Instituts thématiques

Inserm

Institut national
de la santé et de la recherche médicale

An Efficient Modeling and Execution Framework for Complex Systems Development

Isabelle Perseil, Laurent Pautet, Jean-François Rolland, Mamoun Filali, Didier Delanote, Stefan Van Baelen, Wouter Joosen, Yolande Berbers, Frédéric Mallet, Dominique Bertrand, Sébastien Faucou, Abdelhafid Zitouni, Mahmoud Boufaida, Lionel Seinturier, Joel Champeau, Thomas Abdoul, Peter H.Feiler, Chokri Mraidha, Sébastien Gérard

Roadmap

- -1- Introduction
- -2- A modeling and execution framework based on UML/MARTE and AADL models
- -3- Efficient embedded runtime systems through port, communication optimization
- -4- Formal verification of the architecture models
- -5- Conclusions

1 - Introduction

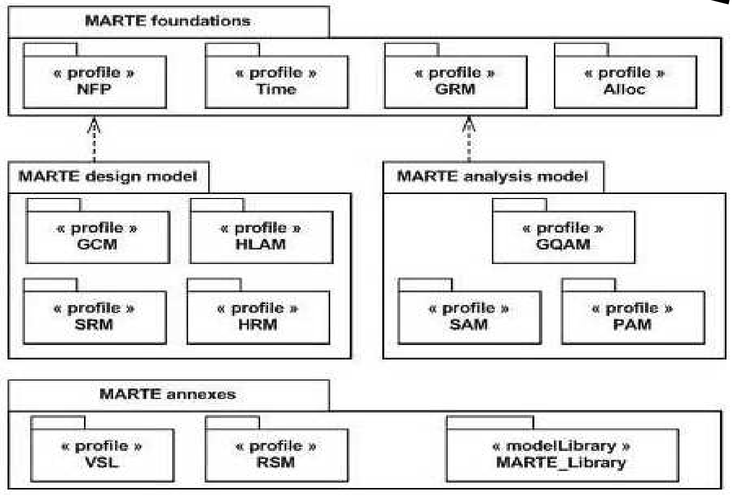
- Increasingly complex architectures, DRE systems, need for a model-driven approach
  - intricacy of the multiple interdependent features they have to manage.
  - meet new requirements of reusability, interoperability, flexibility and portability
  - global vision of the system
  - handling real-time characteristics
- AADL and MARTE - UML extension for RTE - are both modeling formalisms supporting the **analysis of real-time embedded systems**
- MARTE **Time Model facilities ➔** to represent faithfully AADL periodic/aperiodic tasks communicating through event or data ports, in an approach to end-to-end flow latency analysis
- AADL supports **a port communication model:** ensures deterministic processing of signal streams
- Verification of AADL models (operating modes), translation on formal models (IF)

2-a-MARTE in a nutshell

- MARTE: short name for “The UML profile for Modeling and Analysis of Real-time and Embedded systems”.
- This figure denotes the architecture of this UML profile.
- The HLAM profile provides concepts for **modeling** both aspects at a **high abstraction level**:
  - Modeling **quantitative features** such as deadline and period
  - Modeling **qualitative features** that are related to communication, behavior and concurrency
- Stereotypes «RtUnit» and «PpUnit» model **concurrency** and **shared information**

real-time unit

protected passive units



MARTE in a nutshell

- «RtUnit»: similar to the active object concept, with more detailed semantics → properties
  - autonomous execution resource, owns **one ore more schedulable resources** to handle incoming messages
  - owns a **concurrency and behavior controller** for managing message constraints according to its current state and current execution constraints attached to incoming messages
  - may own **one or several behaviors**. A message queue for storing incoming messages is defined for each of these behaviors
- «PpUnit»: enabling real-time units to share information
  - may specify their **concurrency policy** either globally or for all of their provided services through the **concPolicy** property.
  - do not own schedulable resources for execution, uses a **schedulable resource of the RtUnit** that invokes one of its services to execute it.
- «RtFeature» (real-time feature) defines **characteristics** that can be attached to a real-time service, a real-time action, a message or a signal.
  - may specify the occurrence kind of a behavior (e.g., periodic, aperiodic, sporadic, etc.) through the **occKind property**. A time reference used by the other relative timing properties may be specified through the **tRef property**.
  - may specify a deadline (relative or absolute) through the relDl and absDl properties.

Details on a MARTE-executor enabler

- Accord|UML: a model-based methodology ➔ embedded real-time applications development
  - to provide both **the method** and **the underlying toolkit** for specification and prototyping of embedded real-time systems.
- It enables executable models: a model is executable when it can be operated on a computer. ➔ model transformation <u>from MARTE model annotated with HLAM's extensions</u> to an existing executable platform.
- a dedicated **automatic code generation process** taking into account specified real-time features, and a specific execution framework.
  - provides an operating support for some of the high level real-time concepts defined in the HLAM sub-profile.
- The code generation facility provided by Accord|UML consists of:
  - (i) a set of transformation used to apply implementation patterns on real-time concepts;
  - (ii) a standard C++ code generator, "standard" meaning that it is not dedicated to Accord|UML or the execution framework.

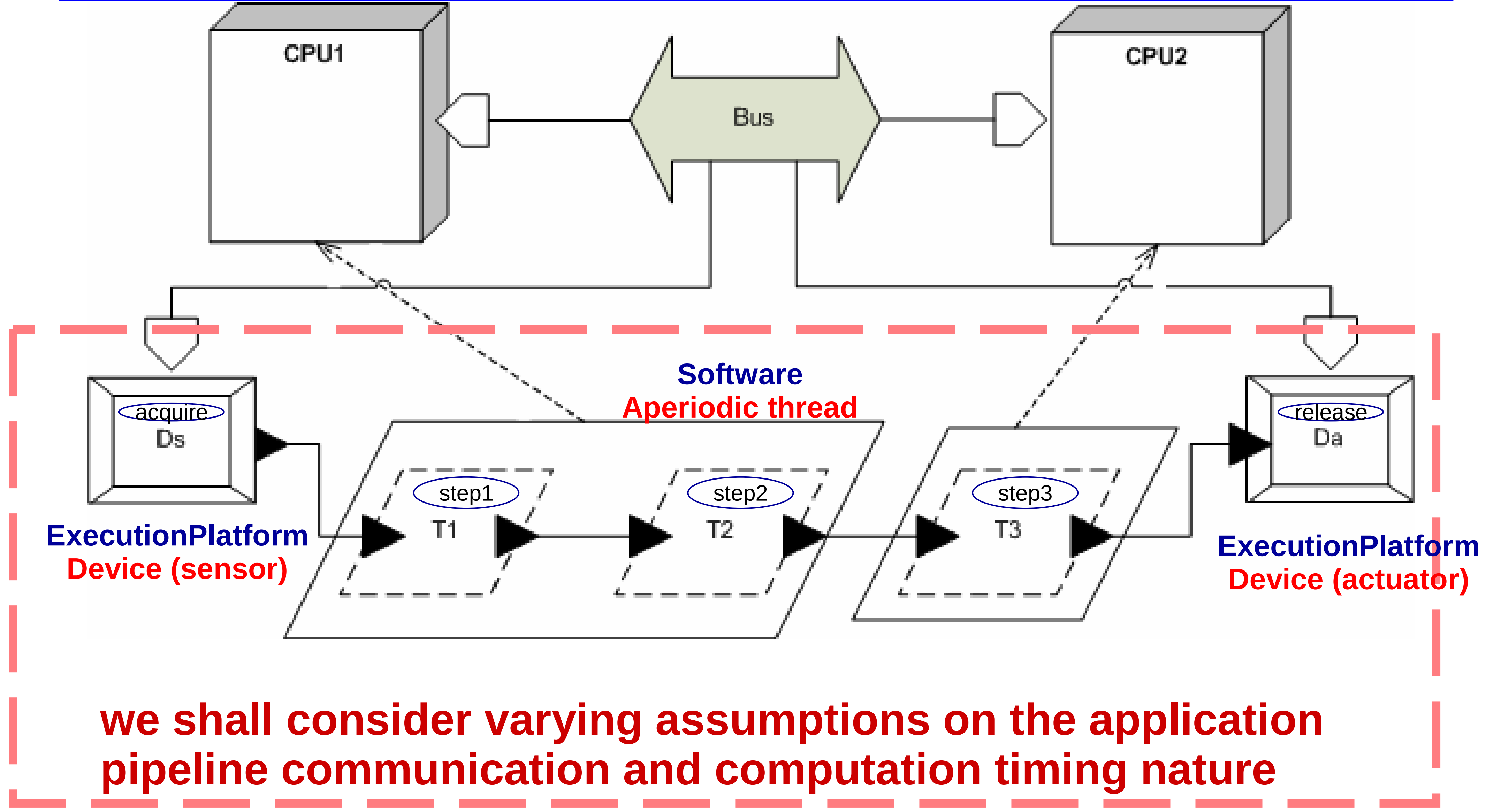
Details on a MARTE-executor enabler

- Within Accord|UML, **modeling of behavior for RtUnits** supported by the platform has to conform the following principle:
  - (i) **RtBehavior** state machine models the **control aspects** of the behavior of a real-time unit;
  - (ii) **RtService** operations for the **algorithmic concerns**.
- The specified behavior is operated using software execution resources.
  - **OS thread based**, managed in a global pool by the **RtUnit** that loads and starts the execution.
  - These threads are dynamically allocated to the **RtService** for execution and released after.
- Regarding the modeling rules, to comply with this execution semantics, **RtUnits** must be used with the **isDynamic** property set to false (execution resources are not created dynamically).
- The framework provides support for **communication and distributed execution**.

2-b-End to end flow latency in AADL and UML MARTE

- AADL focuses on specific aspects amenable to schedulability analysis
- MARTE scope is much larger / encompasses several modeling aspects valuable independently of specific analysis techniques.
- A MARTE model ➔ serve as a model repository to be processed and partially analysed by a wide range of techniques
  - including scheduling, schedulability, performance
  - from several communities: queueing networks, Petri nets, discrete-event simulations, synchronous-/reactive.
- ➔ serve to gather into a single model results from various teams/tools at different stages in the design flow.
- Critical point: ensure that **MARTE Time model** can express the same semantics as the one implied by **AADL periodic/aperiodic tasks communicating through event or data ports**
- AADL input models describe the application (software) parts as a set of tasks with **features** like the so-called ***dispatch_protocol*** (periodic, aperiodic, sporadic, background), latencies, execution times

AADL provides a two-level model with the software part to be bound with the hardware architecture



we shall consider varying assumptions on the application pipeline communication and computation timing nature

Two level AADL vs Three-level MARTE
three layer UML/MARTE → relations among the layers: the allocation mechanism

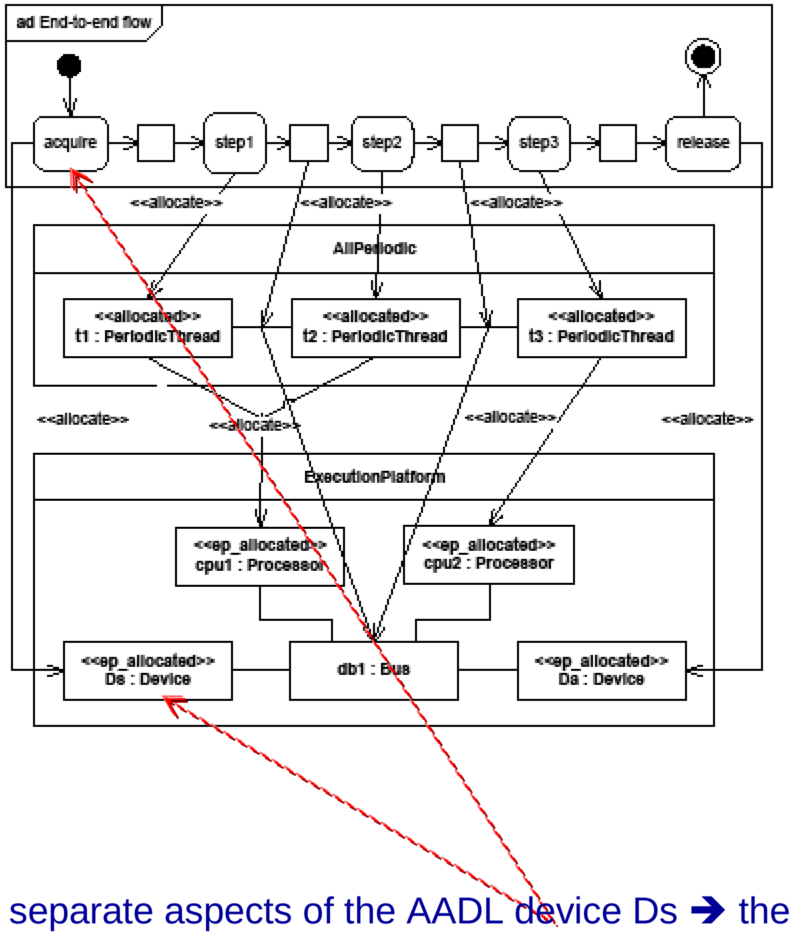
Application

(UML activity diagrams)

Software execution platform

Hardware execution platform

(Composite Structure diagrams)


ad End-to-end flow
acquire
step1
step2
step3
release
<<allocate>>
AllPeriodic
<<allocated>> t1 : PeriodicThread
<<allocated>> t2 : PeriodicThread
<<allocated>> t3 : PeriodicThread
ExecutionPlatform
<<ep_allocated>> cpu1 : Processor
<<ep_allocated>> cpu2 : Processor
<<ep_allocated>> Ds : Device
db1 : Bus
<<ep_allocated>> Da : Device

In MARTE ➔ split the software level in 2 levels:
-pure applicative part with only causality relations and no scheduling information
-The software execution platform describes schedulable resources

AADL hardware architecture

2 separate aspects of the AADL device Ds ➔ the UML action acquire and the hardware resource Ds, combined using the stereotype <<allocate>>

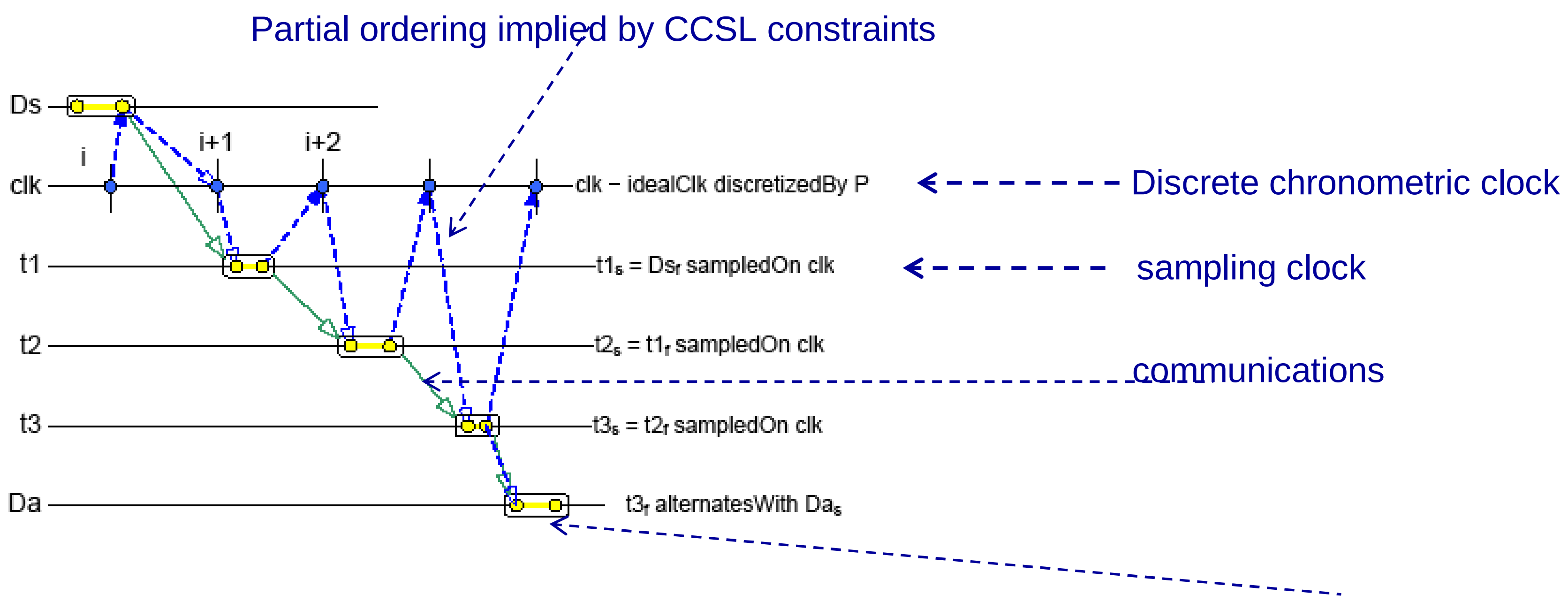
Modeling only periodic tasks with MARTE

- To model AADL communication semantics we use MARTE Clock constraints that extend UML constraints with CCSL (Clock Constraint Specification Language) to relate different parts of the model
- When we only consider periodic threads that are in phase →two kinds of CCSL constraints: ***alternatesWith*** represents event alternation and i***sPeriodic-On*** makes two events periodic relatively to each other.
- we can use these constraints to **link together actions and threads**
- Every time a **thread** is dispatched, the corresponding **action** starts:
  - t1 ***alternatesWith*** step1.
- This constraint also implies that the action must finish before the next dispatch, which is also the semantics of AADL threads. Then, we use the relation ***isPeriodicOn*** to model the periodicity of the threads.

Mixing periodic/aperiodic tasks
CCSL constraints derived from the MARTE model

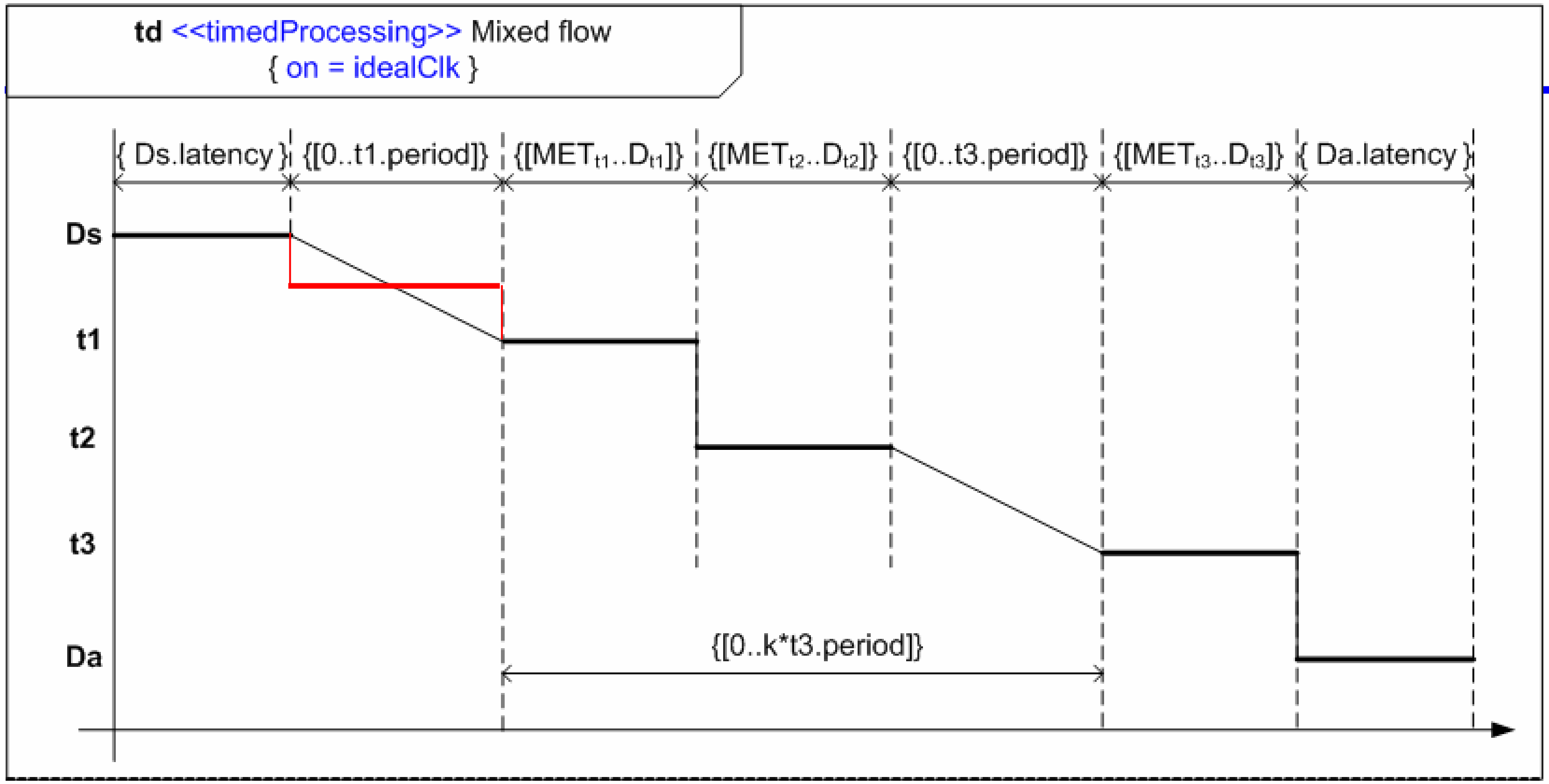
- A task sends a data to an aperiodic task, the communication is purely asynchronous. The relation ***alternatesWith*** can model asynchronous communications.

- an aperiodic task sends a data to a periodic case. In that case, the communication is synchronized, the data will be sampled on the clock of the receiving task, the *sampling clock* ➔ ***sampledOn***



Derived constraints when the three threads t1, t2 and t3 are synchronus
(3 sampled communications are followed by an asynchronus communication
From thread t3 to device Da

Timing analysis results: UML Timing Diagrams



End-to-End Flow Latency = Ds.latency + flow latency + Da.latency

Flow latency$_{\text{Worst-Case}}$ = t1.period + k1 * t3.period + t3.deadline

Flow Latency$_{\text{Best-Case}}$ = t1.period + k2 * t3.period + t3.MinExecTime (k2≤k1)

Latency jitter = t3.deadline – t3.MinExecTime + (k1-k2)*t3.period

3 - Efficient embedded runtime systems through port communication optimization (1)

- AADL supports **a port communication model**
  - includes queued event and message communication,
  - unqueued data stream communication.
- The AADL standard provides well-defined execution and communication semantics for threads in the form of a **input-compute-output model**.
- Data stream communication between periodic threads can be specified to occur
  - as mid-frame communication (immediate data port connections), or
  - as phase-delayed communication (delayed data port connections).
- A **runtime system** that implements the **execution and communication semantics of AADL** must ensures that the deterministic sampling property is preserved.
  - it must use **double buffering** as necessary and
  - perform transfer of data between buffers in such a manner that **deterministic sampling and processing is achieved.**
- It is desirable to **minimize the number of times data must be transferred** between separate buffers
  - if possible to reduce the implementation of the communication to **a shared variable solution**;

Efficient embedded runtime systems through port communication optimization (2)

- An analytical framework that allows to **optimize the implementation of a port-based communication** model by **minimizing the number of separate buffers**
  - necessary to ensure the deterministic sampling properties desirable for control system application
  - and expressed by the semantics of AADL port connections
- Reduce number of variables/buffers necessary by
  - looking at the life span of the content of a port variable/buffer does not overlap with another the two can be mapped into the same memory location.
- The analytical framework based on of buffer life spans allows us to determine
  - how many buffers/variables are necessary and
  - when data must be transferred between them to ensure the input-compute-output model and deterministic sampling.
  - this information can be embedded in an **automatic code generator for a runtime executive** that is tailored to the architecture expressed in the AADL model.

4 - Formal verification of the architecture models
***a-** AADL mode change protocol*

- Changing the operating mode results in
  - modifying the architecture of the system so as **to match the set of functions associated with the new mode**.
- AADL support the specification of multimodal architecture
- AADL proposes a mode change protocol that defines **how and when the configuration of the system is modified** when processing a mode change request.

Formal verification of the architecture models
Modes in AADL (1)

- The architecture of a system is modeled in AADL as a **hierarchy of components**.
- Each component is described as a composition of smaller components.
- The system is composed of the platform (processors,
- memories, buses, devices) and the software (processes, threads, data, sub-programs).
  - Each of these components has **one or more operating modes**.
- A mode defines **a configuration of the component**.
- A configuration is defined by the set of active subcomponents, the topology of internal connections, and specific values for modes dependant properties.

Formal verification of the architecture models
Modes in AADL (2)

- The global state of a system is defined by a vector of modes called System Operational Modes (SOM).
  - Each element of the vector represents the mode of a component.
- The initial SOM is defined
  - by the initial mode of the root component and
  - the initial modes of its recursively active subcomponents.
- The result of a mode transition is a modification of elements involved in the mode switch.
- Performing a mode change imposes to cross a transient stage between two stable configurations.
- The AADL standard defines precisely the instant where the changes of the configuration have to be realized

Formal verification of the architecture models
Modes in AADL (3) / the AADL mode change protocol

- The SOM change protocol comprises three steps:
  - (1) leaving the old SOM;
  - (2) performing the SOM change (the threads to be removed execute their deactivation entrypoint, the threads to be added execute their activation entrypoint);
  - (3) entering the new SOM.
- To ensure the determinism of the mode change, the instants where a thread is activated / deactivated or the instant where a connection is added / deleted cannot be randomly chosen.
- During the SOM transition period,
  - some processing remaining from the old SOM,
  - some processing related to the SOM change process,
  - and some processing attached to the new SOM, are performed.
- By synchronizing the processing of configuration changes with the hyperperiod of the impacted critical threads, this protocol aims at preserving the determinism of inter-thread communications

Formal models (1)
Formalization in Time Petri Nets

- The translation process from an AADL specfcication to a TPN is composed of two steps.
  - First, the specification of the SOM transition system is extracted from the hierarchical AADL model.
    - The output of this step is a labelled transition system (LTS). Each state is a SOM of the system.
    - Each transition is labelled with the name of an event, the occurrence of which triggers the mode change from the source state to the target state.
  - Second, a TPN model is built from the LTS, by composing elementary patterns.
- The resulting model can be used to simulate and analyze the timed modal behaviour of AADL specifications.

Formal models (2)
Formalization of the mode transition in TLA

- In our model, we distinguish three types of threads, normal threads that are stopped on a mode switch, critical threads whose execution must be terminated at the time of the mode transition, and zombie threads that are allowed to end their execution in the new mode.
- The set of available mode transitions and the set of active threads for each mode are also given as constant functions.
- We describe in TLA a transition system
  - The set of possible states is defined by a set of variable
  - and the evolution by a set of transitions.
- The state is define by the following variables:
  - ***currentMode***, ***currentEvent***, ***currentThreads***, and ***zombies***.
  - The set ***currentThreads*** represent the executing threads,
  - ***currentEvent*** store the name of the event that triggered a transition,
  - and ***zombies*** is the set of threads of the old mode still executing after a mode transition.

b- AADL Execution Semantic Transformation for formal verification
Context

- In order to validate formal properties on these AADL models, we must use **formal frameworks** which provide diagnostics on the system.
- This goal is achieved if we have a powerful set of tools which offer the capacity to **transform the AADL model into formal model**.
  - Source language : AADL execution model
  - Target language : IF Language (enables a model whose basic unit is a system ("system") to be described).
- This system is composed of active entities, or "processes".
  - entities interact with messages named "signal" in an asynchronous way in a parallel execution.
  - "processes" are timed automata communicating by message buffers with urgencies.
- The communication channels are managed with buffers in "signalroute" and parameterized to specify:
  - Communication type (multicast, unicast, peer)
  - Media quality (applicable, lossy)
  - Buffer type (fifo, multiset)
  - Communication urgency (urgent, delay, rate)

AADL Execution Semantic Transformation for formal verification

AADL model transformation - Structure

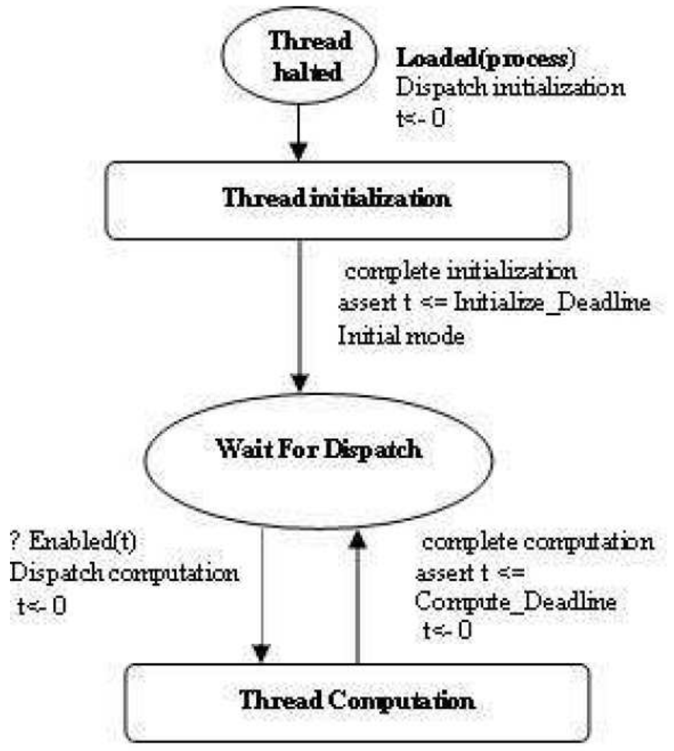
- Basic concept equivalences for **transformation rules specification**:
  - ***SystemImpl*** which represents the component root of the models is transformed into a **IF "system“,** root element of an **IF model**.
  - ***DataType*** is transformed into a **IF type**
  - The ***Process*** and ***Thread*** are concepts which require detailed attention because they incorporate the behaviour in AADL. Inevitably the processes have to contain at least one Thread or ThreadGroup, the transformation transforms the Thread into **IF process** according to the approach detailed in the following paragraph.
- An AADL ***ThreadType*** element has properties and ports.
- Ports provide communication mechanism for ***Event***, Data or ***EventData***.
  - These various types of communication are transformed into type **IF "signal"** and managed by an **IF "process"** whose execution is decoupled from the ***Thread*** behaviour.

AADL Execution Semantic Transformation for formal verification
AADL model transformation - Behaviour

- We chose to provide a behaviour annex based on NTIF language.
- NTIF and IF language are closed. However NTIF provides high level instructions, like ***Select*** which authorizes several transitions going to various target state.
- NTIF transition transformation cannot be one IF transition.
- The Subprogram invocation has a semantic defined by the property ***Server_Call_Protocol*** which can take several values:
  - synchronous (HSER),
  - half synchronous (LSER)
  - or asynchronous (ASER)
- Within this framework, a ***Subprogram*** component is transformed into **IF "process"** which has the same parameters as AADL ***Subprogram***.

AADL Execution Semantic Transformation for formal verification
AADL model transformation - Execution semantics

The execution model is based on the control automaton



This represents an abstraction of the complete automaton
To improve the execution model transformation → add states and IF clocks for transition guards

Conclusions on various contributions

- Model-driven engineering of real-time and embedded applications requires a modeling language providing concepts to capture at a high level of abstraction their qualitative and quantitative features.
- The UML profile for MARTE provides such concepts and is suited for real-time applications modeling.
- MARTE can be used to model mixed systems with both periodic and aperiodic tasks, which is a big issue while modeling embedded systems.
- Comparison of MARTE and AADL: highlighting MARTE capabilities to make the computation formulas explicit.
- Several analysis frameworks. → to determine the impact of different implementation choices for communication on the latency characteristics of data stream being communicated. It establishes criteria for data integrity and for deterministic data communication.
- A formal semantics given as a translation algorithm computing a TPN →simulating the mode change behaviour of an AADL specification.
- An AADL model transformation that provides a formal model for model checking activities → to describe transformation rules distributed in three catagories: rules of component transformations of the AADL language, rules of behaviour transformations of the AADL threads, rules of operational semantics transformations for threads.