Strongly coNP-complete **ECRTS 2015 Best Paper**

!! The problem open for 30 years, theoretically interesting !!

[Ekberg and Wang]

What can we do?

[ECRTS 2012]
• Static priority schedulability is **Strongly NP-hard** for all models except L&L

• What to do?
Combinatorial Refinement
solving “Combinatorial Problems”
(for timing analysis, it works very well!)

[Stigge PhD thesis
ECRTS 2014
TACAS 2015]
A system model = a set of DRT’s modeling the components

This works perfectly for feasibility checking: the global worst case can be constructed from the local worst cases

The system workload:
A system model = a set of DRT’s modeling the components

In general, each component may have a set of behaviors e.g. Paths or traces
A system model = a set of DRT's modeling the components

Often, we have to check some property guaranteed by all the combinations of individual local behaviors and thus may have to enumerate ... (combinatorial explosion)
Construct an **Abstract Tree** for each individual component
Construct an **Abstract Tree** for each individual component

Any non-leaf node **father** should be an over-approximation of its **sons**. In the sense that

\[(\ldots \ldots \text{father} \ldots \ldots) \text{ sat } F \Rightarrow (\ldots \ldots \text{any son} \ldots \ldots) \text{ sat } F\]
Construct an **Abstract Tree** for each individual component.

For instance, the Combination of all roots satisfies the desired property implies that all combinations of the leaves satisfy the same property.

\[(\text{roots}) \text{ sat } F \implies (\text{any leaf, any leaf, ... any leaf}) \text{ sat } F\]
Construct an **Abstract Tree** for each individual component

For instance, the Combination of all roots satisfies the desired property implies that all combinations of the leaves satisfy the same property.

\[(\text{roots}) \text{ sat } F \Rightarrow (\text{any leave, any leave, ... any leave}) \text{ sat } F\]

Thus in case all combinations sat F (i.e. System satisfies F) the top-down search may terminate quickly.
Construct an **Abstract Tree** for each individual component

For instance, the Combination of all roots satisfies the desired property implies that all combinations of the leaves satisfy the same property.

\[(\text{roots}) \text{ sat } F \rightarrow (\text{any leave, any leave, ... any leave}) \text{ sat } F\]

In case “some combination of leaves violates F” (i.e. System violates F or a bug) it may go though all combinations ... but it may also find the “bug” quickly
Lowest-Priority Feasible Tasks

Definition

A task \( T \) is \textit{lowest-priority feasible} if there is a priority order \( P \) such that:

1. \( P(T) = \text{"lowest"} \)
2. All tasks meet their deadlines.

Connection of \textit{Schedulability} vs. \textit{Feasibility}:

- SP Schedulability (priority order \textit{given})
  - Test task of lowest priority: lp-feasible?
  - If YES, remove and iterate
  - If NO, unschedulable
  - \( \mathcal{O}(N) \) tests

- SP Feasibility (priority order \textit{wanted})
  - For each task \( T \): lp-feasible?
  - If YES, assign lowest priority, remove task, iterate
  - If NO, try next task
  - No task lp-feasible: infeasible
  - \( \mathcal{O}(N^2) \) tests; “Audsley’s Algorithm”
Lowest-Priority Feasible Jobs

- How to test a task for lp-feasibility?
  - Test all jobs separately!
  - If some job can miss deadline: task not lp-feasible
  - If no job misses deadline: task lp-feasible

- How to test a job?
  - Test its scheduling window
Testing the Scheduling Window

Is C lowest-priority feasible?

Scheduling window of C
Testing the Scheduling Window

High priority

Medium priority

Low priority

Is $C$ lowest-priority feasible?

A\(_2\), B\(_2\)

A\(_1\) B\(_3\) A\(_3\)

Scheduling window of C

Problem: Combinatorial Explosion!
A Different Characterization of System Workload:

Request Function
DRT: Request bound over a Path

Path $\pi = (v_4)$
DRT: Request bound over a Path

Path \( \pi = (v_4, v_2) \)
DRT: Request bound over a Path

Path \( \pi = (v_4, v_2, v_3) \)
Request Functions

![Graph with vertices and edges labeled with numbers and pairs, and a line graph below showing a function $rf(t)$ and $rf_{(v_4,v_2,v_3)}$ over time $t$.](image)
Request Functions

The image shows a directed graph with nodes labeled $v_1, v_2, v_3, v_4, v_5$ and edges labeled with tuples indicating the relationship between nodes. The graph includes the following edges and weights:

- $v_1$ to $v_2$: (2, 5)
- $v_2$ to $v_3$: (1, 8)
- $v_2$ to $v_4$: (20)
- $v_3$ to $v_4$: (3, 8)
- $v_1$ to $v_5$: (20)
- $v_5$ to $v_4$: (5, 10)
- $v_3$ to $v_5$: (11)

Additionally, there is a graph showing the function $rf(t)$ with two functions $rf(v_4, v_2, v_3)$ and $rf(v_5, v_4, v_2)$ plotted over time $t$. The graph indicates the behavior of these functions at different time intervals.
Abstract Request Functions

The figure shows a graph with labeled vertices and edges. The vertices are labeled as $v_1, v_2, v_3, v_4, v_5$. The edges are labeled with the values 10, 11, 15, 20, and 20, respectively. The graph also includes time-related functions $rf(t)$, $arf$, and $rf(v_4, v_2, v_3)$, $rf(v_5, v_4, v_2)$. The $rf(t)$ graph is shown with different lines for each function, indicating their behavior over time.
Request Functions: Schedulability Test

Lemma

A vertex $v$ is $lp$-feasible if for all combinations of request functions $rf^{(T)}$ of higher priority tasks:

$$\exists t \leq d(v) : e(v) + \sum_{T \in \tau} rf^{(T)}(t) \leq t.$$  \hfill (1)

Problem: Combinatorial Explosion!
Overapproximation: \( mrf \)

- **Trick:** Define an overapproximation
- Let \( mrf^{(T)}(t) \) be *maximum* of all \( rf^{(T)}(t) \) for a task \( T \).
- **New test:**
  \[
  \exists t \leq d(v) : e(v) + \sum_{T \in \mathcal{T}} mrf^{(T)}(t) \leq t.
  \]
- **Efficient:** Only one test, no combinatorial explosion
- **Problem:** Imprecise!
Overapproximation: *mrf*

- Trick: Define an overapproximation
- Let $mrf^{(T)}(t)$ be *maximum* of all $rf^{(T)}(t)$ for a task $T$.
- New test:
  \[ \exists t \leq d(v) : e(v) + \sum_{T \in \tau} mrf^{(T)}(t) \leq t. \]
- *Efficient*: Only one test, no combinatorial explosion
- Problem: Imprecise!

* How can we get efficiency *and* precision?
Abstraction Tree for each DRT

Define an *abstraction tree* per task:

- Leaves are concrete *rf*
- Each node: maximum function of child nodes
- Root is maximum of *all rf*
Define an *abstraction tree* per task:

- Leaves are concrete $rf$
- Each node: maximum function of child nodes
- Root is maximum of all $rf$
Abstraction Tree  for each DRT

Define an *abstraction tree* per task:

- Leaves are concrete *rf*
- Each node: maximum function of child nodes
- Root is *mrf*, maximum of *all* *rf*
Combinatorial Abstraction Refinement

New Algorithm:
- Test one combination of all mrf.
- If fp-feasible: done
- Otherwise: Replace one mrf with all child nodes, get 2 new combinations to test
- Repeat until:
  - All combinations show fp-feasibility, or
  - A combination of leaves shows non-fp-feasibility
Evaluation: Runtime vs. Utilization

Comparing runtimes of
- EDF-test using dbf (pseudo-polynomial)
- SP-test based on Combinatorial Abstraction Refinement
Evaluation: Tested vs. Total Combinations

10^5 samples of single-job tests.
- Executed tests: in 99.9% of all cases, less than 100
- Total combinations possible: up to 10^{12}
Combinatorial Abstraction: Related Work

• Static Priority Feasibility: ECRTS 2013
• Response Time Analysis: RTSS 2014
• Task synchronization: TACAS 2015, ECRTS 2016
• Self-suspending tasks: RTNS 2016

(Rejected by RTSS 2016!)
Architecture & Features of MASI

Modelling of CPS: Discrete & Continuous Components:
DRT, Conditional Differential Equations

Abstraction/Approximation

Verification of
Non-Functional Correctness
(Timing Analysis)

Verification of
Functional Correctness

Code Generation for
(1) Real-Time Simulation
(2) Implementation/Deployment
**Pacemaker + ”Random” Heart**

**Aget:** Atrial events that are generated by the heart which work as input of the pacemaker

**AP:** Atrial pacing actions that the pacemaker applies on the heart

**Vget:** Ventricular events that are generated by the heart which are input of the pacemaker

**VP:** Ventricular pacing actions that the pacemaker applies on the heart

The sign ! indicates a signal is an output signal.

The sign ? Indicates a signal is an input signal.


**Random Heart Model**

**Atrial component**

**Ventricular component**

**Await** and **Vwait** timing values are generated within configurable range using a random number generator.
Pacemaker Model in TIMES++

t_PVARP, t_TAVI, t_TLRI, t_URI, t_TVRP, t_PVARP are different timing requirements
ECG: heart and pacemaker
## Summary

### Models and Analysis Complexity

<table>
<thead>
<tr>
<th>Models</th>
<th>Analysis Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility i.e. EDF-Schedulability</td>
</tr>
<tr>
<td>General graphs (Di-graph)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Trees/DAGs</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Cyclic graphs (GMF)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic (L&amp;L, deadline≠period)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>L&amp;L (sporadic, deadline=period)</td>
<td>Linear</td>
</tr>
</tbody>
</table>

For systems with utilization bounded by a constant less than 1 (or below 100%) **RTSS 2015 Best Paper**

Otherwise Strongly coNP-complete **ECRTS 2015 Best Paper**

!! The problem open for 30 years, theoretically interesting !! **[Ekberg and Wang]**

**NEW WORK!**

**What can we do?**

**[ECRTS 2012]**

!! The problem open for 30 years, theoretically interesting !!

**[Ekberg and Wang]**