Parallel Processes with Real-Time and Data: The ATLANTIF Intermediate Format

Jan Stöcker, Frédéric Lang, and Hubert Garavel

INRIA Grenoble Rhône-Alpes / LIG
Montbonnot Saint-Martin
VASY project team

http://www.inrialpes.fr/vasy

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche GRENOBLE - RHÔNE-ALPES

VASY

Context and objective

- Design of realistic industrial applications (e.g., embedded systems)
- Formal methods integration: from modeling to formal verification
- Need for formal and concise languages to represent:
  - **Complex data**: arrays, unions, lists, etc.
  - **Control & concurrency**: events, synchronization, communication, dynamic process activation, etc.
  - **Real-time**: delays, urgency, latency, etc.

Existing languages & models

- Process algebras
  - Extensions of CCS and ACP: aimed to study theoretical problems
  - Extensions of CSP and of the LOTOS ISO standard (T-LOTOS, RT-LOTOS, ET-LOTOS, …): application oriented, but with steep learning curve
  - Emergence of new languages: E-LOTOS, LOTOS NT
  - But few verification tools exist
- Graphical models
  - Timed automata, time Petri nets, …
  - Existence of tools (e.g., Uppaal, Tina, Red, …)
  - But hard to model realistic applications

E-LOTOS example

task A

"red button", robots stop

specification *Two_Robots* **is**

gates *Next*, *RB*

behaviour (loop wait *3*; *Next* **endloop**

|[*Next***]|**

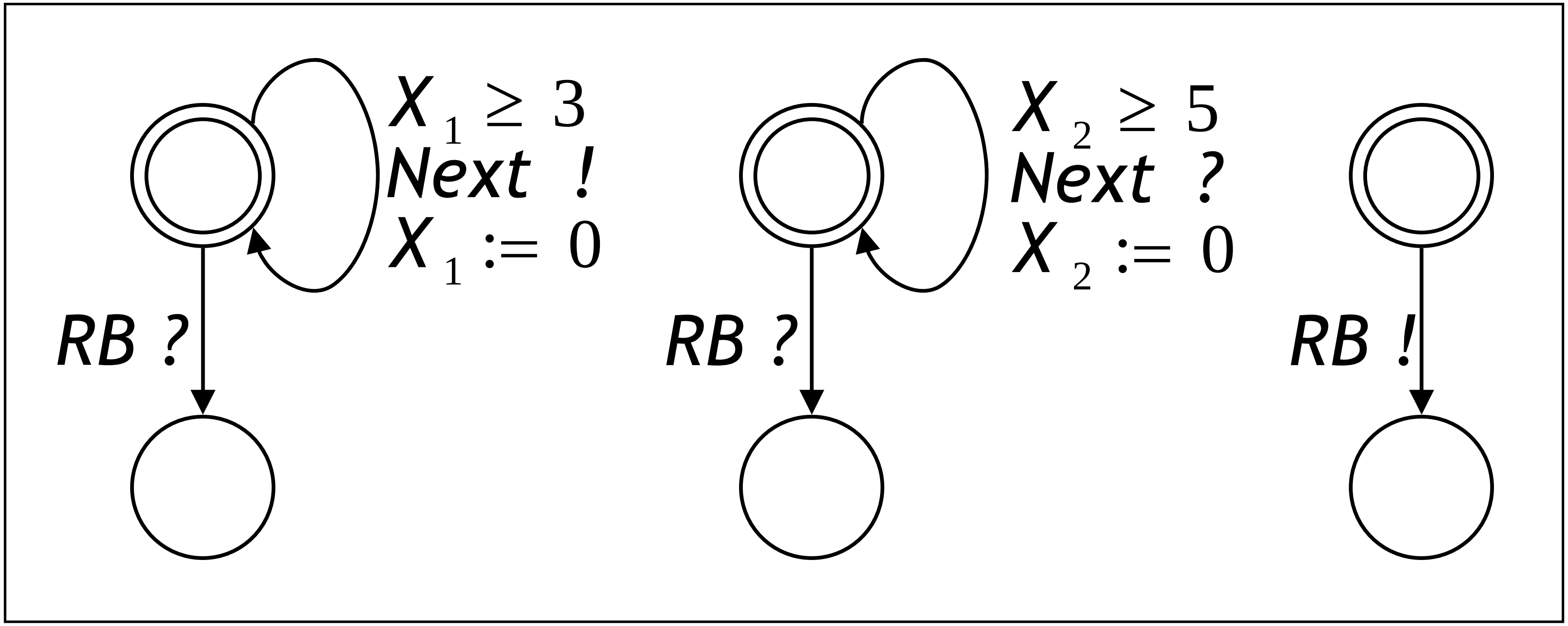
loop wait *5*; *Next* **endloop) [>** *RB*; **null**

endspec

task B

tasks finished, next product arrives

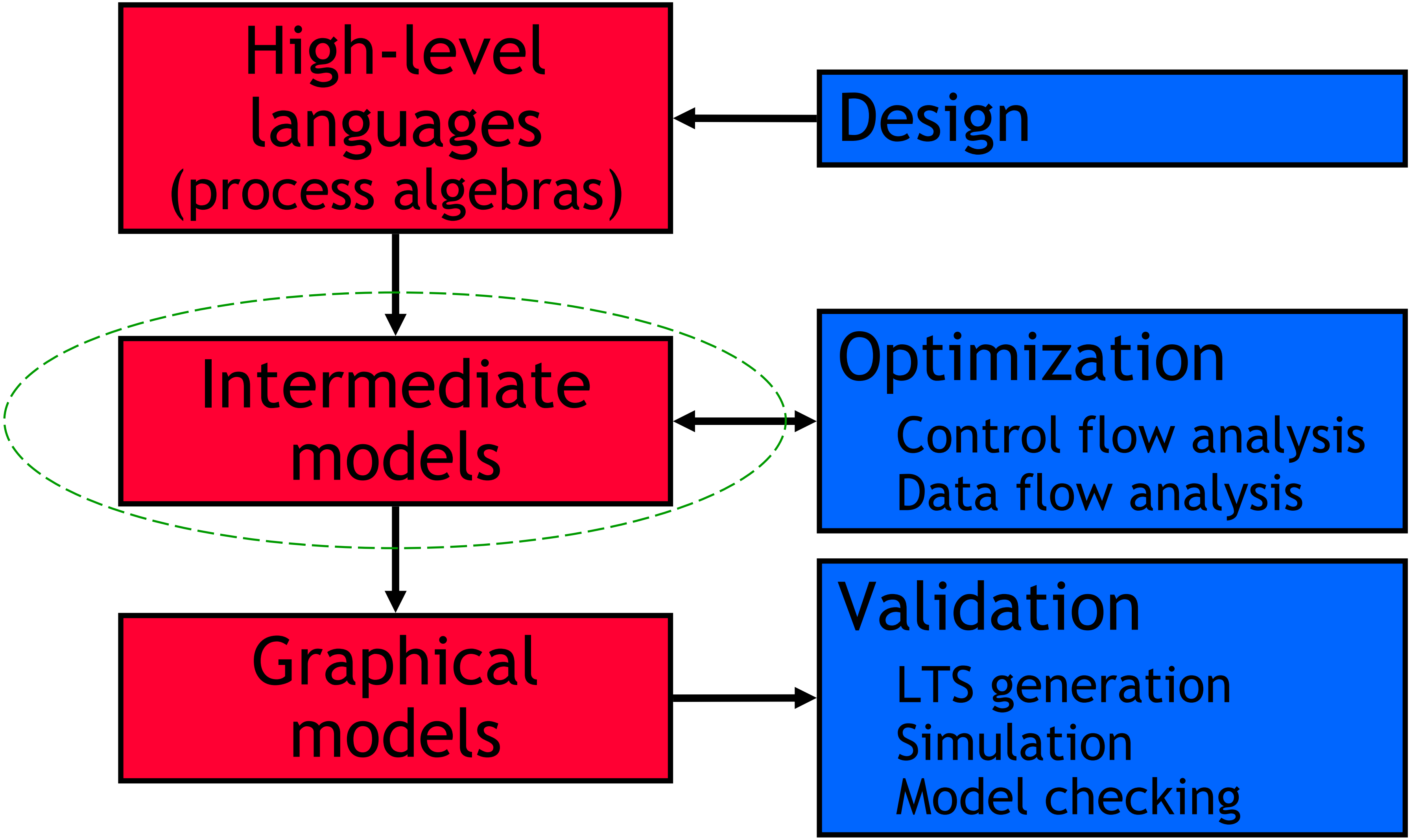
Timed automata example



Our goal

- Make connections between
  - high-level languages convenient to model realistic applications
  - and graphical models for which efficient verification tools exist
- Need for an *intermediate model*, that
  - concisely expresses high-level constructs
  - preserves the semantics
  - allows automated translations to graphical models
- This talk: define a suitable intermediate model named ATLANTIF

Intermediate models



Existing intermediate models

- MoDeST [D'Argenio-Hermanns-Katoen-Klaren-2001]
  - Probabilistic model without concurrency
- BIP [Basu-Bozga-Sifakis-2006]
  - Concurrent model, restricted data manipulation
- NTIF [Garavel-Lang-2002]
  - Manipulation of complex data structures
  - Sequential processes without concurrency or time
- Fiacre [Berthomieu-Bodeveix-Farail-et-al-2008]
  - Pivot language in translations to tools (CADP, Tina)
  - Real-time syntax restricted to TPN-like constructs

The ATLANTIF intermediate model

An ATLANTIF program consists of:

- A set of data type and function definitions
- A set of hierarchical real-time asynchronous sequential processes, named units
- A set of synchronizers defining the parallel composition, process activations, and synchronizations between units

An ATLANTIF program

module *(name)* **is**

[(no | discrete | dense) time]

real-time option

Type and function declarations

Synchronizers

init *unit_1, unit_2, …, unit_x*

initially started units

Units

Subunits

end module

Data types and functions

- Inherited from the NTIF model
- Data types
  - predefined: **int**, **bool**, **float**
  - user-defined: enumerations, structures, arrays, lists, trees, etc.
  - constant and parameterized constructors
- Functions
  - predefined: +, -, =, ≤, >, etc.
  - user-defined: typed parameters, typed return value, sequential statement

Units

- Definition of sequential behaviour
- Extension of NTIF processes with real-time constructs and hierarchical structure

unit *(name)* **is**

variables $V_1 : T_1$ [:= E_1], ..., $V_n : T_n$ [:= E_n]

from *state_1* <*action*>

discrete state

...

from *state_m* <*action*>

multibranch transition action

Subunits

subunits can access the variables V_1, ..., V_n

end unit

Actions

A ::= **null**

| **wait** E ← *delay action*

| $V_0,\ldots,V_n := E_0,\ldots,E_n$

| $V_0,\ldots,V_n :=$ **any** $T_0,\ldots,T_n$ [**where** E]

| **reset** $V_0,\ldots,V_n$

variable manipulation

| $G\,O_1\ldots O_n$ [[**must** | **may**] **in** W]

gate communication *with time restriction*

| **to** s ← jump

| $A_1 ; A_2$

| **select** $A_0 [] \ldots [] A_n$ **end**

| **case** E **is** $P_0 \to A_0 \mid \ldots \mid P_n \to A_n$ **end**

| **while** E **do** A **end**

composition

two extensions w.r.t. NTIF action syntax

Communication timing constraints

- Optionally, gate communication actions have a *time window W*:

$$W ::= [E_1,E_2] \mid\,]E_1,E_2] \mid [E_1,E_2[\mid\,]E_1,E_2[\mid [E_1,\ldots[\mid\,]E_1,\ldots[\mid W_1 \textbf{ or } W_2 \mid W_1 \textbf{ and } W_2$$

  – intervals
  – unions and intersections of intervals
- A keyword defines the behaviour at the end of the time window
  – “**may**”: time can elapse further
  – “**must**”: time elapsing blocks

Concurrency and synchronizations

- A subset of the units executes (asynchronously)
- Gates are synchronized following *synchronizers*

synchronizer / gate name

$$
\begin{aligned}
S \ ::= \ & \textbf{sync } G \\
& \left[: \left(\textbf{silent} \mid \textbf{hidden} \mid \textbf{urgent} \mid \textbf{visible}\right)\right] \\
& \textbf{is } C \\
& \left[\textbf{stop } u_1, \ldots, u_m\right] \\
& \left[\textbf{start } u_1', \ldots, u_n'\right] \\
& \textbf{end sync}
\end{aligned}
$$

synchronization formula

optional: visibility

optional: unit starting and stopping

$$
\begin{aligned}
C \ ::= \ & u \\
\mid \ & C_1 \textbf{ and } C_2 \\
\mid \ & C_1 \textbf{ or } C_2 \\
\mid \ & N \textbf{ among } \left(C_1, \ldots, C_n\right) \\
\mid \ & \left(C_0\right) \\
N \ ::= \ & n \\
\mid \ & N_1 \textbf{ or } N_2 \\
\mid \ & \left(N_0\right)
\end{aligned}
$$

Gate visibility

- The visibility of a gate *G* defines how it appears in the semantics:
  - **“visible”**: transitions labeled with G and offers
  - “**hidden**”: internal transition (τ)
  - “**urgent**”: internal transition (τ), time is blocked when synchronization is possible
  - “**silent**”: no transition in the semantics
- By default, gates are *visible*

Examples of synchronizers

Every synchronization formula defines one or several sets (called synchronization sets) of units that may synchronize on the gate *G*

- Example 1: **Competition**
  - "**sync** *G* **is** u_1 **and** (u_2 **or** u_3) **end sync**" expresses synchronization of u_1 with either u_2 or u_3
  - *Synchronization sets:* $\{u_1, u_2\}, \{u_1, u_3\}$
- Example 2: **Multiway synchronization** (*n* processes synchronizing altogether)
  - "**sync** *G* **is** u_1 **and** u_2 **and** u_3 **end sync**" expresses synchronization of u_1, u_2, and u_3 altogether
  - Synchronization set: $\{u_1, u_2, u_3\}$

Examples of synchronizers

- Example 3: **Generalized parallel composition**
  - **"sync** G **is** $(2$ **or** $3)$ **among** (u_1, u_2, u_3) **end sync"**
  - Synchronization sets: $\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_3\}, \{u_1, u_2, u_3\}$
- Example 4: dynamically **stopped** and **started** units
  - **"sync** G **is** u_1 **and** u_2 **stop** u_1, u_2 **start** u_3, u_4 **end sync"**
  - Synchronization set: $\{u_1, u_2\}$
  - At synchronization on G, u_1 and u_2 are stopped and u_3 and u_4 are started

Semantics

- *Static* semantics imposes restrictions on the definition of a specification:
  - typing
  - variable initialization
  - variable access conflicts, etc.
- *Dynamic* semantics rules define a **timed labeled transition system**, which satisfies several "good" properties:
  - time additivity
  - time determinism
  - maximal progress of urgent actions

Example

module *Two_Robots* **is dense time**

sync *Next* **is** *Rob1* **and** *Rob2* **end sync**
sync *RB* **is** *Rob1* **and** *Rob2* **stop** *Rob1*, *Rob2* **end sync**

init *Rob1*, *Rob2*

unit *Rob1* **is**
from *Task_A*
    select wait 3; *Next* [] *RB* **end**; **to** *Task_A*
end unit

unit *Rob2* **is**
from *Task_B*
    select wait 5; *Next* [] *RB* **end**; **to** *Task_B*
end unit

end module

Translation to Uppaal

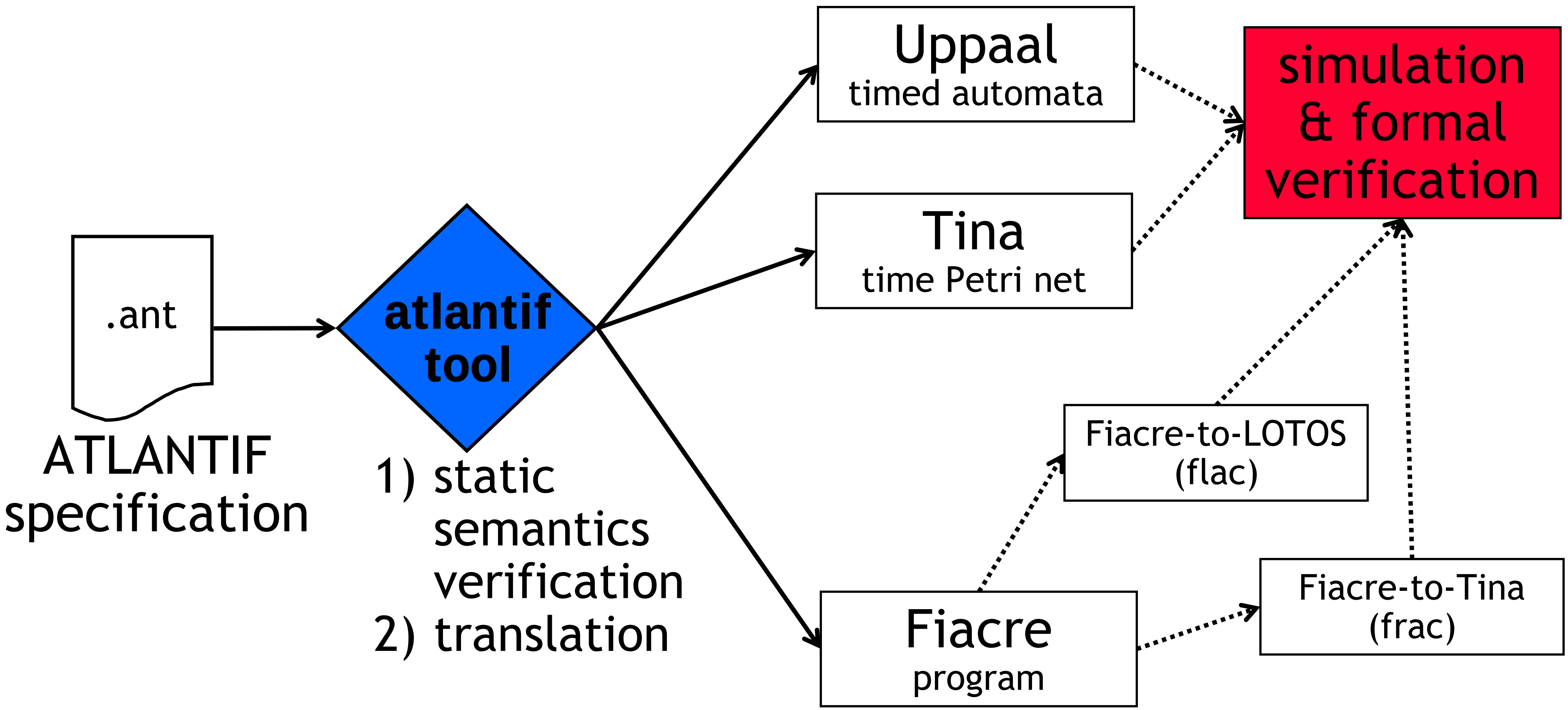
ATLANTIF construct	translated to (Uppaal)
module	network of timed automata
synchronizer each synchronization set containing 1 or 2 units each synchronization set containing n > 2 units	one or several channels: one binary channel (n – 1) binary channels to emulate multiway synchr.
unit	one timed automaton
discrete state	location
multibranch transition	one transition per path
gate communication	action label on transition
timing constraints	guards and invariants
communication offers	emulated by global variables

Translation to Tina

ATLANTIF construct	translated to (Tina)
module	one time Petri net (TPN)
synchronizer	no direct translation
unit	subset of the TPN
discrete state	place
multibranch transition	first one transition per path, then multiplication and fusion with synchronizing transitions
gate communication	action label on transition
timing constraints	auxiliary transitions, priorities, and inhibitor arcs
data manipulation, communication offers	translated to C functions

Tool overview

- ~18,000 lines of code in LOTOS NT, C, Syntax



Conclusion and future work

- Specifications from high-level languages can easily be represented in ATLANTIF:
  - intuitive textual syntax, easy to read and to write
  - process-algebra-inspired high-level constructs for synchronization, choice, etc.
- ATLANTIF is linked to formal verification tools:
  - translations to Uppaal TA and Tina TPN
- Future work:
  - extend the subsets of ATLANTIF understood by the translator tool
  - explore automated translations of process algebras