Modern languages for modeling and verifying asynchronous systems

Damien THIVOLLE

Advisors:

Dr. Valentin CRISTEA (UPB)

Dr. Hubert GARAVEL (INRIA)

PLAN

- Introduction
- Formal verification of GALS systems
- Formal verification of BPEL Web services
- Conclusion

Overview

- Objective is to create connections between:
  - modern modelling languages (compatible with the Model-Driven Engineering paradigm), and
  - formal verification tools (typically CADP)
- How?
  - By creating connections at a language level, using semantic transformations

Why?

- Complementarity at different levels:

	MDE Languages	Formal Methods Languages
Syntax	graphical, attractive	textual, unattractive
Semantics	informally defined	mathematically defined
Industrial acceptance	almost standard	weak

- MDE languages lack verification tools

Applications

- TFTP case study
  - Given by Airbus
  - Verification of a variant of the TFTP protocol used for the A350
  - Specification written in SAM, modelling language from Airbus
- BPEL
  - Language for describing the logic of Business Processes and exposing their interface as Web Services
  - MDE-oriented (graphical syntax that fits the MDE paradigm)

Model-Driven Engineering

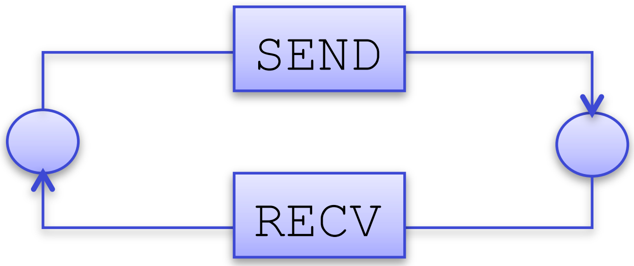
- Development paradigm where everything is a model:
  - Application, requirements, executable code…
- Environments like Eclipse, Netbeans provide necessary tools:
  - Model transformations, editors, code generators…
- Adopted in the industry (TOPCASED project, with Airbus, Thales, EADS…)
- Suited to dedicated languages (DSLs)

CADP

- Formal verification toolbox (http://vasy.inria.fr/cadp)
- Systems specified in process algebras (LOTOS / LOTOS NT):

```
process P [SEND, RECV:any] is
   SEND; RECV; P [SEND, RECV]
end process
```

- Process algebra code compiled into transition systems:



- Model checking = evaluation of temporal logic formulas (requirements)

```
[true* . SEND . (not RECV)* . SEND] false
```

LOTOS NT (1/2)

- Simplified version of E-LOTOS (Sighireanu-99)
- Function definitions:

```
function funcName (in ArgIn1:T1, … ,in ArgInm:Tm,
                   out ArgOut1:T'1, out ArgOutn:T'n) is

end function
```

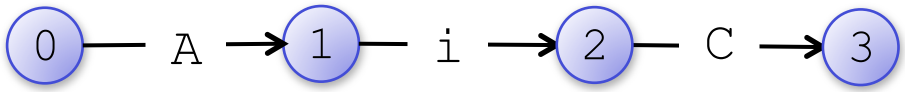
- Type definitions (with constructors):

```
Type NatList is
  Cons (head:Nat, tail:NatList),    <- definition
  Nil
end type
…
    Cons (1, Cons (2, Cons (3, Cons (4, Nil))))    <- instantiation
```

LOTOS NT (2/2)

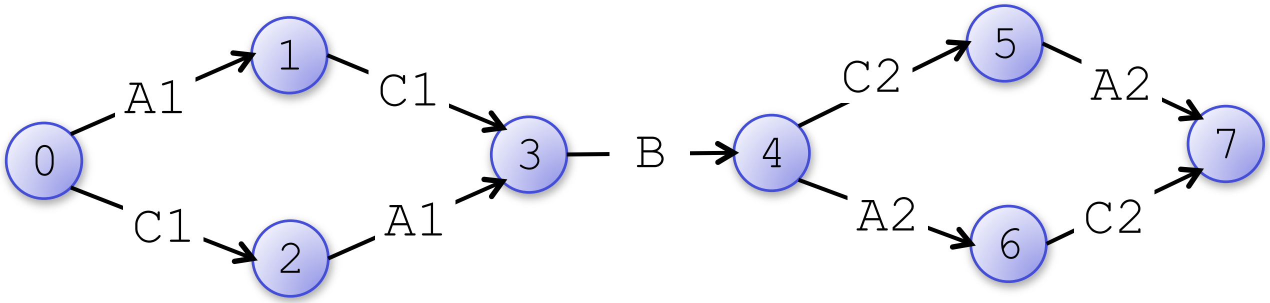
- `hide` **operator**

```
hide B in
  A; B; C
end hide
```



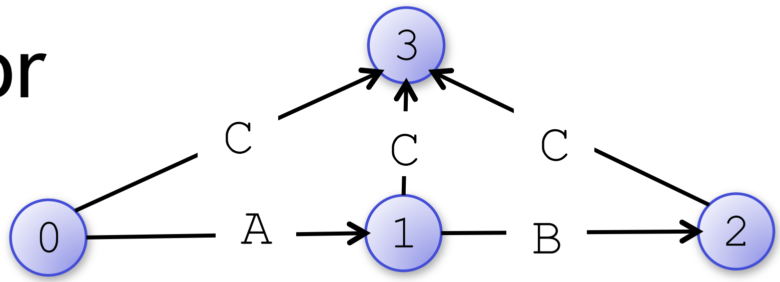
- `par` **operator**

```
par B in
  A1; B; A2
||
  C1; B; C2
end par
```



- `disrupt` **operator**

```
disrupt
  A; B
by C
end disrupt
```

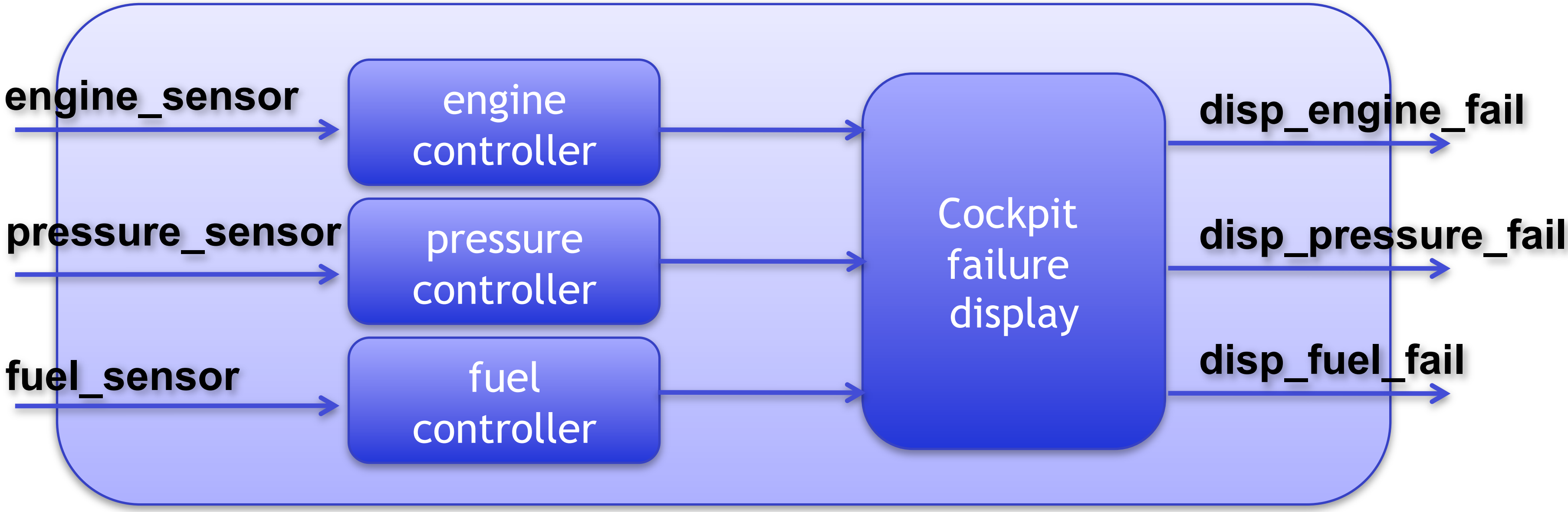


Verification of GALS Systems

Synchronous languages

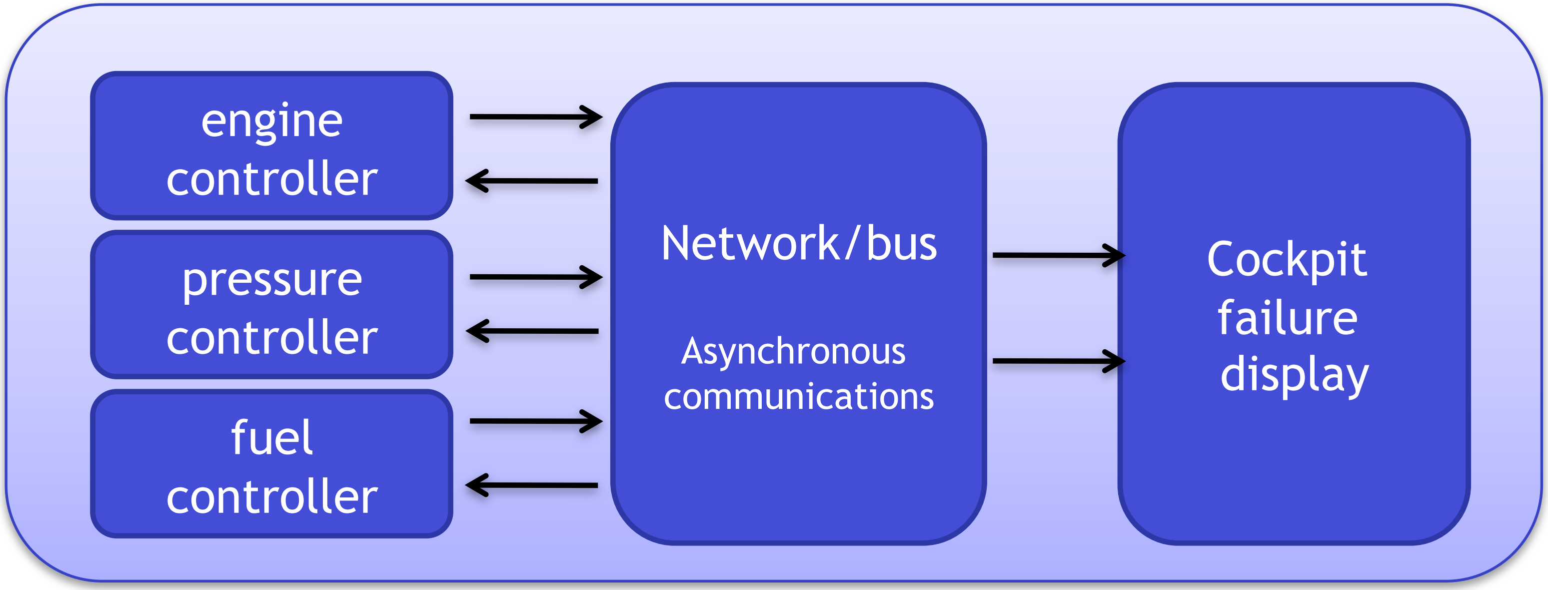
- Synchronous systems receive a set of inputs and reply a set of outputs
- They are deterministic and the computation of the outputs is intantaneous
- For programming these systems, synchronous languages are used:
  - ESTEREL
  - SCADE/LUSTRE
  - SIGNAL
- Many « synchronous » tools for verification

Synchronous paradigm



- One function = one cable/wire
- In modern designs (car, plane, train), too many wires needed

GALS Paradigm



- GALS = Globally Asynchronous Locally Synchronous
- One bus/network = many functions (Fly-by-wire, X-by-wire)
- Problems:
  - Verification of complex of communication protocols (Toyota ABS recall)
  - “synchronous tools” not suited to asynchronous communications

Related work

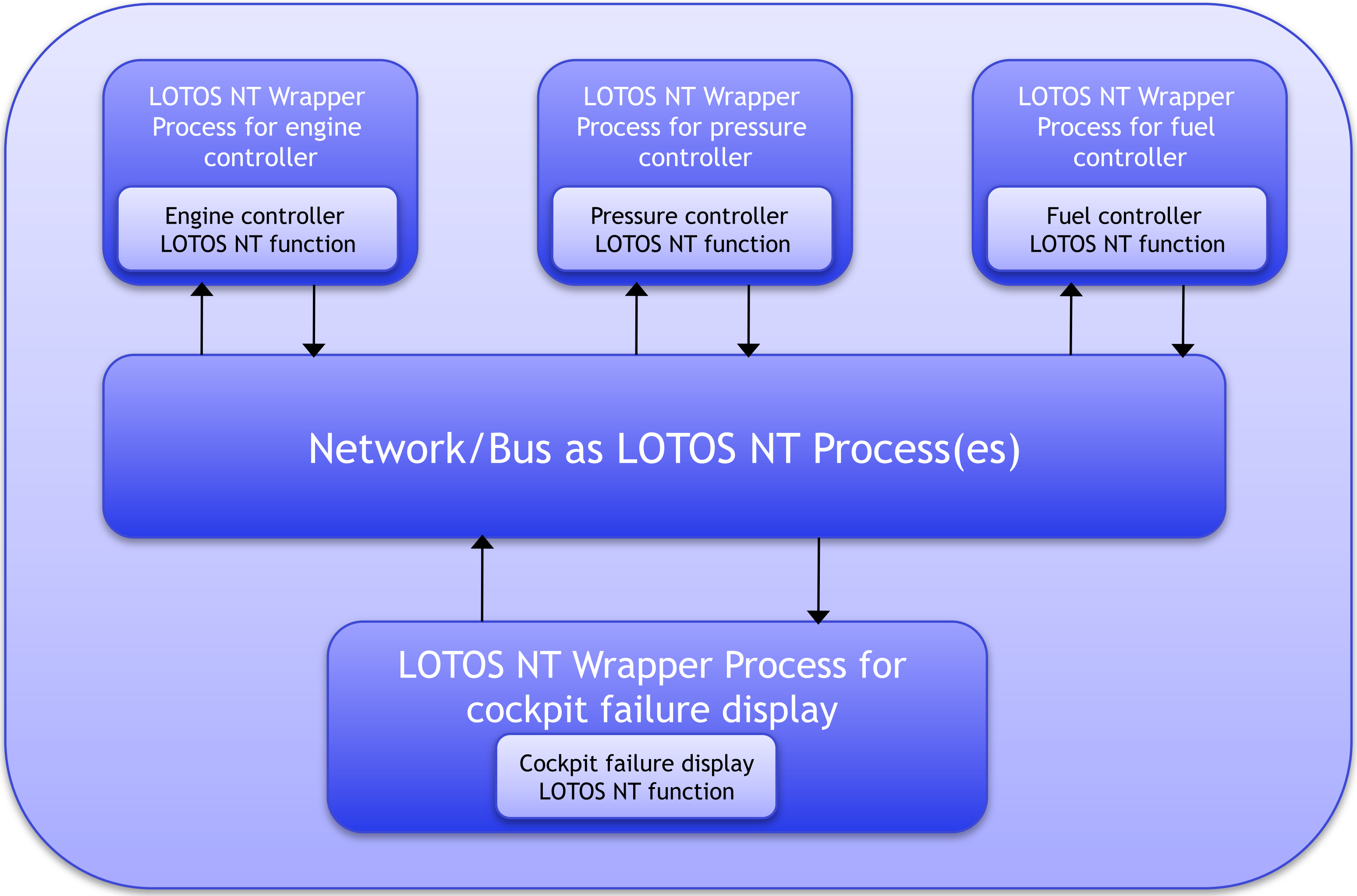
- Exclusively from the synchronous community
- Attempts to model GALS systems:
  - with synchronous languages (proved possible by Milner but cumbersome)
  - By adding new operators to synchronous languages to introduce a degree of asynchrony
- A problem remains, synchronous tools not made to handle asynchrony (lack of optimizations for interleaved semantics)
- Severely limits the size of verifiable systems

Our method (1/2)

- Garavel-Thivolle-09, proceedings of SPIN'09
- Each synchronous component is a function:
  - Inputs: current state and input values
  - Outputs: next state and output values
- We encode that function in LOTOS NT:

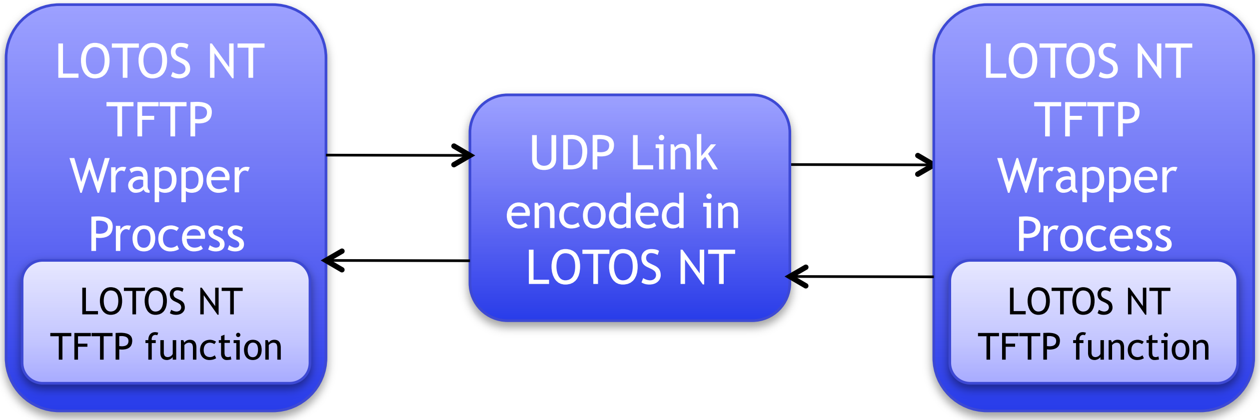
```
function transition (in state:State, in input1:T1…
                     out nextState:State, out output1:T'1…) is
    …
end function
```

Our method (2/2)



Case-study from Airbus

- TFTP variant written in SAM, a DSL from Airbus, and used for the upcoming A350 (plane-aiport communications)
- TFTP protocol entity encoded as SAM program: 7 states, 39 transitions
- GALS system: 2 TFTP protocol entities connected asynchronously by a UDP link



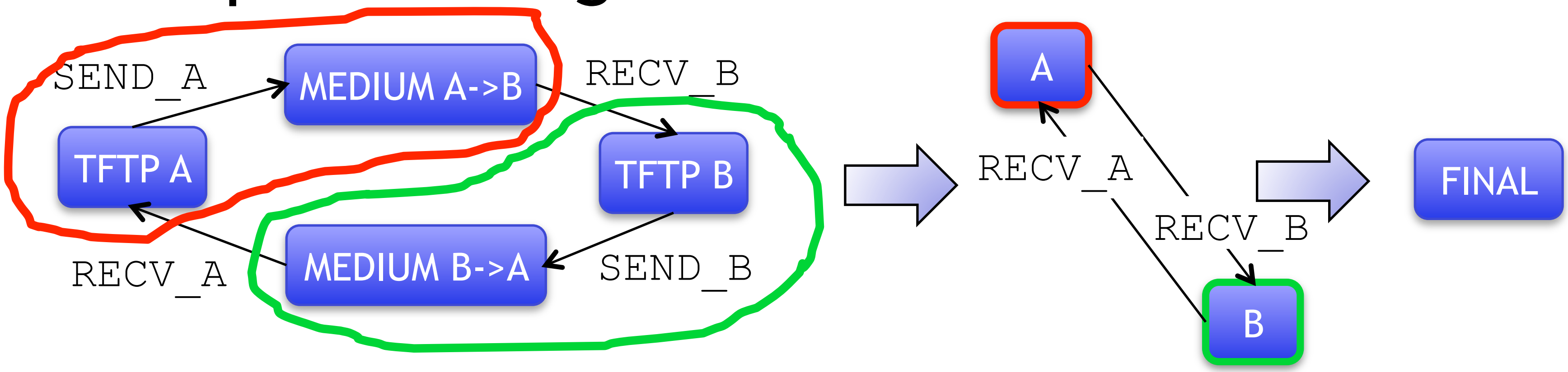
- Requirements expressed as temporal logic formulas (29 in total)

TFTP Wrappers

- Simple TFTP Wrapper
  - No real TFTP messages, straightforward asynchronous connection of outputs of one entity to the inputs of the other (and vice versa)
  - Rapid implementation
  - Followed Airbus recommendations (head-to-tail)
  - Enabled us to find 11 errors
- Accurate TFTP Wrapper
  - Implementation of the TFTP protocol which uses the Mealy function to dictate its behaviour
  - Enabled us to find 8 more errors

Generation issues

- Direct generation (compiling the entire specification) is not giving good results because the specification is too complex
- Compositional generation



- We tried different strategies for compositional generation

Verification results

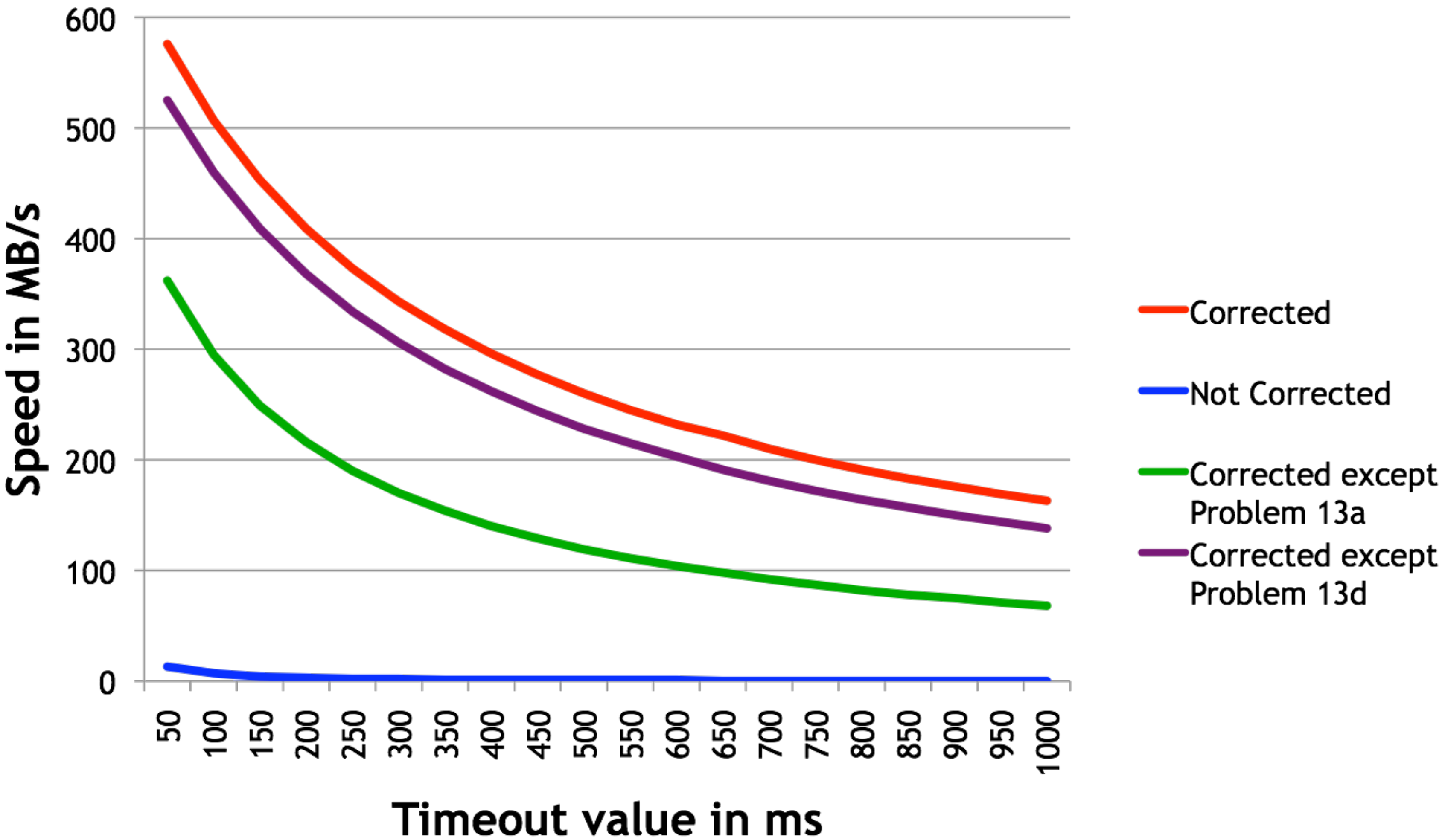
- In total, we found 19 errors
- These errors do not prevent transfers from finishing (probably why they had remained undetected)
- All these errors were acknowledged as real errors from Airbus

- Do they affect runtime performances?
  - ⇨ Simulation

Simulation

- TFTP has an error recovery mechanism which depends on waiting for timeouts and resending messages
- The errors in the TFTP automaton cause transfer to abort and restart without having to wait for timeouts
- Is an error-free TFTP automaton more efficient? With varying timeout values?
- Technical details:
  - We used Executor from CADP
  - Weights were given to transitions (1/10000 for internal errors, 1/100 for medium errors, 1 for other actions)
  - We considered a medium of 1 MB/s and data fragments of 32 KB
  - We made timeout values (length of waiting period) vary from 50 ms to 1 s

Simulation results (full duplex)



Results & Conclusion

- Results:
  - 19 errors found in the Airbus TFTP variant
  - Errors acknowledged by Airbus
  - Not critical errors but greatly affect transfer speeds (close to 0 in some cases)
- Conclusion:
  - Approach works and is efficient:
    - Allows to reuse existing « synchronous » tools for the standalone verification of synchronous components
    - Enables mixing different synchronous languages
  - Led to an on-going collaboration with Airbus

Formal verification of BPEL Web Services

Web Services

- Remote applications accessed through the Internet, and complying to a set of W3C standards:
  - Application interfaces exposed with WSDL (functions, data types of arguments)
  - Arguments (messages) encoded with SOAP
  - Data (function calls) transferred with HTTP
- Increasingly popular (W3C support)
- Used in critical systems (online payment systems for example)

Overview of BPEL

- Business Process Execution Language
- Defines an application using a Business Logic oriented language (with XML syntax)
- Exposes the application as a Web Service
- BPEL fits in MDE paradigm (Eclipse BPEL and BPMN notation)
- Inspired by two languages:
  - WSFL (IBM, workflow theory)
  - XLANG (Microsoft, process algebras, pi-calculus)
- Industrial support (Microsoft, IBM, Oracle…)

More details

- Structured-programming constructs (`if`, `while`, `for`, `sequence`…)
- Concurrency: `flow` operator and concurrent access to variables
- Communications: `receive`, `reply`, `invoke`
- Error management: fault, compensation, termination handlers
- Relation to other standards:
  - WSDL: communication links and messages definitions
  - SOAP: encoding of messages (not considered for verification)
  - XML Schema: data types definitions
  - XPath: data expressions

Related work in verification

- Workflow community (WSFL):
  - Data not considered
  - Workflow analysis (reachable or unreachable activities)
- Process algebra community (XLANG):
  - Data not considered or poorly handled
  - Not all BPEL constructs processed and no explanations
  - Translation of BPEL processes in a process algebra to enable model checking

Comparison (data)

Approach	Types	Expressions	Variables	Constants
Salaün et al.	--	--	--	--
Koshkina & Breugel	--	--	--	--
Yeung	--	--	--	--
Ouyang et al.	--	--	--	--
Qian et al.	--	--	--	--
Mateescu & Rampacek	--	--	--	--
Foster et al.	-	--	+	--
Fu et al.	-	+	+	--
Humbolt-Universität	--	--	-	--
Fisteus et al.	-	--	-	--
Nakajima	--	--	-	--
Bianculli	-	--	-	--
Moser et al.	--	-	-	--
Our approach	++	+	+	+

Comparison (behaviours)

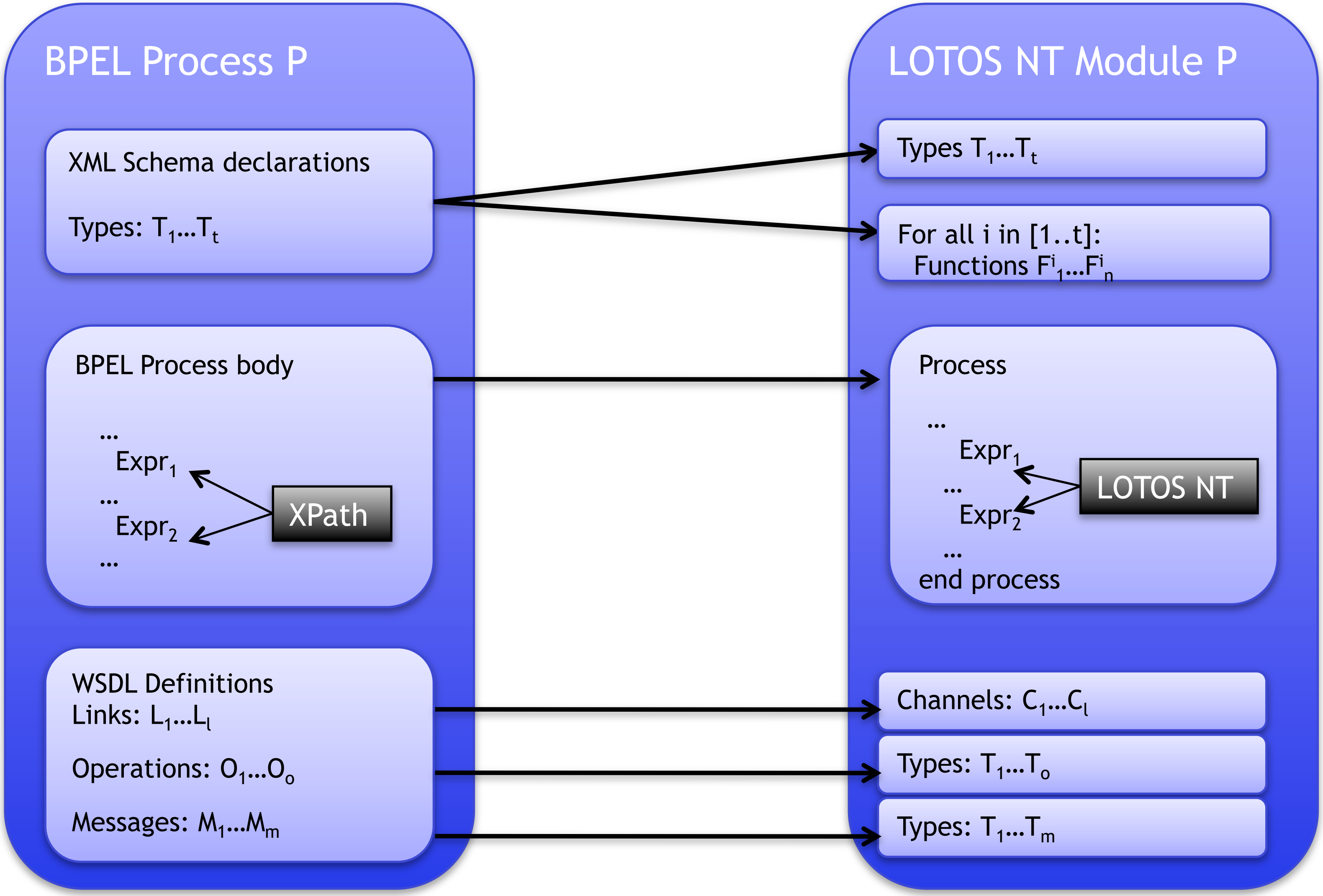
Approach	SA	exit	FH	EH	At	CL	Time	Env
Salaün et al.	+	--	--	--	--	--	--	yes
Koshkina & Breugel	+	--	--	--	--	-	--	no
Yeung	+	-	--	-	--	--	--	no
Ouyang et al.	++	++	++	+	--	++	--	no
Qian et al.	+	--	-	--	--	-	-	yes
Mateescu & Rampacek	++	--	-	--	--	--	++	yes
Foster et al.	++	--	--	-	--	--	--	yes
Fu et al.	++	--	-	--	--	-	--	yes
Humbolt-Universität	++	++	++	+	--	++	--	no
Fisteus et al.	++	--	--	--	--	--	--	no
Nakajima	++	--	--	--	--	-	--	no
Bianculli	+	-	-	-	--	-	--	no
Moser et al.	++	--	--	-	--	-	--	no
Our approach	++	++	++	+	+	++	-	yes

Legend	
SA	Simple Activities
FH	Fault Handlers
EH	Event Handlers
At	Atomicity
CL	Control Links
Env	Environment

Our approach

- Translation from BPEL to LOTOS NT to enable verification by model checking
- Heavy focus on data and data types
- Collection of 350 examples to identify useful subsets of each language
- Explanations for every construct left out (termination handlers, for example)

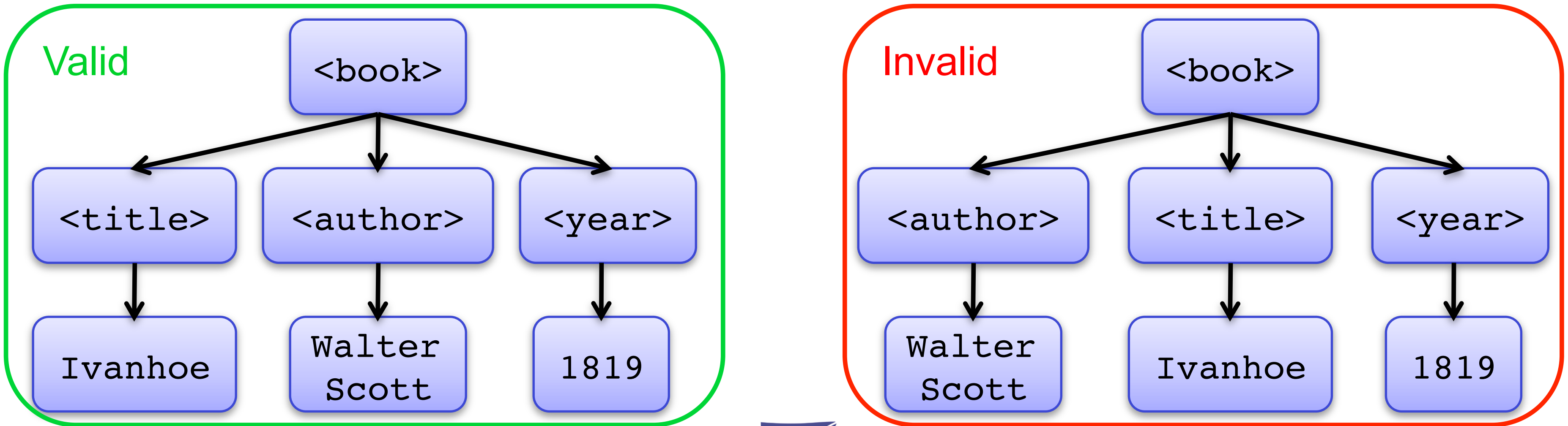
Overview of the translation



XML Schema

- An XML Document is a tree-like structure made of intermediary nodes with strings as leafs
- XML Schema express constraints on that structure

```
<complexType name=“Book”>
  <sequence>
    <element name=“title” type=“string” />
    <element name=“author” type=“string” />
    <element name=“year” type=“unsignedShort” />
  </sequence>
</complexType>
<element name=“book” type=“Book” />
```



XML Schema generic solution

- A first solution would be to encode XML values with a generic type in LOTOS NT

```
type Node is
  IntermediaryNode (name:String, nodes:NodeList),
  Leaf (content:String)
end type

type NodeList is
  Cons (head:Node, tail:NodeList),
  Nil
end type
```

- Validation functions would check whether the tree conforms to an XML Schema Type
- In terms of efficiency, this solution performs poorly: execution time + memory consumed

XML Schema optimised solution

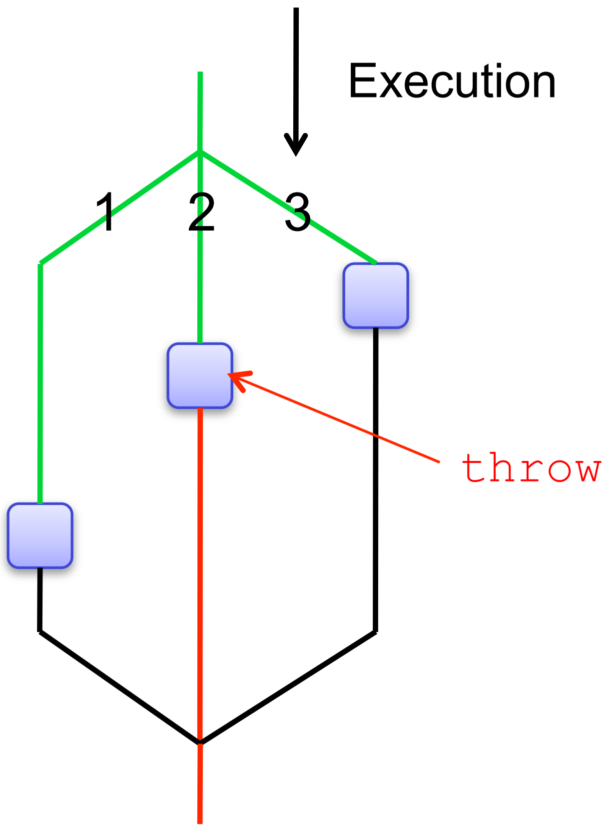
- Each XML Schema type is translated by one, optimised LOTOS NT type

```
type Book is
  Book (title:String,
          author:String,
         year:unsignedShort)
end type
```

- More complex translation
- Yields much more efficient data types

BPEL exception mechanism

- Usual operators: `<throw>`, `<catch>`, `<recatch>`



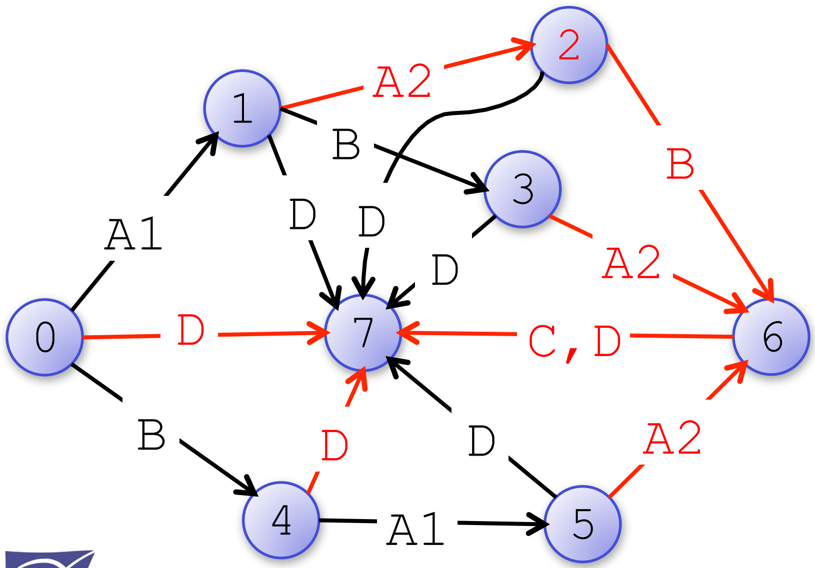
- After branch 2 stops, where do branch 1 and 3 stop?
- The BPEL standard is not explicit enough, different interpretations exist

LOTOS NT exceptions mechanism

- Exceptions can be raised but not caught (incomplete implementation)
- Effectively, the `raise` instruction is an abort
- `disrupt` is the only LOTOS NT operator we can use (but it allows for unwanted cases)

```
disrupt
   par
    A1;A2 ||
    B
   end par;
   C
by D
end disrupt
```

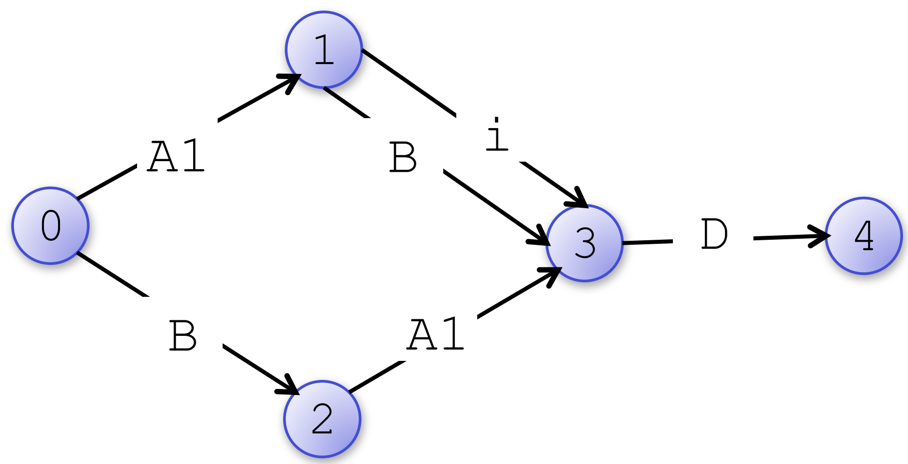
throw



Use disrupt to simulate throw

- We use stop and synchronisations to remove unwanted cases

```
hide F,G:any in
  par F,G in
    disrupt
      par
        A1;F;stop;A2
      ||
        B
      end par;
      C
    by G;D
    end disrupt
  ||
    F;G
  end par
end hide
```



Current state & Conclusion

- Current state
  - Translation algorithm entirely defined and written down
  - Compiler is being implemented
- Conclusion
  - To date, the most complete translation, but
  - Not yet tested on a real application

Conclusion (1/2)

- Two contributions to connecting MDE languages to formal verification toolboxes:
  - Generic approach for verifying GALS systems using process algebras
  - An efficient method for verifying BPEL processes (a compiler is being implemented)
- We tested the limits of MDE-based transformation tools, which are not suited to complex compilations

Conclusion (1/2)

- Generic approach for verifying GALS systems using process algebras:
  - any process algebra with parallel composition, types and functions is suitable
  - multiple synchronous languages can be mixed
  - illustrated on a complex case-study
    - two different wrappers used
    - two different medium processes used
    - had to resort to advanced compositional generation strategies
    - 19 errors found

Conclusion (2/2)

- Almost complete translation from BPEL to LOTOS NT:
  - No other translation covers as many constructs from BPEL
  - Translation is formally defined
  - Heavy focus on data which are ignored by other approaches
  - Enables formal verification of Web Services with CADP
- Interesting conclusion regarding MDE
  - From SAM to CADP, transformation chain is fully MDE
  - From BPEL to CADP, MDE tools reached their limit, they do not scale with the input language complexity

In the future

- Apply our GALS method to other synchronous languages than SAM (current collaboration with Airbus)
- Improve some aspects of the BPEL to LOTOS NT translation (compensation handlers for example)
- Finish the automated translator from BPEL to LOTOS
- Find complex BPEL case studies to verify