Context

- **Systèmes temps-réel embarqués critiques**
  - De plus en plus complexes
  - Contraintes de temps et coûts de production
  - Importance des exigences non-fonctionnelles et en particulier du temps.

Model Based Software Engineering: automatiser l'analyse et la production de logiciels

Modèle → abstraction

- **Un modèle est une abstraction de la réalité, abstraction à partir de laquelle il devient possible de raisonner**
  - Calculer des **dimensions caractéristiques** (consommation énergétique, temps de réponse, température, etc.).
  - **Vérifier la conformité** des dimensions caractéristiques vis-à-vis d'exigences (poids d'un avion, nombre de passager, besoin en carburant par km…).
  - Fournir des **points de vus** différent en fonction des rôles, préoccupations et/ou expertises de chacun (constructeur ou utilisateur d'un train, d'un avion, d'une voiture).

- **C'est une définition plutôt vague, et c'est le but: pour modéliser, il faut de la rigueur (objectif = calcul, vérification) et de l'ouverture d'esprit (moyen = abstraction, point de vu)...**

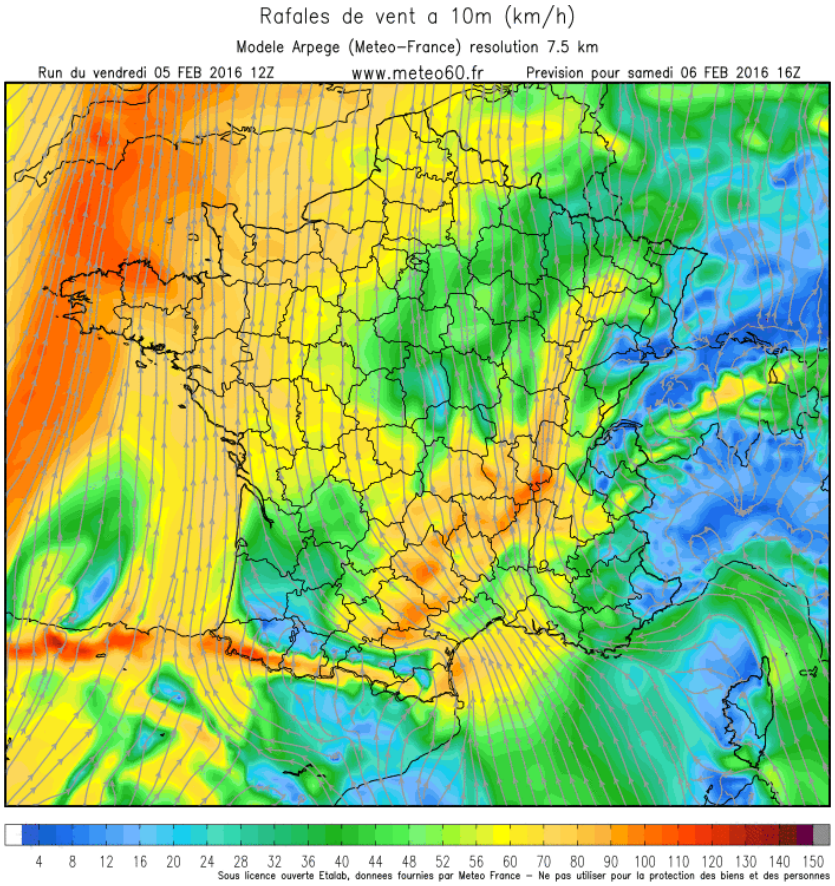
- **Prenons quelques exemple pour rendre tout cela plus concret…**

Définition de base

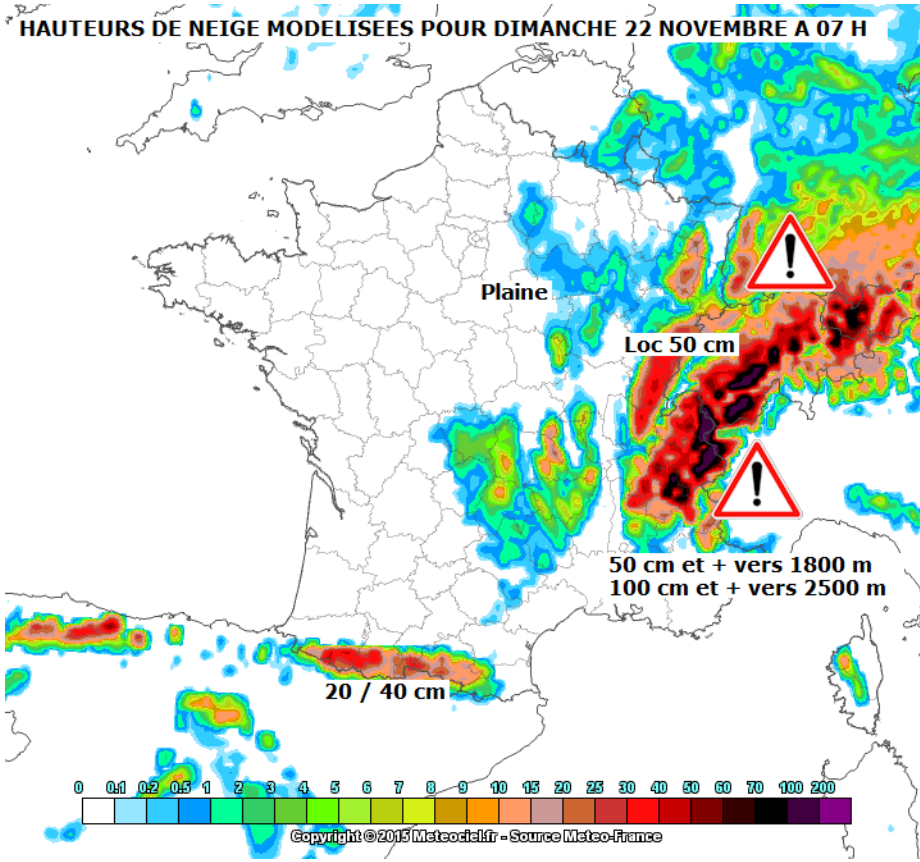
*Modeling, in the broadest sense, is the **cost-effective use of something** in place of something else for some **cognitive purpose**. It allows us to use something that is **simpler, safer or cheaper** than reality instead of reality for some purpose. A model represents reality for the given purpose; the **model is an abstraction** of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.*

"The Nature of Modeling" Jeff Rothenberg in Artificial Intelligence, Simulation, and Modeling, L.E. William, K.A. Loparo, N.R. Nelson, eds. New York, John Wiley and Sons, Inc., 1989, pp. 75-92

Exemple … Cartes météo

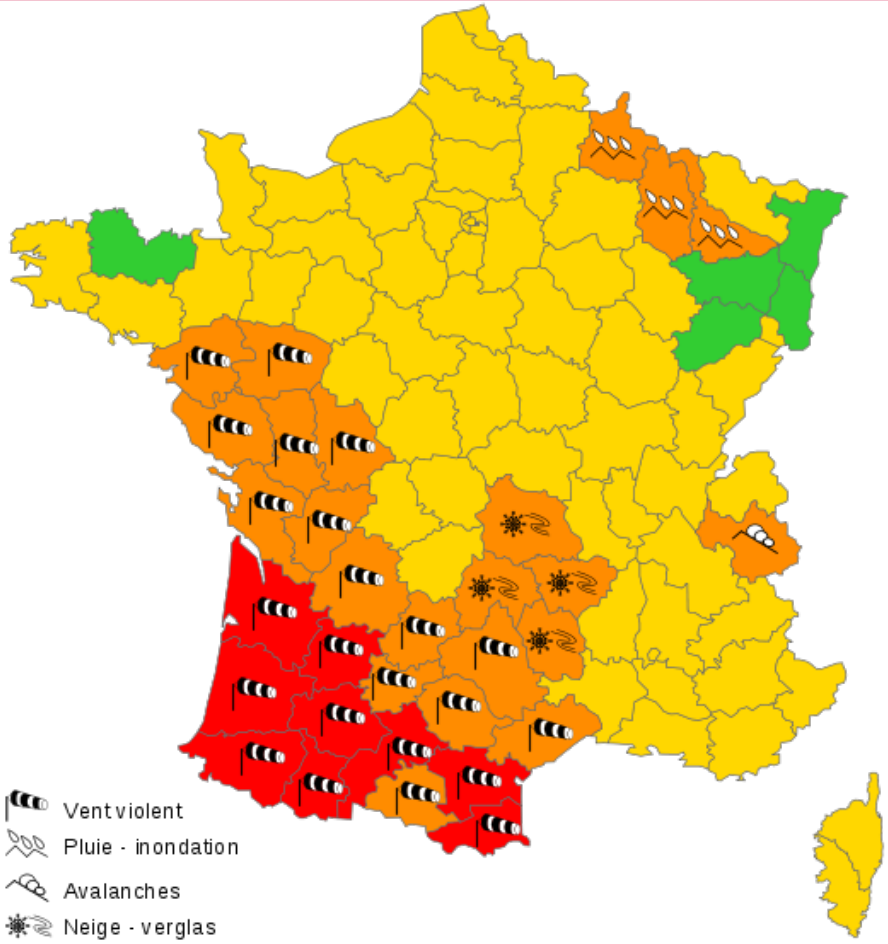


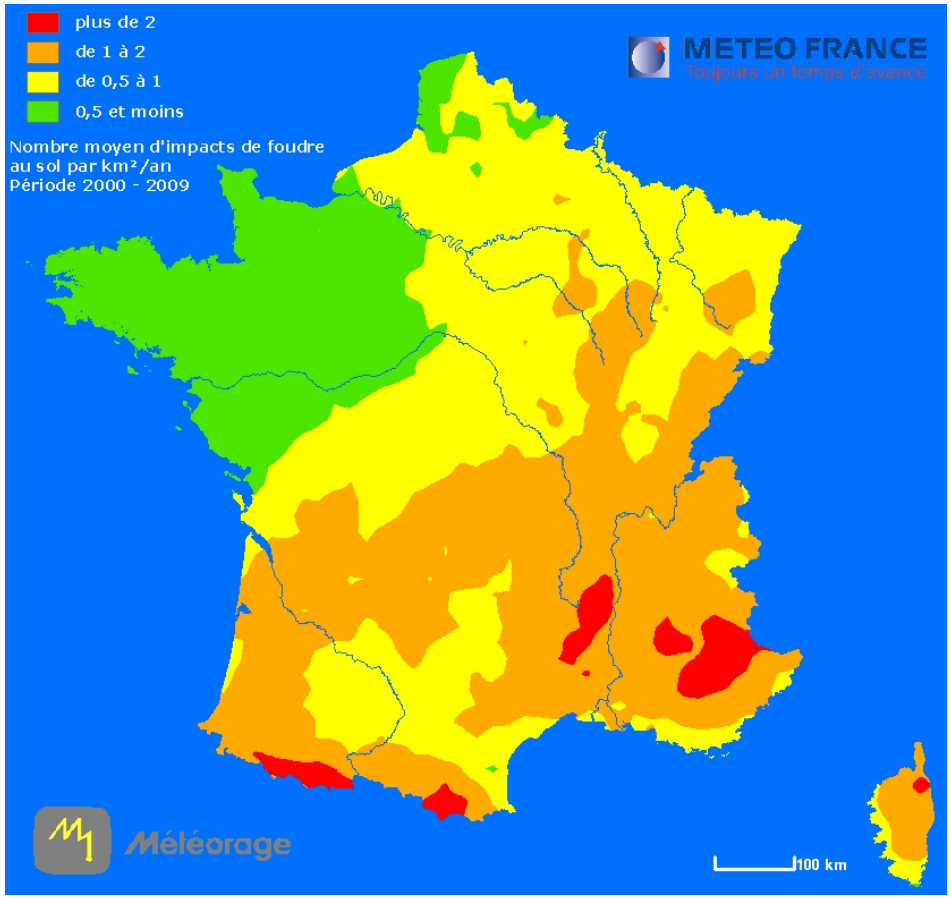
Carte des rafales de vents, avec direction et vitesse



Carte de l'enneigement, avec Risques d'avalanches

Cartes météo





Carte des niveaux de vigilance, vent/neige/avalanches

Nombre moyen de coup de foudre, par km2, par an, entre 2000 et 2009

Carte météo = abstraction

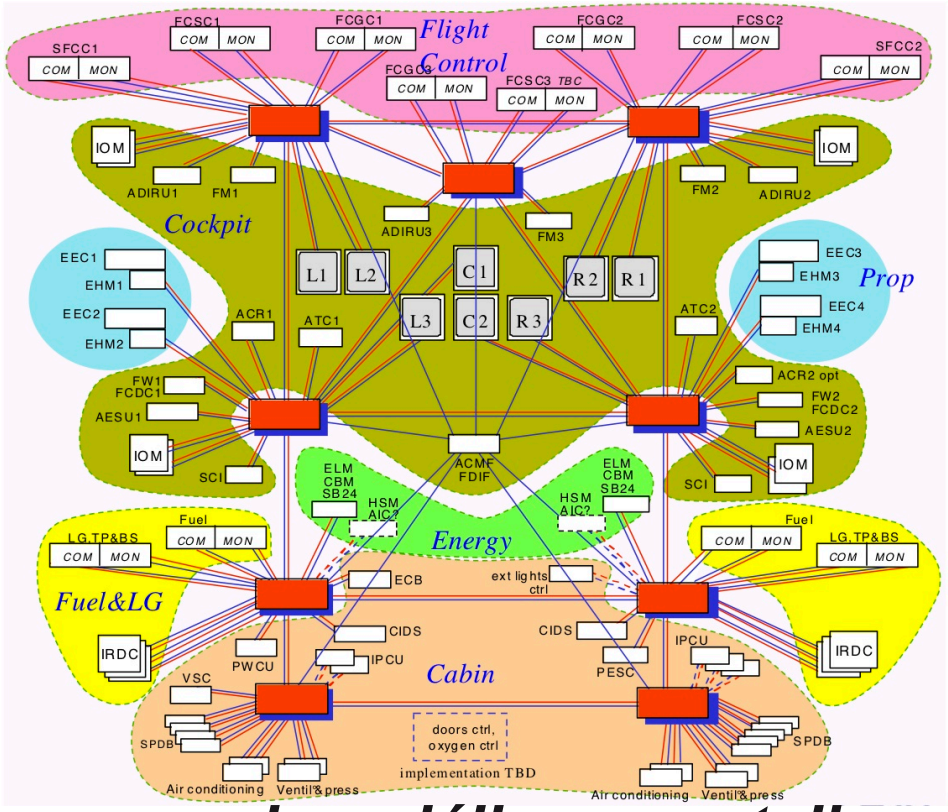
- **Un modèle est une abstraction de la réalité visant à:**
  - Calculer des **dimensions caractéristiques**
    - *Niveau de risque d'inondation/incendie/avalanche*
    - *Température max/min/moyenne*
    - *Vitesse des rafales de vent*
  - **Vérifier la conformité** des dimensions caractéristiques vis-à-vis d'exigences
    - *Un marin var vérifier la présence de vent, sa direction et sa vitesse…*
    - *Un alpiniste vérifiera la présence de neige, la présence de vent, la température…*
  - Fournir des **points de vus** différents en fonction des rôles, préoccupations et/ou expertises de chacun.
    - *Carte des températures, des vents, des précipitations, etc…*

Modèle = syntaxe + sémantique

- **Un modèle a aussi pour objectif de donner une représentation facile à comprendre, interpréter, retenir.**
- **Il faut pour cela:**
  - Une représentation textuelle et/ou graphique.
  - Une sémantique associée, plus ou moins abstraite/ rigoureuse.
- **Sur les exemples précédents, les couleurs servent souvent de représentation graphique (syntaxique) alors que la légende explique la sémantique associée.**
- **La sémantique associée est plus ou moins bien définie.**
- **Les modèles sont souvent des moyens de prise de décision par « prédiction ».**

Modèle d'architecture

- **Représentation structuré des éléments (logiciels, matériels, et/ou sous-systèmes) d'un système informatique.**



Comment, et pourquoi modéliser une telle architecture?

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

AADL: objectif principal et moyens

- **Faciliter la conception (*structuration et analyse d'architectures*) des systèmes temps-réels embarqués**
  - Définit une sémantique, **standardisée**, aussi précise (et concrète) que possible pour l'ensemble des éléments du langage. La sémantique est décrite en **langage naturelle** (**standard**, ~340 pages hors annexes)
  - Ne pouvant couvrir l'ensemble des exigences de conception du domaine, AADL propose des **mécanismes d'extension** (*i.e.* langage de propriétés et annexes)
  - Propose trois niveau de modélisation:
    1. Système
    2. Logiciel
    3. Matériel

AADL: objectifs détaillés

- **Conception, documentation, mais aussi**

- **Analyse de systèmes temps réel**
  - Temps de réponse des tâches
  - Temps de transmission des données (temps de latence, gigue)
  - Disponibilité, fiabilité, …

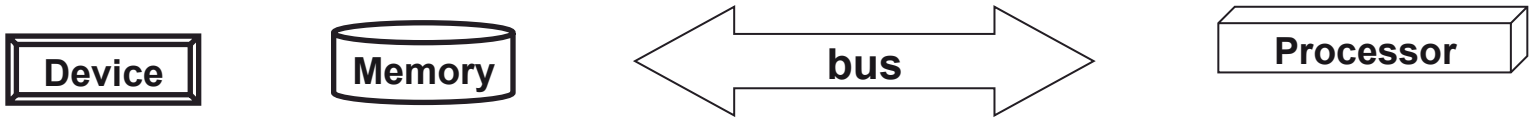
- **Production de code**
  - Représentation du code et des données d'un programme legacy
  - Modélisation du comportement (exhaustif ou partiel)
  - Génération automatique de code
    - Différents langages de programmation (Java, C, Ada, …)
    - Différents systèmes d'exploitations avec leurs API (RT-POSIX, FreeRTOS, VxWorks, ARINC653, OSEK…)

Caractéristiques générales

- **Langages de description d'architecture: les *composants* sont les éléments de base du langage.**
- **En AADL, les composants appartiennent à une Catégorie (Process, thread, data, processor, etc.)**
- **La déclaration d'un composants est structurée en**
  - Déclaration du type de composant: vise à définir ses interfaces de communications avec d'autres composant (ports, access points, …)
  - Déclaration de l'implémentation de composant: vise à définir la structure interne du composants (sous-composant, code, spec. comportementale, etc…)
- **Pour 1 Catégorie, on peut déclarer plusieurs types; pour 1 type on peut déclarer plusieurs implémentations**
- **Des propriétés (prédéfinies ou définies par le concepteur) peuvent être associées à chaque élément de modélisation (type de composant, implémentation, sous-composant, ports, connections,…)**

Description de la plate-forme d'exécution

- **Plusieurs catégories**
  - processor : unité de calcul + kernel
  - memory : composant de stockage (*e.g.* disque dur, mémoire vive, RAM, cache, etc.)
  - bus : lien physique de communication (bus, réseau, etc.)
  - device : interfaces physique/logique du système avec l'environnement (capteur/actionneur/pilote de périphérique)
- **Remarques:**
  - un processor modélise processeur + noyau contenant par exemple un ordonnanceur.
  - Un device sert typiquement à modéliser un capteur + le pilote de ce capteur (composant matériel et logiciel)



Description du logiciel

- **Plusieurs catégories**
  - thread : tâche qui exécute des fonctions du système (pas forcément un thread du noyaux…)
  - data : type de données abstrait (pas forcément de correspondance avec un type de donnée en programmation)
  - process : processus, un espace mémoire d'adressage logique pour l'exécution des threads qu'il contient
  - subprogram : procédure, comme pour les langages de programmation. N'as pas de valeur de retour
- **Remarque:**
  - Le standard définit de contraintes de validité d'un modèle; *e.g.* un process doit contenir au moins un thread.

Description système

- **Permet de :**
  - Structurer la description (sous-systèmes, système matériel et/ou logiciel)
  - Définir l'allocation des composants logiciels sur les composant matériels (via les binding)
  - Définir les modes de fonctionnement principaux du système

Les propriétés en AADL: principes de base

- **Une propriété permet d'associer une valeur d'un certain type à un élément du modèle.**
  - Le standard AADL définit un ensemble de ***propriétés standard***
  - il est possible de définir de nouvelles propriétés dans des ensembles de propriétés (**property sets**)
    - un langage complémentaire au langage de description d'architecture proprement dit
- **Les propriétés peuvent être associées à quasiment tous les éléments d'une description d'architecture en AADL**
- **Dans le langage de définition des propriétés, on peut contraindre l'applicabilité d'une propriété à un ensemble d'éléments (p.ex. ceux de catégorie processor uniquement)**

Propriétés en lien avec l'analyse d'ordonançabilité

- **Les composants AADL concernés sont**
  - Les threads, composant « principaux » en AADL car seuls composants « actifs »
  - Les processeurs car ils contiennent l'ordonnanceur
  - Les data qui peuvent représenter des données partagées.
- **L'ordonnancement est définie grâce à des propriétés prédéfinies**
  - *Dispatch_protocol*, le thread est
    - ***periodic***, le thread est réveillé périodiquement
    - ***sporadic***, le thread est réveillé sur réception de messages, avec un délais minimale entre deux réveils
    - ***aperiodic,*** le thread est réveillé sur réception de messages
    - ***timed,*** le thread est réveillé ***soit*** sur réception de messages ***soit*** sur échéance temporelle (timer réinitialisé sur réception de message)
    - ***Hybrid,*** le thread est réveillé ***à la fois*** sur réception de messages ***et*** sur échéance temporelle
  - *Compute_execution_time* représente l'intervalle de temps d'exécution d'un thread ou d'un sous-programme.
  - *Priority* permet d'associer une valeur de priorité à un thread.
  - *Deadline* permet d'associer une valeur d'échéance à un thread.
  - *Scheduling_Protocol* précise la politique d'ordonnancement associé à un processeur.
  - *Concurrency_access_protocol* précise la politique de protection associée à une donnée (aucune par défaut)
  - *Actual_Processor_Binding* précise sur quel processeur s'éxécutent les tâches

Exemple (1/2)

```
PACKAGE synchronous_Pkg
PUBLIC
WITH Base_Types;

SYSTEM synchronous
END synchronous;

SYSTEM IMPLEMENTATION synchronous.others
  SUBCOMPONENTS
    my_platform : PROCESSOR CPU;
    my_process : PROCESS my_process.impl;
PROPERTIES
Actual_Processor_Binding =>
         ( reference(my_platform) )
           applies to my_process;
END synchronous.others;
```

```
PROCESSOR CPU
PROPERTIES
    Scheduling_Protocol => (RMS);
END CPU;


PROCESS my_process
END my_process;
```

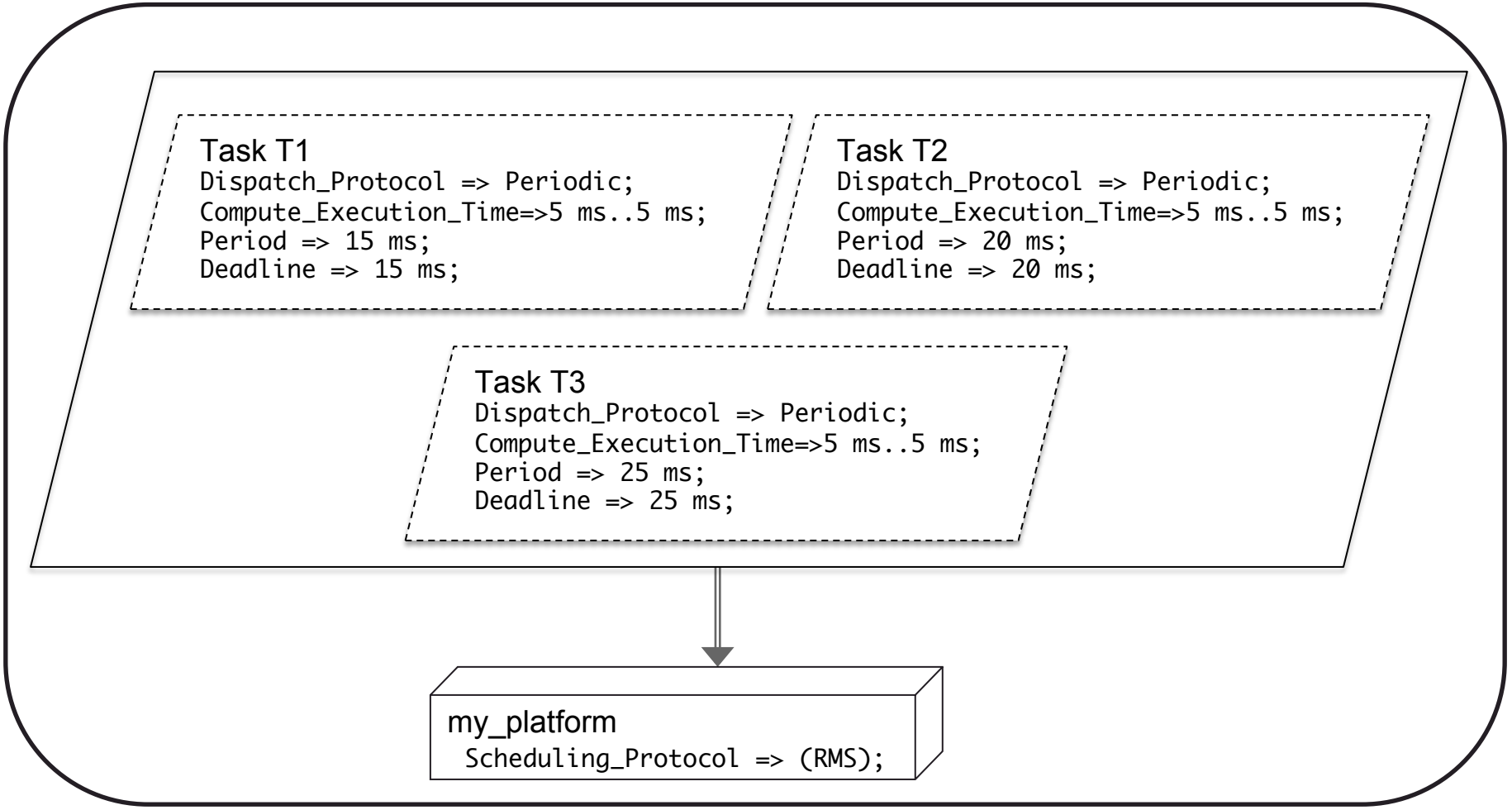
Exemple (2/2)

```
PROCESS IMPLEMENTATION my_process.impl
SUBCOMPONENTS
T1 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 15 ms;
     Deadline => 15 ms; };
T2 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 20 ms;
     Deadline => 20 ms; };
T3 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 25 ms;
     Deadline => 25 ms; };
END my_process.impl;
```

```
THREAD a_thread
END a_thread;

END synchronous_Pkg;
```

Représentation graphique



Exploitation avec AADL Inspector

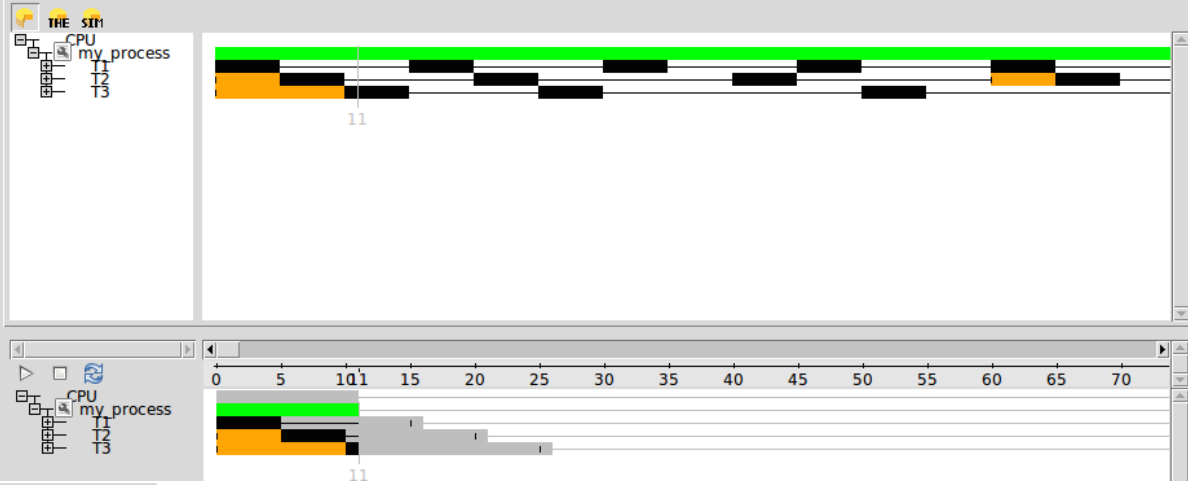
AADL inspector (/home/borde/Install/AI-1.5-beta-patched/examples/patterns/synchronous.aadl)

File View Wizards Tools ?

Behavior_Properties | Data_Model | Base_Types | HW | synchronous

```
346 SUBCOMPONENTS
347   T1 : THREAD a_thread
348     { Dispatch_Protocol => Periodic;
349       Compute_Execution_Time => 5 ms .. 5 ms;
350       Period => 15 ms;
351       Deadline => 15 ms; };
352   T2 : THREAD a_thread
353     { Dispatch_Protocol => Periodic;
354       Compute_Execution_Time => 5 ms .. 5 ms;
355       Period => 20 ms;
356       Deadline => 20 ms; };
357   T3 : THREAD a_thread
358     { Dispatch_Protocol => Periodic;
359       Compute_Execution_Time => 5 ms .. 5 ms;
360       Period => 25 ms;
361       Deadline => 25 ms; };
362 CONNECTIONS
363   C0 : PORT input -> T1.input;
364   C1 : PORT T1.output -> T2.input;
365   C2 : PORT T2.output -> T3.input;
366   C3 : PORT T3.output -> output;
367 END my_process.others;
368
369 THREAD a_thread
```

Static Analysis | Schedulability | AI Scripts



Static Analysis | Schedulability | AI Scripts

test	entity	
processor utilization factor	root.my_platform.CPU	We can not prove that the tas
worst case task response time	root.my_platform.CPU	All task deadlines will be met
response time	root.my_platform.CPU.my_process.T	15.00000
response time	root.my_platform.CPU.my_process.T	10.00000
response time	root.my_platform.CPU.my_process.T	5.00000

Static Analysis | Schedulability | AI Scripts

test	entity	
Task response time computed from simulatio	root.my_platform.CPU	No deadline missed in the computed scheduling : the ta
Number of preemptions	root.my_platform.CPU	0
Number of context switches	root.my_platform.CPU	46
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 5, best = 5 and average = 5.00000
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 10, best = 5 and average = 6.66667
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 15, best = 5 and average = 9.16667

Ce qu'on a vu jusque là…

- **Principales catégories de composants AADL**
- **Un modèle AADL simple**
  - Quelques composants et leur composition
  - Les propriétés pour un calcul de temps de réponse très simple
- **Une première transformation de modèle horizontale (même niveau d'abstraction) de AADL vers Cheddar (encapsulé par AADL Inspector)**
- **Pour aller plus loin dans les analyses temporelles**
  - Interactions entre composants: interfaces et connections
  - Représentation plus fine du comportement: appels de fonction, paramètres, et automates

Les features— les ports

- **Les ports modélisent les échanges d'information.**
  - data : transport de données ; pas file d'attente; contient une information utile (caractérisé par le type – AADL-- de donné transmis).
  - event : émission/réception d'un événement; mise en file d'attente possible; peut réveiller un thread; ne contient pas d'information autre que reçu/non-reçu.
  - event data : évènement + données ; mise en file d'attente possible; peut réveiller un thread; contient une information utile (caractérisé par le type -- AADL -- de donné transmis).

- **les ports peuvent être déclarés en**
  - entrée (`in`)
  - sortie (`out`)
  - entrée-sortie (`in out`)

Features: les accès aux composants

- **Un composant peut indiquer qu'il requiert (`requires`) ou qu'il fournit (`provides`) un accès à un sous-composant**
  - un bus, p.ex. pour un processor ou une memory
  - une data, p.ex. pour une donnée partagée entre plusieurs threads

Representation graphique 

- **Un composant thread ou data peut offrir des sous-programmes comme interface**
  - un thread serveur
    - p.ex. dans le cas d'un appel de procédure distante (RPC)
  - un composant de donnée proposant des méthodes d'accès
    - analogie avec les classes des langages objets

Representation graphique 

Features: paramètres et groupes de ports

- **Paramètres**
  - S'utilisent comme des ports mais réservés aux composants de la catégorie sous-programme

- **Regroupement de ports**
  - facilite la manipulation au niveau de la description
  - analogie avec un câble

Les connexions

- **Pour relier les « features » du même type entre elles**
  - ports
  - Paramètres
  - accès aux sous-composants
  - …
- **Les connexions sont définies dans les implementations de composants**
  - Entre les interfaces du composant englobant et de ses sous-composants
  - Entre sous-composants inclus dans le même composant.
- **Remarque: le standard définit des règles de cohérence**
  - Les features de sortie peuvent être connectés en « 1 vers n »
  - `event data ports` & `event ports` entrants: connections « n vers 1 » possible car gestion de files d'attente
  - data port: Connections « n vers 1 » **impossibles** car pas de files d'attente

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

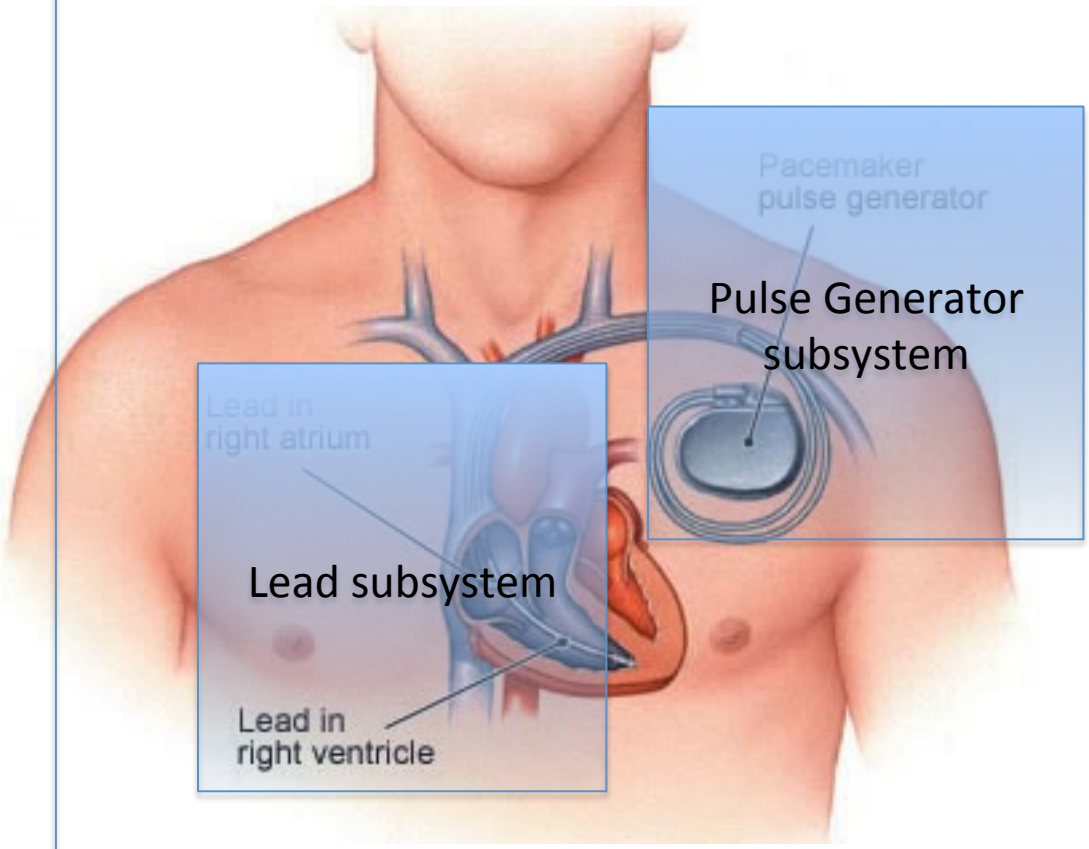
2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

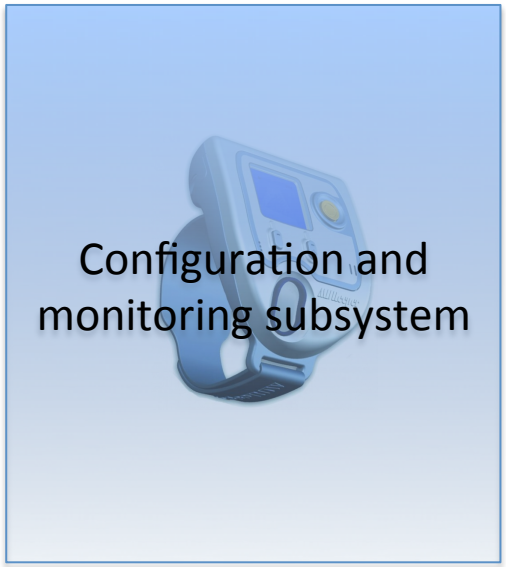
Prenons un exemple concret : un PACEMAKER

- **Implant qui contrôle les battements du cœur**
  - Surveille la présence de battements naturels du cœur
  - Stimule des battements « artificiels » en cas de besoin
- **Différents modes et différentes configuration en fonction des pathologies cardiaques et des patients (âge, musculation du coeur…)**
  - Surveille l'oreillette gauche? Droite? Aucune?
  - Stimule l'oreillette gauche? Droite? Aucune?
  - Surveille le ventricule gauche? Droite? Aucune?
  - Stimule le ventricule gauche? Droite? Aucune?
- **Système embarqué, temps-réel, critique … relativement simple (spécification en langage naturel ~60 pages, très denses)**

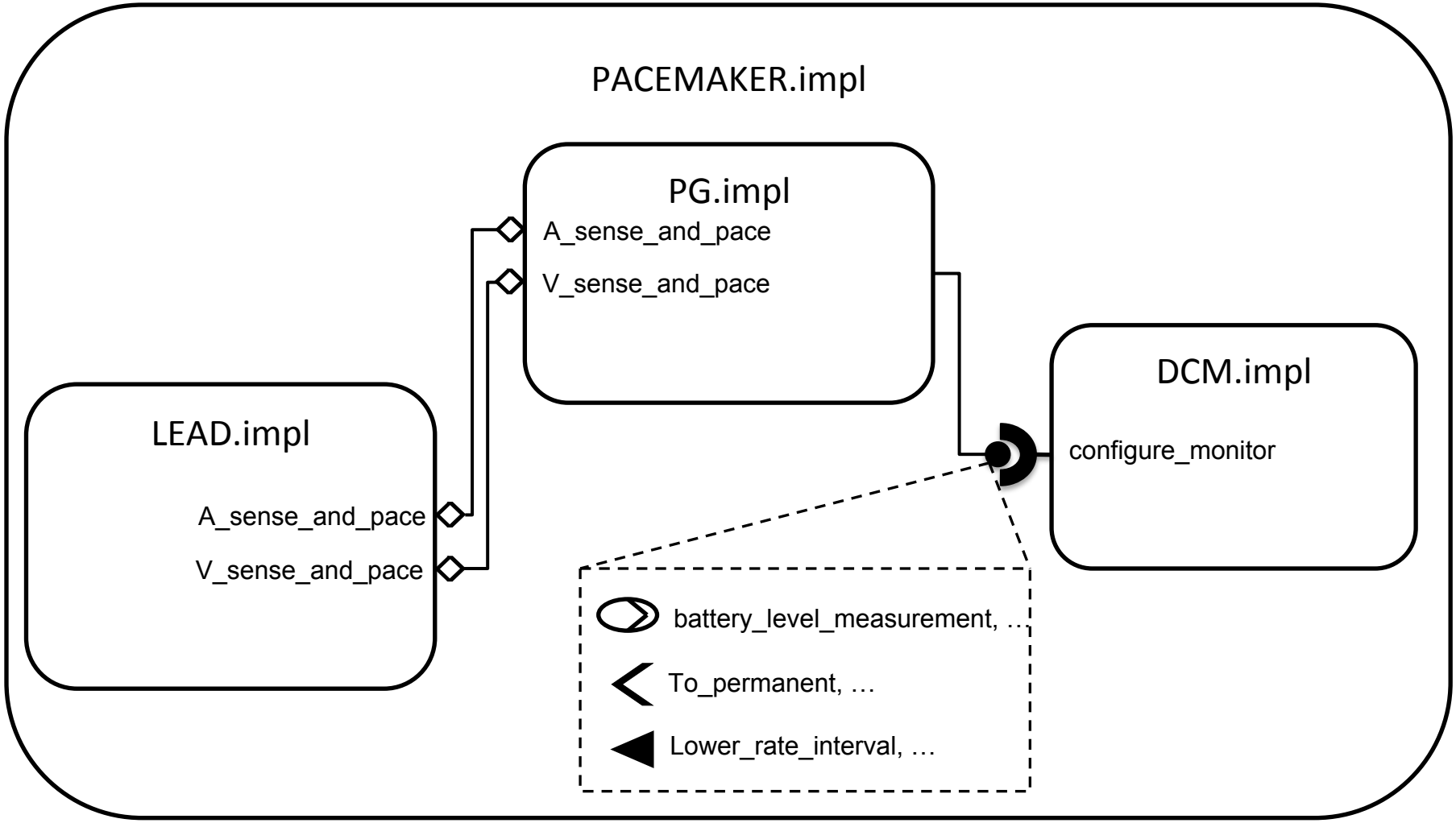
Décomposition systèmes/sous-systèmes

PACEMAKER system

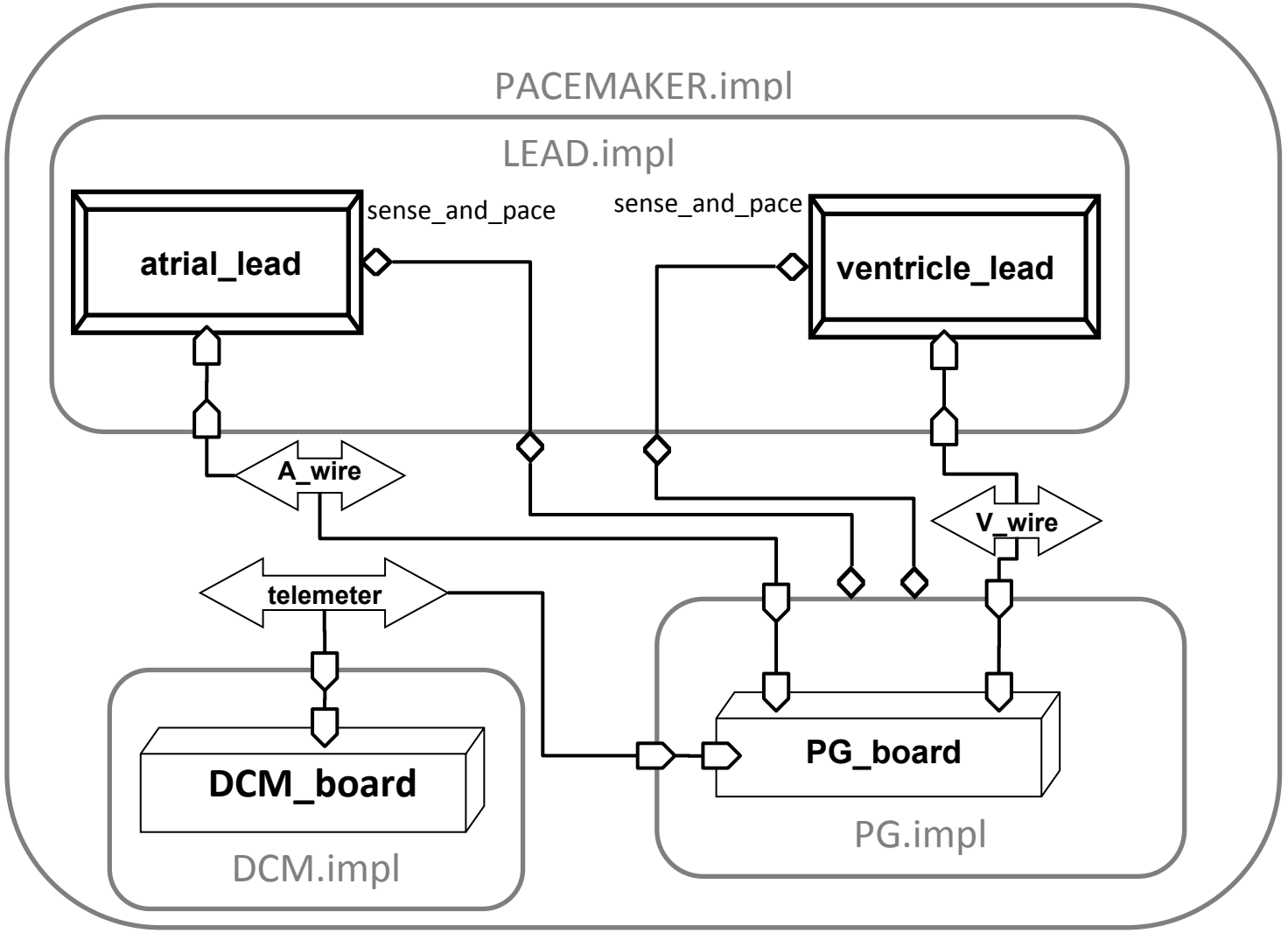




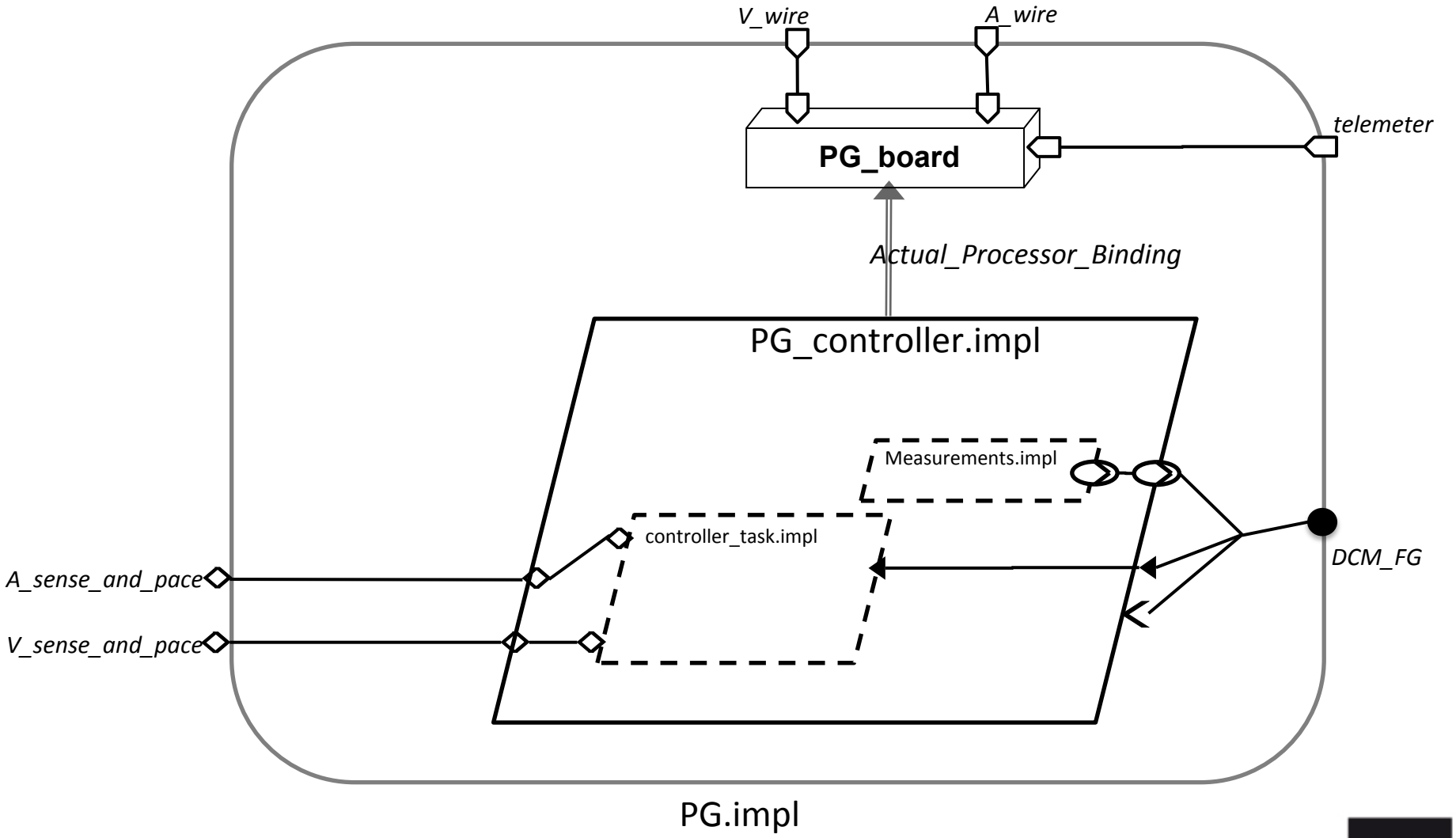
Modèle AADL: système global



Composants matériel



Composants logiciels



Pourquoi est-ce un système temps réel?

- **Considérons un mode dans lequel le PACEMAKER surveille et stimule l'oreillette et le ventricule.**
  - Le PACEMAKER doit respecter une période minimal et une période maximale entre deux battements
  - Le PACEMAKER doit respecter un délais fixe entre un battement dans l'oreillette et un battement dans le ventricule (délais AV)
  - Le PACEMAKER ne doit pas stimuler l'oreillette ni le ventricule si un battement naturel est capté
- **Comportement pas purement périodique, avec différents états pour la tâche de contrôle.**
- **On utilisera l'annexe comportementale.**

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

Annexe comportementale

- **Annexe standardisée**
- **Modéliser le comportement logiciel (thread, sous-programmes) par le biais de machines à état**
- **Une clause comportementale AADL est composée de trois parties:**
  - Déclaration des états
  - Déclaration des transition entre ces états
  - Déclaration des variables

Les états dans l'annexe comportementale

- **Initial, correspond à l'état dans lequel se trouve le composant après initialisation**
- **Final, correspond à l'état dans lequel se trouve le composant après finalisation (retour d'un appel de fonction ou finalisation d'une tâche)**
- **Complete, utilisable pour les threads seulement: correspond à un état dans lequel le thread n'utilise pas la ressource d'exécution (préempté, attente passive, etc…)**

- **Une clause comportementale doit contenir un état initial et au moins un état final**
  - L'état final et initial peuvent être le même

Les transitions dans l'annexe comportementale

- **Un Etat source**
- **Une ensemble de conditions**
  - conditions d'exécution, correspondant à un switch/case dans l'exécution d'une fonctionnalité
    - Une condition peut porter sur le contenu d'un port, d'un paramètre, d'une donnée, d'une propriété, etc…
  - conditions de dispatch, correspondant aux conditions de réveil d'un thread à partir d'un état *complete* donné
    - à mettre en regard de la propriété dispatch_protocol du thread (Periodic, Sporadic, Aperiodic, Timed ou Hybrid)
- **Un état cible**
- **Un ensemble d'actions**
  - Séquence d'actions pouvant contenir une structure de contrôle (if/then/else; while; do … until) et des interactions avec l'environnement d'exécution
  - Les conditions dans les structures de contrôle des actions sont similaires à des conditions d'exécution

Interaction des clauses comportementales avec l'environnement d'exécution AADL

- **Appel de sous-programmes: sub!(v1, v2, r1);**
- **Computation (10 ms); représente un calcul de 10 ms**
- **p! sur un port de sortie p permet d'envoyer le message contenu dans p; p!(v) permet d'envoyer le message v à travers p**
- **p?(x) sur un port d'entrée permet de lire les message contenus dans le buffer associé à p et stocker le résultat dans x;**
- **p'count et p'fresh pour connaitre l'état d'un buffer correspondant à un port p**
  - p'count retourne le nombre d'éléments dans le buffer
  - p'fresh retourne true si la valeur a été mise à jour depuis sa dernière utilisation; false sinon

Exemple d'utilisation de l'annexe comportementale pour le PACEMAKER

```
Thread controller_task
  features
    V_i: in event port;
    V_o: out event port;
    A_i: in event port;
    A_o: out event port;
end controller_task;
```

```
thread implementation controller_task.impl
  properties
    Dispatch_Protocol => Timed;
    Period => 1000 ms;
  annex Behavior_Specification {**
    states
      idle: initial complete final state;
      detect_V: state;

    transitions
      idle -[on dispatch Period]-> idle {A_o!};
      idle -[on dispatch A_i]-> detect_V {
                    V_i?;
                    A_i?;
                    A_o!;
                    Computation(20 ms)};
      detect_V -[V_i'count > 0]-> idle {V_i?;};
      detect_V -[V_i'count = 0]-> idle {V_o!};
  **};
end controller_task.impl;
```

Outils de vérification formelle de l'annexe comportementale

- **Transformation vers**
  - BLESS, preuve formelle
  - FIACRE, model checking
- **Transformations de modèles horizontales**
- **Plus complexes que les transformations permettant le calcule de temps de réponse avec Cheddar.**
  - Plus grand écart sémantique
  - Langages d'entrée et de sortie plus riches

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

Analyse d'ordonnancement avec l'annexe comportementale: code AADL (1/3)

```
PACKAGE shared_data_Pkg
PUBLIC

PROCESSOR CPU
PROPERTIES
    Scheduling_Protocol => (RMS);
END CPU;

SYSTEM shared_data
END shared_data;

SYSTEM IMPLEMENTATION shared_data.others
SUBCOMPONENTS
  my_platform : PROCESSOR CPU;
  my_process : PROCESS my_process.others;
PROPERTIES
  Actual_Processor_Binding =>
      ( reference(my_platform) )
      applies to my_process;
END shared_data.others;
```

```
PROCESS my_process
END my_process;
PROCESS IMPLEMENTATION my_process.others
SUBCOMPONENTS
  T1 : THREAD T.i1;
  D1 : DATA D
     { Concurrency_Control_Protocol =>

Priority_Ceiling; };
  D2 : DATA D
    { Concurrency_Control_Protocol =>

Priority_Ceiling; };
  T2 : THREAD T.i2;
CONNECTIONS
  C1 : DATA ACCESS D1 -> T1.D1;
  C2 : DATA ACCESS D2 -> T1.D2;
  C3 : DATA ACCESS D1 -> T2.D1;
  C4 : DATA ACCESS D2 -> T2.D2;
END my_process.others;
```

Analyse d'ordonnancement avec l'annexe comportementale: code AADL (2/3)

```
THREAD T
FEATURES
D1 : REQUIRES DATA ACCESS D;
D2 : REQUIRES DATA ACCESS D;
END T;


DATA D
END D;
```

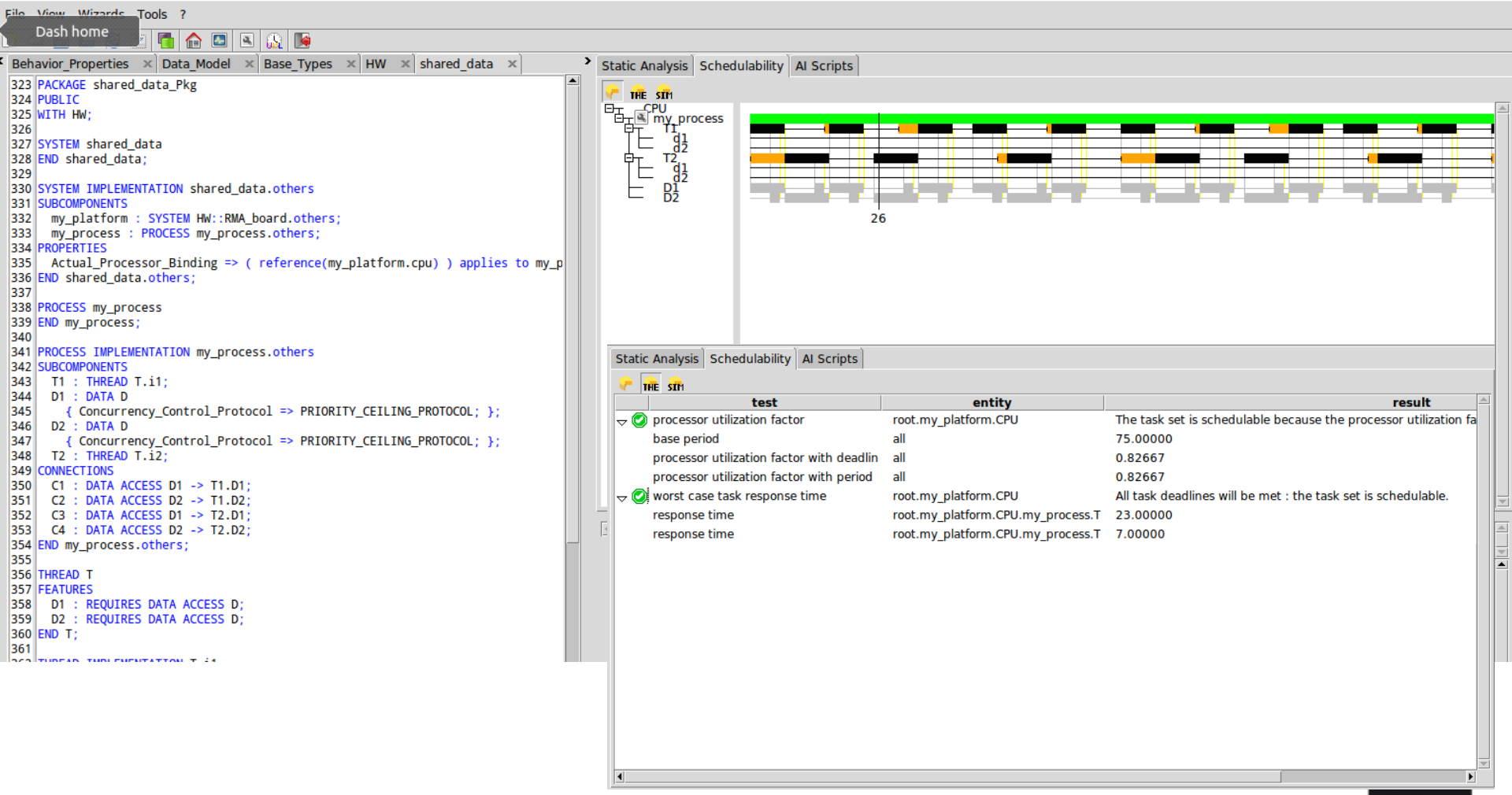
```
THREAD IMPLEMENTATION T.i1
PROPERTIES
Dispatch_Protocol => Periodic;
Compute_Execution_Time => 5ms..5ms;
Period => 15 ms;
ANNEX Behavior_Specification {**
States
    s : initial complete final
state;
Transitions
    t : s -[on dispatch]-> s {
      D1 !<;
      computation(3 ms);
      D2 !<;
      D2 !>;
      D1 !>
};
**};
END T.i1;
```

Analyse d'ordonnancement avec l'annexe comportementale: code AADL (3/3)

```
THREAD IMPLEMENTATION T.i2
PROPERTIES
Dispatch_Protocol => Periodic;
Compute_Execution_Time => 5ms..5ms;
Period => 25 ms;
ANNEX Behavior_Specification {**
States
    s : initial complete final state;
Transitions
    t : s -[on dispatch]-> s {
      D2 !<;
      computation(5 ms);
      D1 !<;
      D1 !>;
      D2 !>
};
**};
END T.i2;

END shared_data_Pkg;
```

Exploitation avec AADL Inspector



Exercice/question: calcul de temps de réponse

```
thread Task1
features
  l: requires data access lock;
  d1: requires data access d;
  d2: requires data access d;
annex behavior_specification {**
  states
    s_init: initial complete final state;
    s_comp: state;
  transitions
    t1: s_init -[on dispatch]-> s_comp;
    t2: s_comp -[true]-> s_init
      { computation(1 ms .. 2 ms);
        l!<;
        computation(0 ms .. 1 ms);
        l!>;
        computation(1 ms .. 2 ms) };
 **};
 end Task1;
```

On suppose que :
- Task1 et Task2 sont connectées aux mêmes données
- via l, d1, et d2 respectivement.
- Period (Task1) = 40 ms; Period(Task2) = 80 ms;
- Ordonancement = RMS

```
thread Task2
  features
    l: requires data access lock;
    d1: requires data access d;
    d2: requires data access d;
  annex behavior_specification {**
    states
      s_init: initial complete final state;
      s_comp: state;
    transitions
      t1: s_init -[on dispatch]-> s_comp;
      t2: s_comp -[d1 = d2]-> s_init
         { computation(0 ms .. 1 ms);
           l!<;
           computation(3 ms .. 8 ms);
           l!>;
           computation(4 ms .. 1 ms) };
        t3: s_comp -[otherwise]-> s_init
        { computation(12 ms .. 15 ms);
          l!<;
          computation(2 ms .. 2 ms);
          l!>;
          computation(9 ms .. 13 ms) };
  **};
end Task2;
```

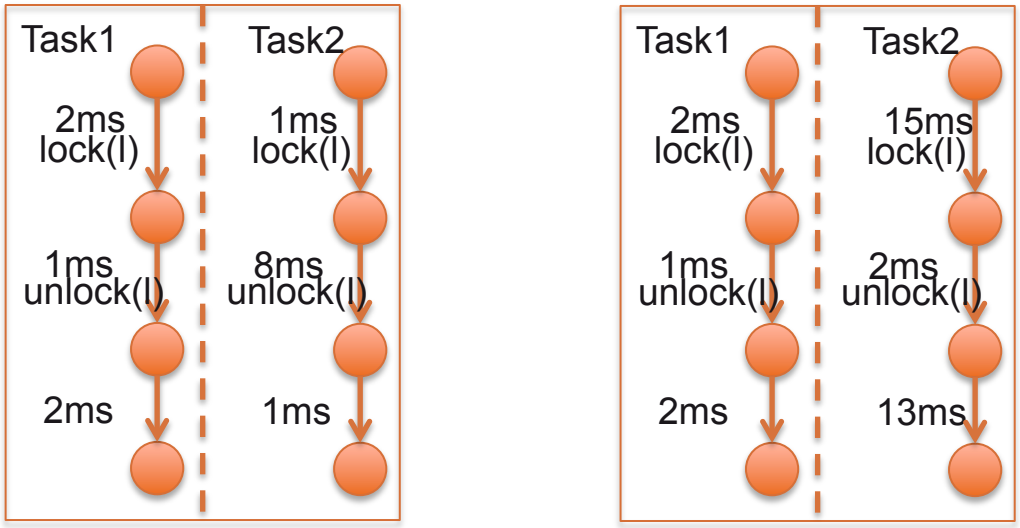
Comment calculer le pire temps de réponse de Task1 et Task2?

Limitations : sous-ensemble d'AADL supporté pour l'analyse de temps de réponse

- **Méthode 1, rapide mais pessimiste: approximation**
  - WCET (Task1) = 5
  - WCET (Task2) = 30
  - WCET (I) = 8

WCRT(Task1) = 13; WCRT(Task2) = 56

- **Méthode 2, lente mais moins pessimiste: produit croisé des branches de l'automate**



WCRT(Task1) = 13; WCRT(Task2) = 37

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

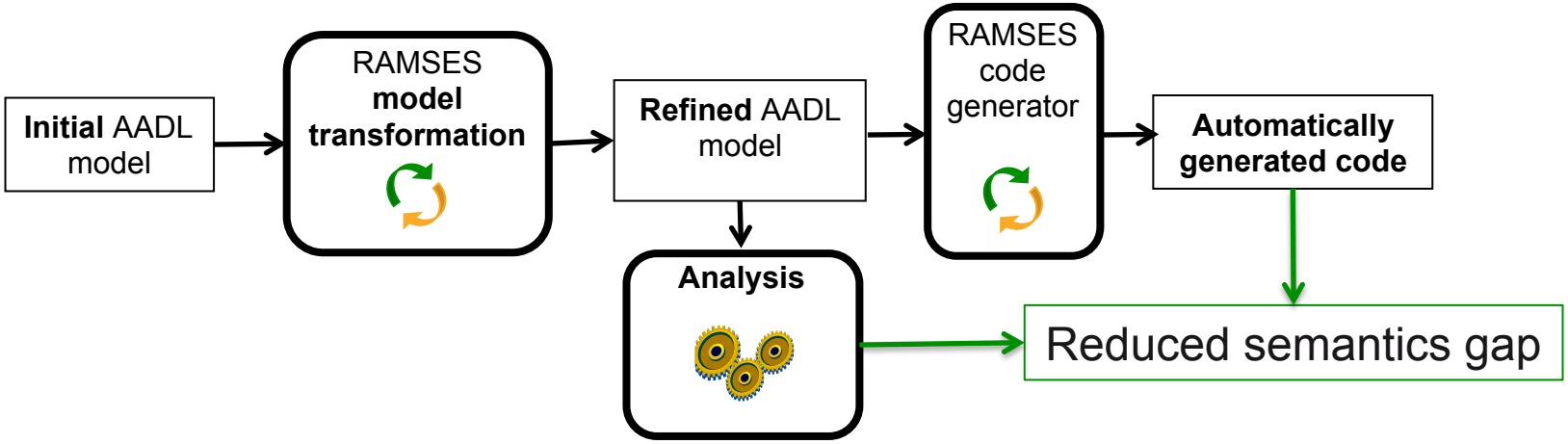
Problème avec l'analyse de modèles "abstraits"

- **L'implémentation d'un modèle abstrait nécessite une interprétation pour produire le code et les données associées:**
  - Les connections entre thread sont implémenté via des variables partagée ou files d'attente; ainsi que des fonctions avec potentiellement des sections critiques
  - Les modes opérationnels peuvent nécessiter l'ajout de threads pour gérer les changements de modes,
  - La gestion des erreurs requièrt des méchanismes de détecion et de recouvrement,
  - etc, etc.

Impact sur les résultats d'analyse?

Méthode proposée: automatisation de l'interprétation par transformations verticales de modèles

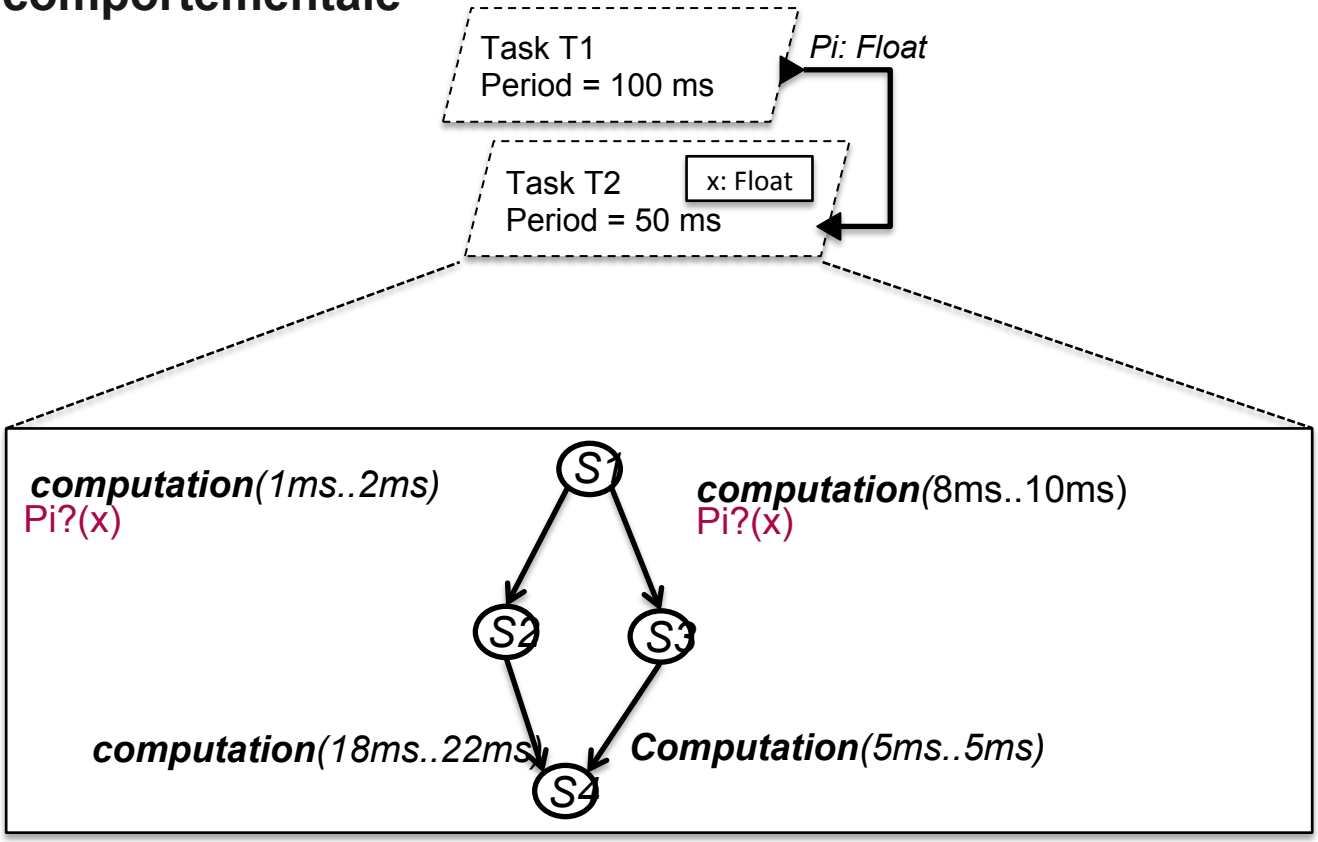
- **RAMSES: Refinement of AADL Models for Synthesis of Embedded Systems**



- **Papiers associés:**
  - Design Patterns for Rule-based Refinement of Safety Critical Embedded Systems Models. *International Conference on Engineering of Complex Computer Systems (ICECCS 2012)*
  - Architecture Models Refinement for Fine Grain Timing Analysis of Embedded Systems. *IEEE International Symposium on Rapid System Prototyping (RSP 2014)*

Modèles AADL d'entrée

- Des tâches périodiques ou sporadiques interconnectées avec leur propriétés temporelles
- Description du comportement des tâches, e.g. avec l'annexe comportementale

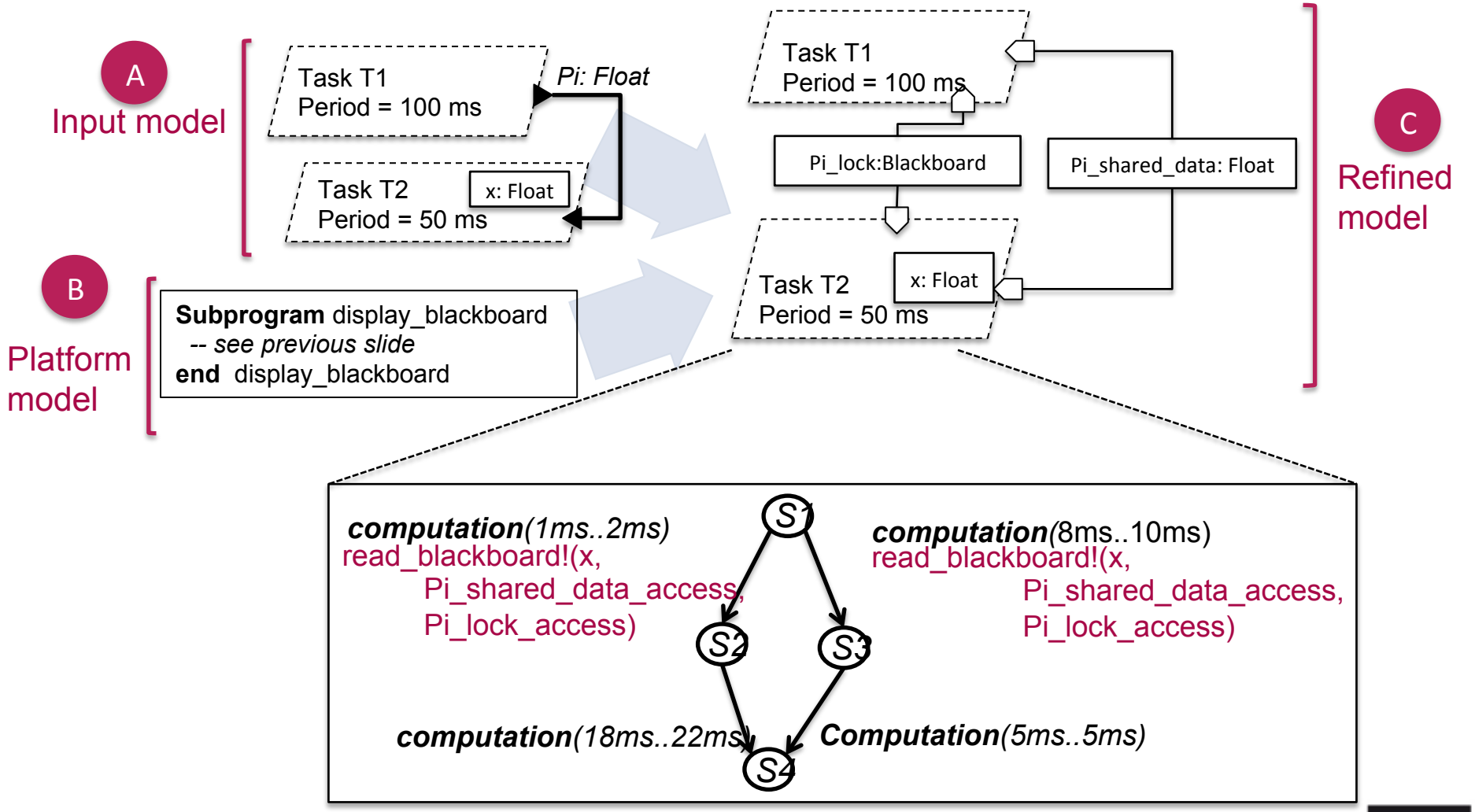


Modèles AADL de la plate-forme d'exécution

- Types de données et sous-programmes de l'OS, avec politique d'accès aux données partagées
- Comportement des sous-programmes e.g. avec l'annexe comportementale

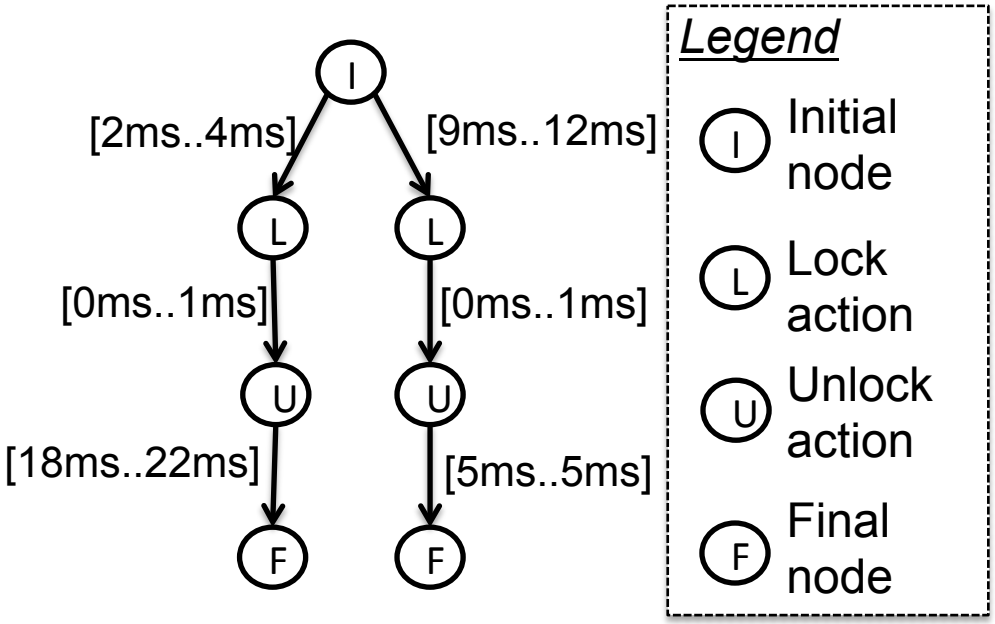
```
subprogram Read_Blackboard
features
  BLACKBOARD_ID: requires data access BLACKBOARD_ID_TYPE
                                {Concurrency_Control_Protocol => Priority_Ceiling; };
  …
annex behavior_specification {**
  states
    s: initial final state;
  transitions
    t: s-[]->s{computation(4 us .. 5 us) in bindings (RAMSES_Platform::POK_X86);
                BLACKBOARD_ID!<;
                computation(1ms..2ms) in bindings (RAMSES_Platform::POK_X86);
                BLACKBOARD_ID!>};
**};
end Read_Blackboard;
```

Illustration de la transformation de modèle



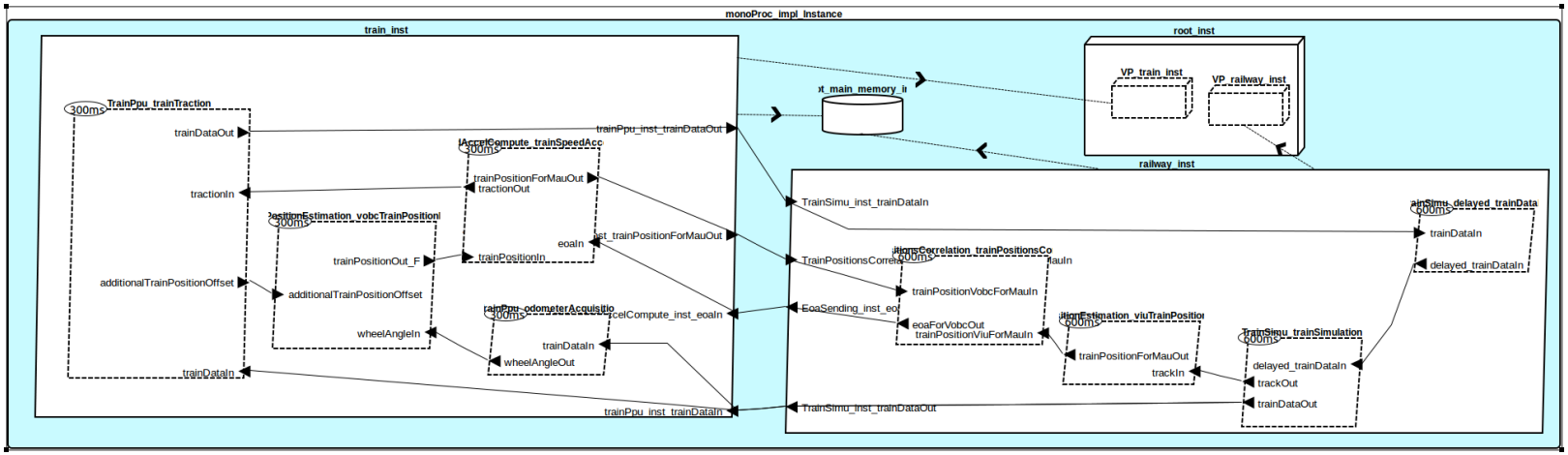
Extraction de scenarii d'ordonnancement

- **Par exploration du comportement, on extrait des arbres d'exécution possibles, *pour chaque thread*:**



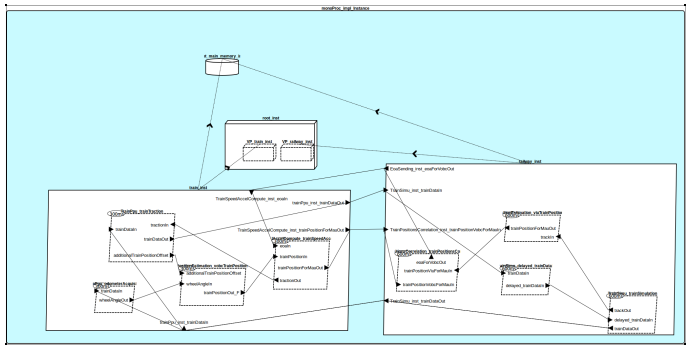
- **Un modèle AADL d'analyse est alors produit pour chaque combinaison de branches de l'ensemble des thread (produit Cartesian)**

Cas d'étude ferroviaire



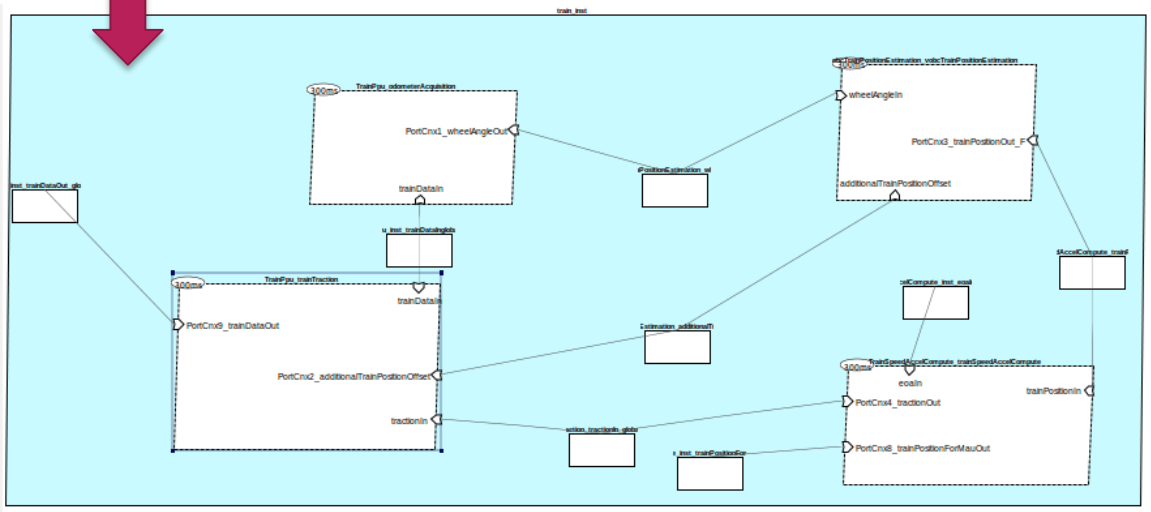
- **Cette méthode a été expérimentée sur un cas d'étude ferroviaire constitué de**
  - 2 process
  - 8 threads, avec 2 branches par thread.

Experimentation



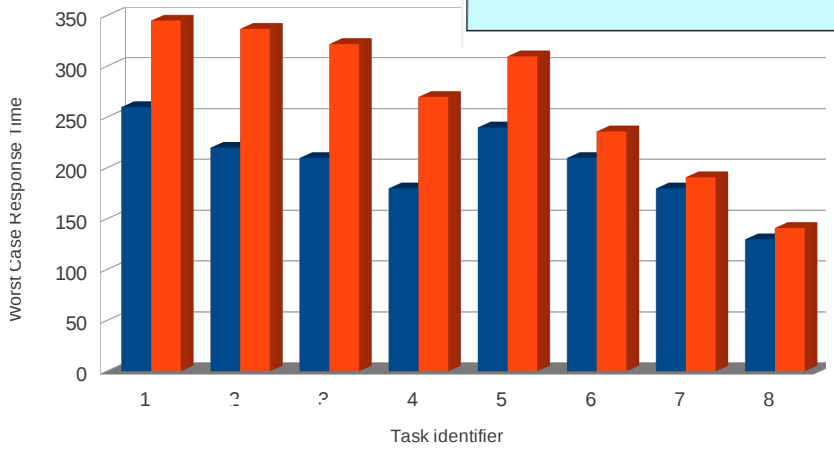
Design model

RAMSES refinement
AADL to AADL/ARINC653 model transformation



Implementation model

WCRT with overhead
WCRT without overhead



64 RTA,
Analysis results

Generated
C/Ada code

Plan de la présentation

1. **Modélisation d'architecture avec AADL**
   a. Présentation du langage AADL et analyse de temps de réponse sans section critique
   b. Exemple du PACEMAKER
   c. Présentation de l'annexe comportementale

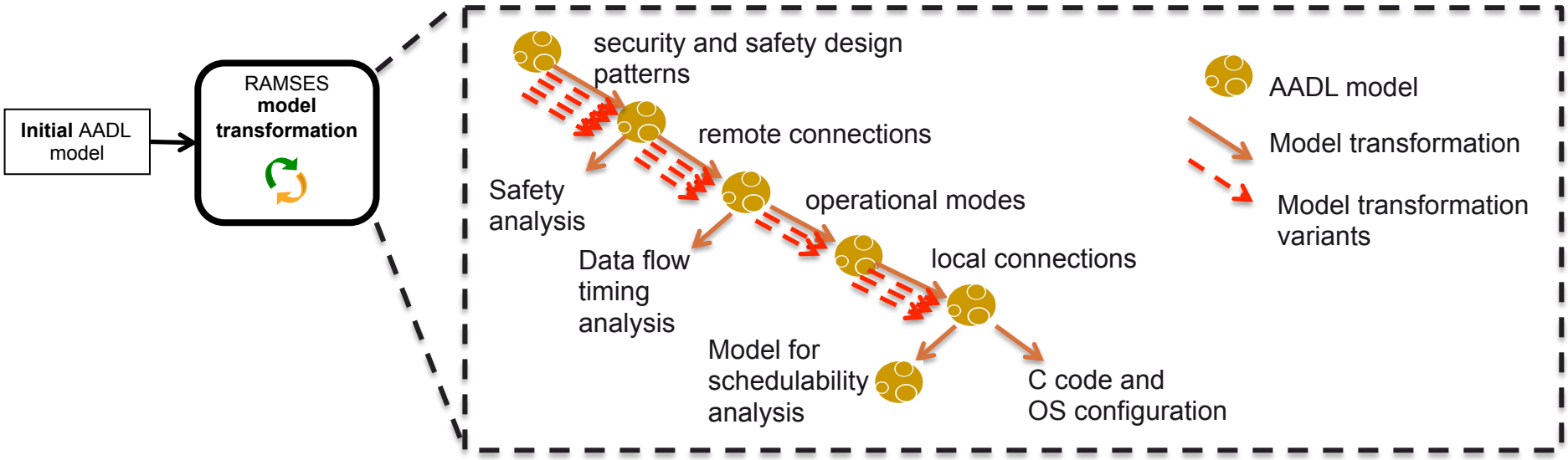
2. **Focus sur des travaux de recherche récents**
   a. Compléments pour l'analyse de temps de réponse (sections critiques)
   b. Model transformations for timing analysis
   c. Model transformations for architecture optimization (design space exploration)

Extensions de l'approche présentée

- **Transformation verticale : changement du point de vue.**
  - Prise en compte de l'impact sur les propriétés non-fonctionnelles de l'architecture
  - Peut-être poursuivi sur des transformations de modèles visant d'autres objectifs que la génération de code
    - Potentiellement des chaines de transformations
  - Variantes possibles dans les transformations avec des impacts variables en fonction des propriétés non-fonctionnelles
    - Exploration d'espace de conception par sélection des transformations

Transformations de modèles d'architecture et optimisation

- **Des alternatives de conception existent pendant le processus de raffinement**
  - Les transformations peuvent être chainées
  - Des variantes peuvent existe pour chaque maillon de la chaine



- **Papiers sur le sujet**
  - An Automated Approach for Architectural Model Transformations. *22nd International Conference on Informations Systems Development (ISD 2013).*
  - Multi-Objectives Refinement of AADL Models for the Synthesis Embedded Systems. *20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015).*

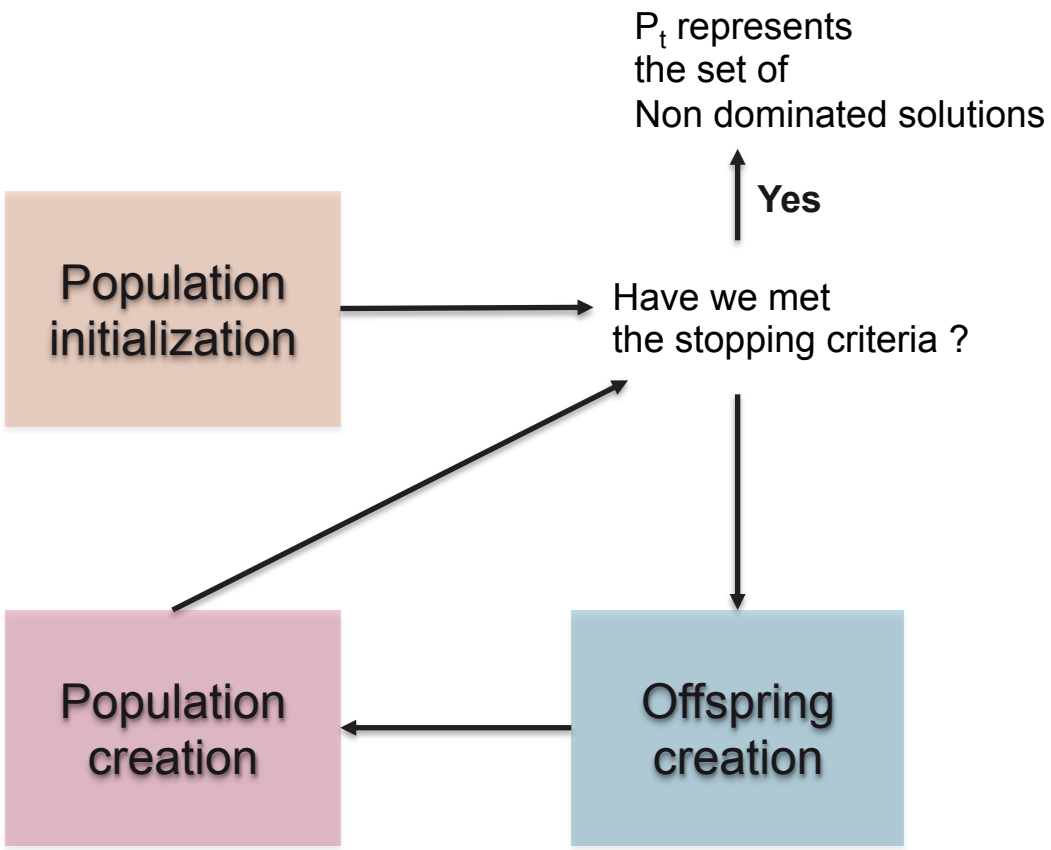
Problèmes techniques

- **Problem 1. Comment formaliser les alternatives de conception en utilisant des transformations de modèles**
- **Problem 2. Evaluer l'impact des alternatives sur les propriétés non-fonctionnelles**
- **Problem 3. Parcourir (un sous-ensemble) de l'espace de conception**

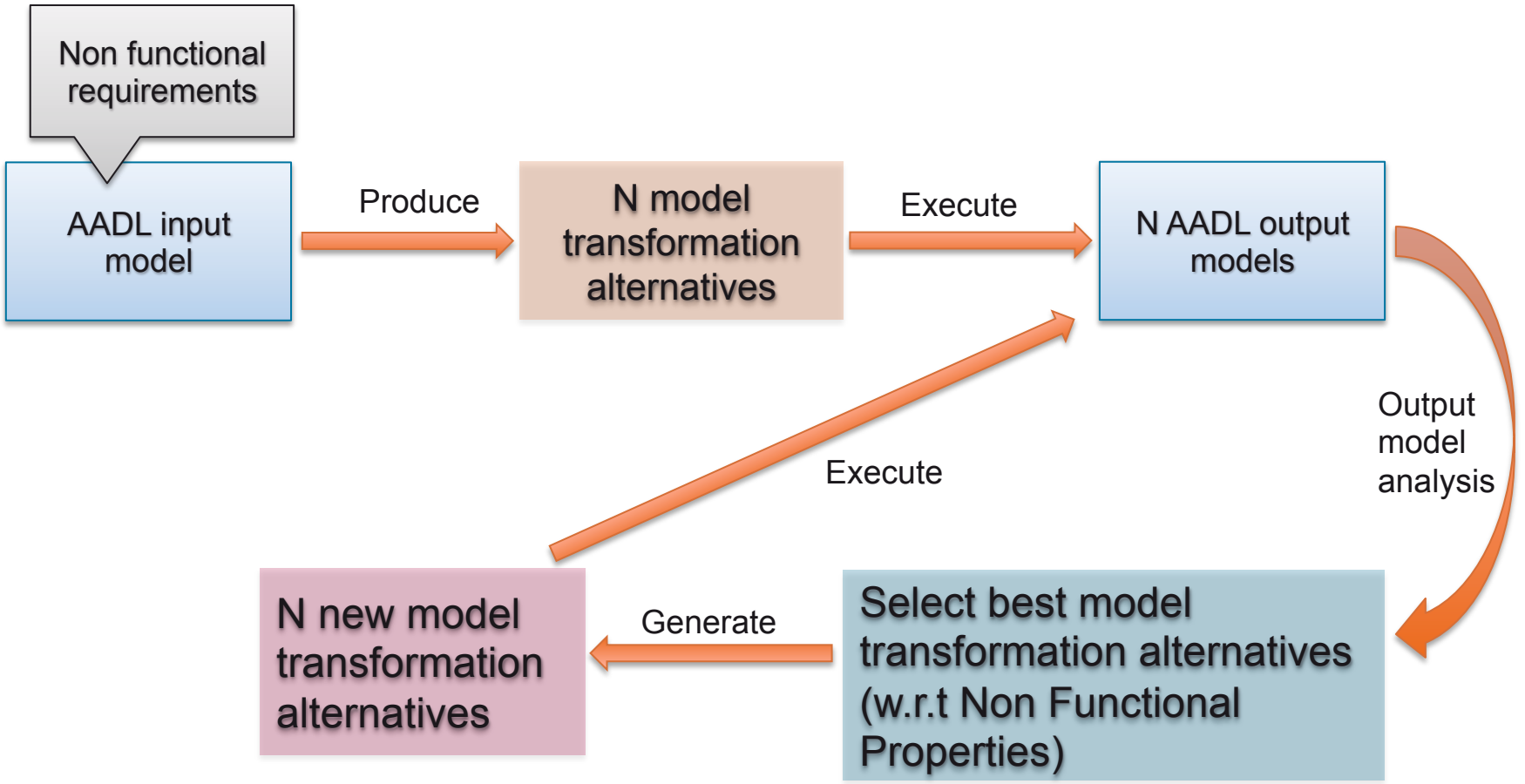
- **Contraintes additionnelle**
  - Sur les architectures produites par transformation; e.g. two variants of producer-consumer communications (linked lists and lookup tables) : If the producer uses a linked list the consumer has to use a linked list (vice versa)
  - NFPs are often competing: improving a NFP requires to degrade another NFP

Problème: combiner des techniques de **transformation de modèles** et d'**optimisation multiple-objectifs** tout en **préservant** les contraintes architecturales

Algorithme génétique, présentation du processus



Adaptation des Eas aux transformations de modèles

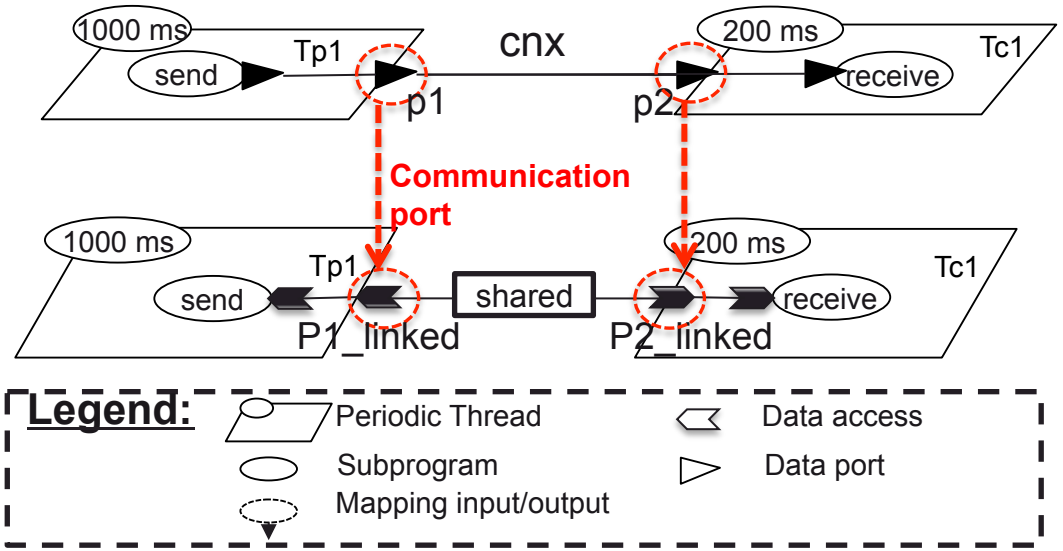


Choix techniques : transformations de modèles à bas de règles

- **Transformation de modèles → ensemble de règles de transformation**
  - Declarative matched rules
  - Pattern matching semantics
- **Transformation AADL vers AADL : ports de thread *transformé* en data access (comme illustré précédemment)**

```
rule communication_port
{
from   --- pattern matching section
     AADLModel!Port
to       --- creation section
     AADLModel! DataSubcomponent
…
}
```

(incomplete) ATL example

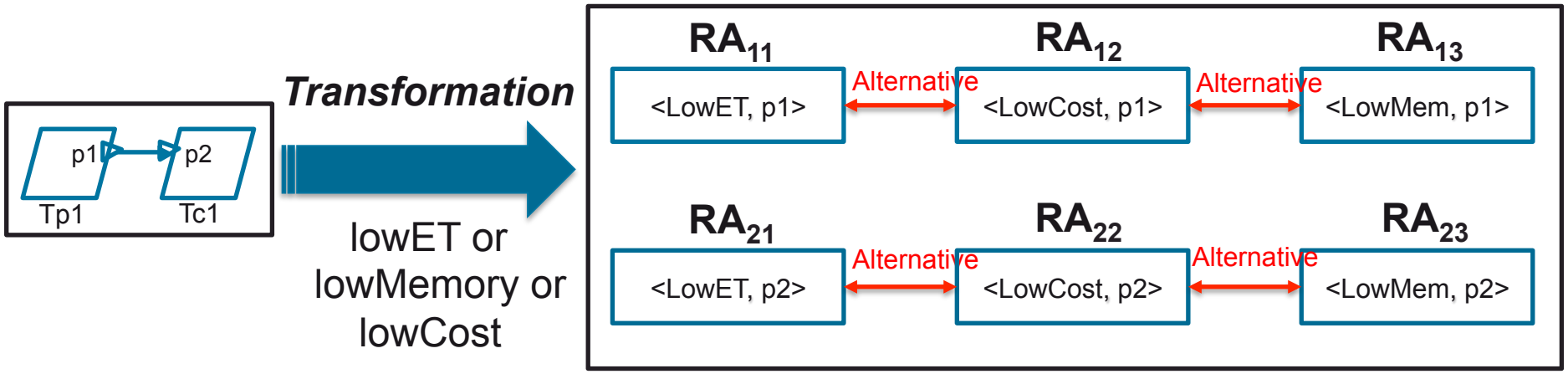


Structuration des transformations de modèles: RAs

- **Les transformations de modèles sont exécutées comme un ensemble d'application de règles (rule applications Ras-:**

  RA = <R, E, A>; with R = transformation rule;
  E=Elements from the input model;
  A=actions performed by R

- **Exemple: trois implémentation d'un protocole de communication inter-tâches**
  - LowET: port -> lookup table avec des indices calculés offlines et stockés en mémoire
  - LowMemory: port -> lookup table avec des indices calculés « at runtime »
  - LowCost: port -> liste chainée avec appels systèmes
- RA_i **= <(LowCPU |LowMemory | LowCost),** $port_i$ **, transform()>**



Objective: select on RA among alternatives to produce and evaluate candidate architectures

Validation des architectures produites

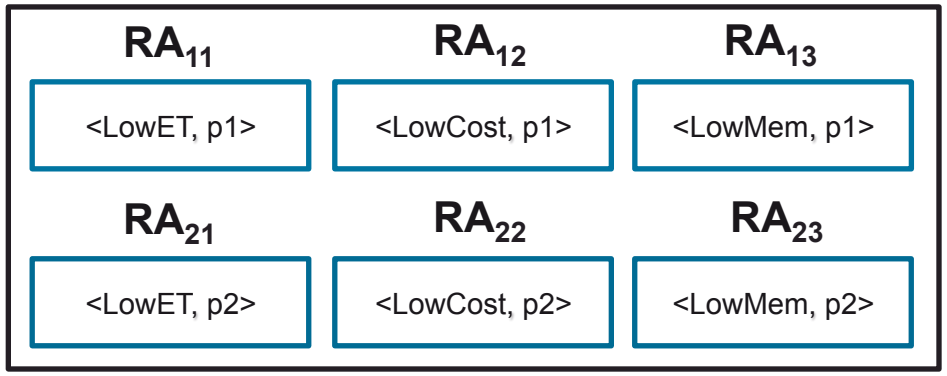
- **On doit assurer que les transformations produisent des architectures corrects *i.e.:***
  - Qui respectent un ensemble de contraintes architecturales (*e.g.* OCL…)
  - Qui satisfassent les exigences non-fonctionelles (NFR).

- **Pour les NFRs, on valide l'architecture *a posteriori*: les modèles de sortie sont analysées (potentiellement avec des outils AADL existants).**

- **Pour les contraintes structurelle, on les vérifie *a priori*: on transfer les contraintes exprimées sur les architectures de sortie en contraintes sur la selection des alternatives d'application de règles (RAs)**

Constraints on model transformations composition

- **Nous formalisons les contraintes sur la selection des applications de règles sous la forme d'une conjonction d'expressions booléenne:**
  - Une fonction Select et les opérateurs booléen classiques *and* ($\wedge$); *or*($\vee$); *not* ($\neg$);
  - La fonction Select doit satisfaire l'expression booléenne: retourne *true* si on sélectionne une application de règle (RA_i);
- **Exemple: Incompatibilité entre LowCost and (LowET or LowMemory)**



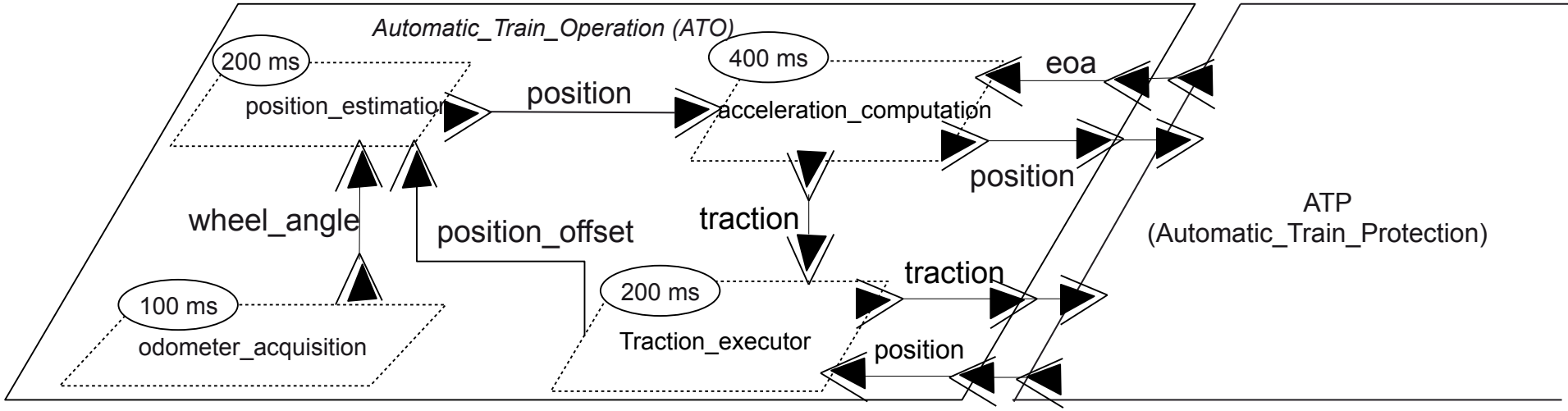
Boolean expression β

$$\begin{aligned} &\text{Select}(RA11) \Rightarrow \text{Select}(RA21) \vee \text{Select}(RA23) \\ \wedge\ &\text{Select}(RA12) \Rightarrow \text{Select}(RA22) \\ \wedge\ &\text{Select}(RA13) \Rightarrow \text{Select}(RA21) \vee \text{Select}(RA23) \end{aligned}$$

- **L'expression booléenne β peut être résolue en utilisant des techniques de satisfaisabilité (SAT solvers)**
  - L'application de SAT à β fournit 5 solutions: $\{RA_{11}, RA_{21}\}, \{RA_{11}, RA_{23}\}, \{RA_{12}, RA_{22}\}, \{RA_{13}, RA_{21}\}, \{RA_{13}, RA_{23}\}$

Cas d'étude ferroviaire

- **Objectif: generation de code**
- **Problème:** conflits entre les propriétés non-fonctionnelles
  - Temps d'exécution, empreinte mémoire, cout de maintenance
- **Processus Automatic Train Operation (ATO)**
  - Contrôle la position, vitess et accélération du train
  - Communique avec l'ATP (Automatic Train Protection) pour vérifier la validité des données (moniteur d'exécution)



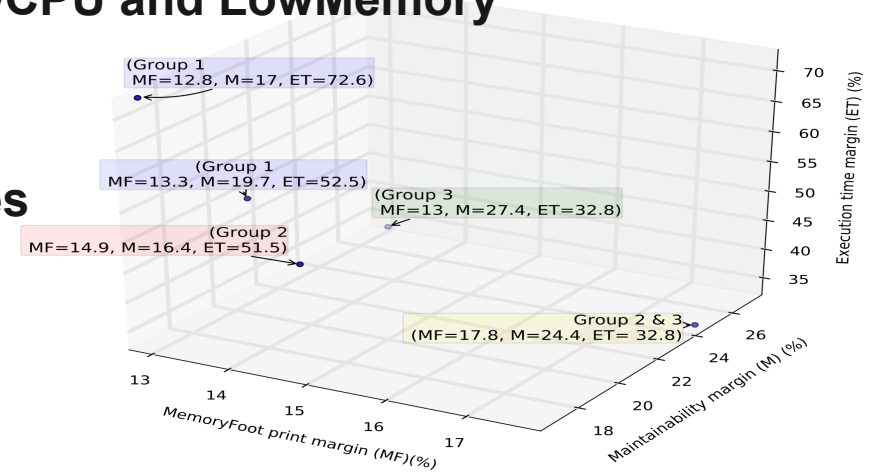
Résultats expérimentaux

- **Trois variantes d'implémentation du même protocole de communication**
  - LowET(Low Execution Time) LowCost (Low Maintenance Cost) and LowMemory (Low Memory Footprint)

- **Chaque variante transforme les ports et connections entre threads d'un même processus: jusqu'à 4 × 10^6 MTAs**

- **En considérant les contraintes structurelles: incompatibilité de l'implémentation LowCost avec les implémentations LowCPU and LowMemory**
  - Nombre de MTAs réduit à 16807

- **Production de 5 solutions non-dominées**



Conclusion and perspectives

- **De nombreux travaux sur**
  - La modélisation et l'analyse de modèles d'architectures de systèmes temps-réel
  - Les techniques de transformations de modèles
  - L'optimisation d'architectures de systèmes temps-réel

… Mais peu de travaux à l'intersection de ces domaines.

- **Malgré la présence de problèmes récurrents**
  - Courbe d'apprentissage des langages de modélisation.
  - Quel sous-ensemble du langage utiliser pour une exploitation donnée (analyse…)?
  - Complexité des transformations mise en œuvre.
  - Comment chainer des outils/transformations de modèles? processus ou méthodologie d'utilisation; interventions manuelles possibles, dépendances et conflits entre outils/ transformations.
  - Explosion combinatoire (analyse, exploration de l'espace de conception).

- **Usage de AADL:**
  - Industrie: langage pivot dans une chaine d'outils
  - Recherche: aide à l'expérimentation si possibilité de ROI

Thank you for your attention

etienne.borde@telecom-paristech.fr