Towards Certified Compilation for IEC 61499

Per Lindgren and **Marcus Lindner**
Luleå University of Technology

David Pereira and Luís Miguel Pinho
CISTER / INESC TEC, ISEP

2016-09-06

LULEÅ UNIVERSITY OF TECHNOLOGY

Goal: verified software

Rationale

Correct programming is hard
Also ***tools*** *may contain bugs*

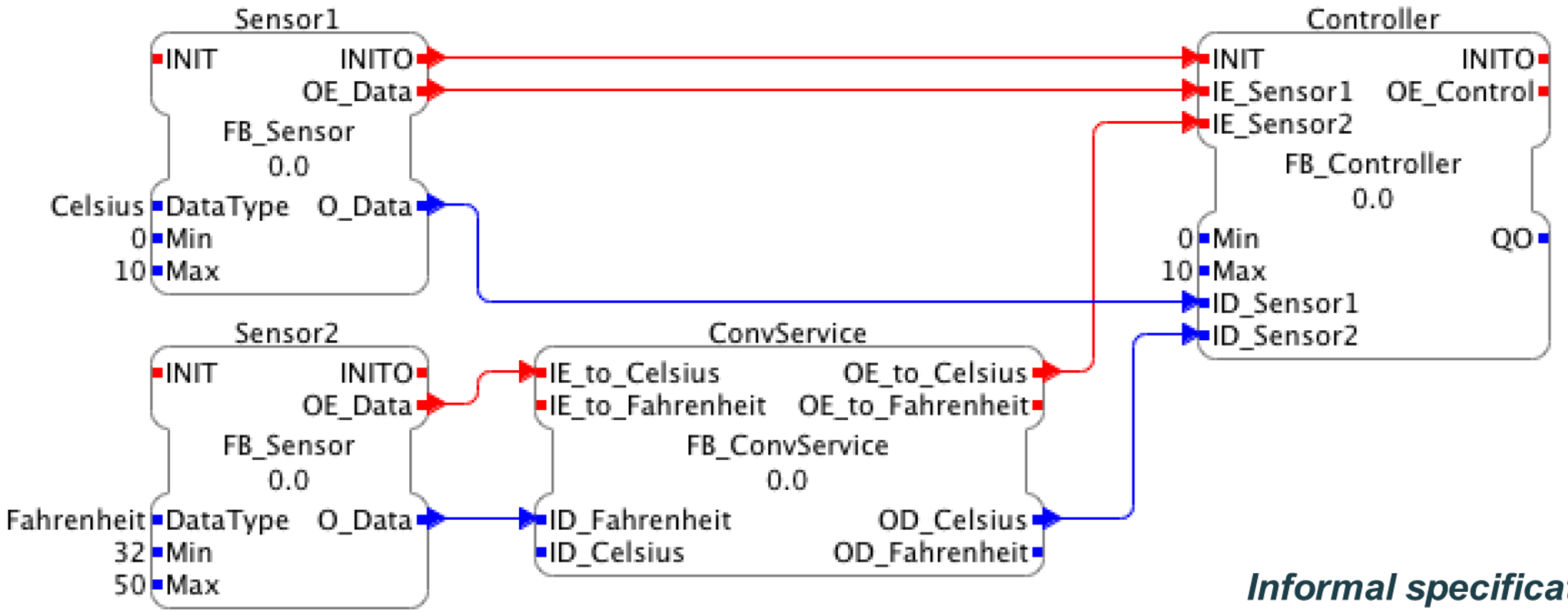
Our research

Provide means for programmers to facilitate *correct software* development in the context of industrial automation

Project objective

"Improving the *robust operation of variable-frequency drives* in energy production plants in terms of software"

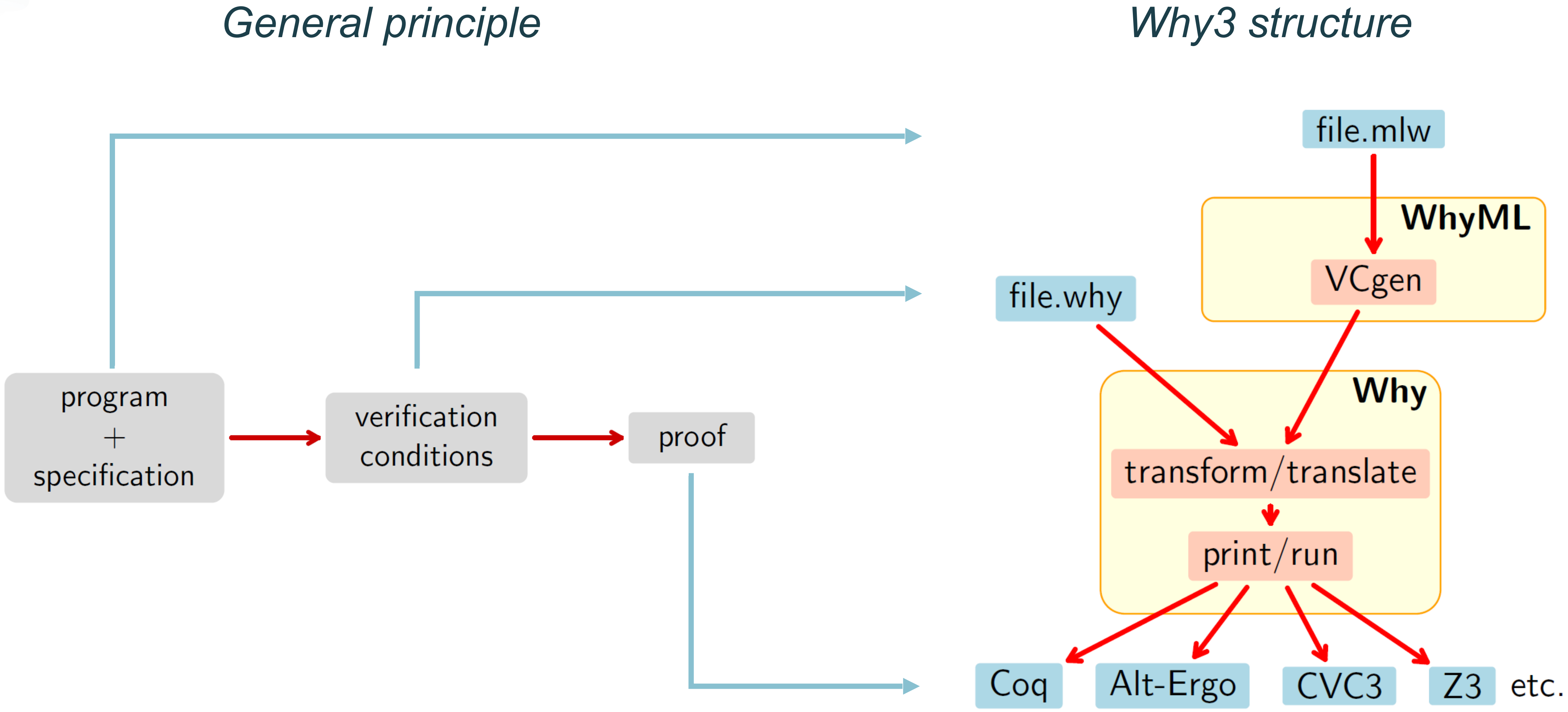
INDIN 2016, Verification of IEC 61499 applications (first ideas was presented at ETFA 2015)



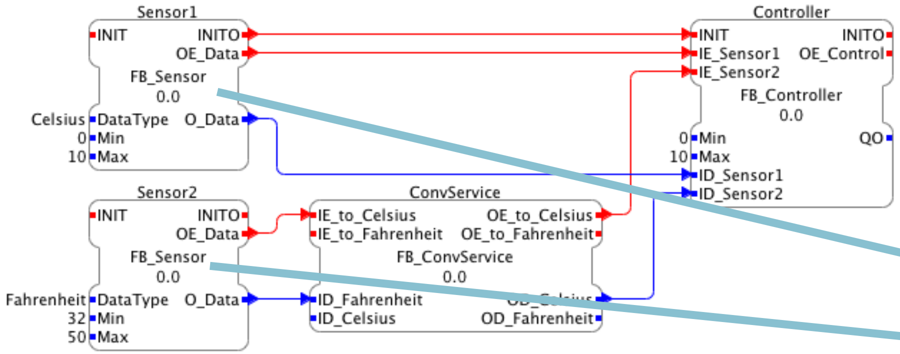
Informal specification:

- 2 sensors
  - S_1: $0°C \leq c < 10°C$
  - S_2: $32°F \leq f < 50°F$
- 1 bang-bang controller (output: $S_1 < S_2$)
- Celsius-Fahrenheit conversion service

Why3: deductive program verification



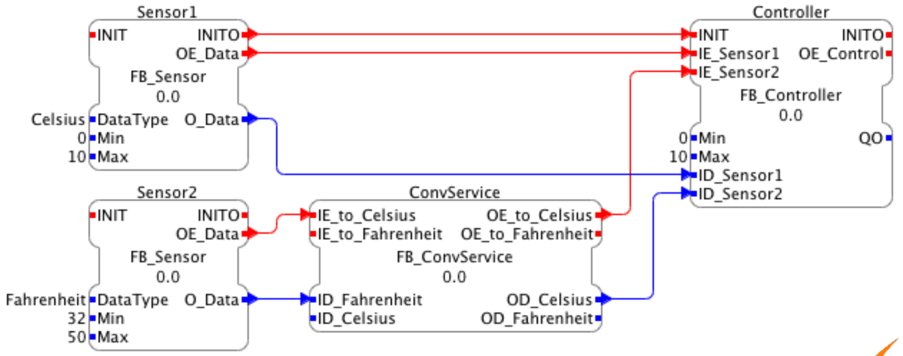
Example: formalized specifications



Formal specification for component interfaces
→ the contract!


```
module SensorGen
  use export Data

  constant min    : int
  constant max    : int
  constant metric : metric_t

  type val_t

  predicate in_range (v : val_t) =
    min <= fst v < max /\ metric = snd v

  val read () : val_t
    ensures  { in_range result }

  function range_of () : int = max - min

end
```

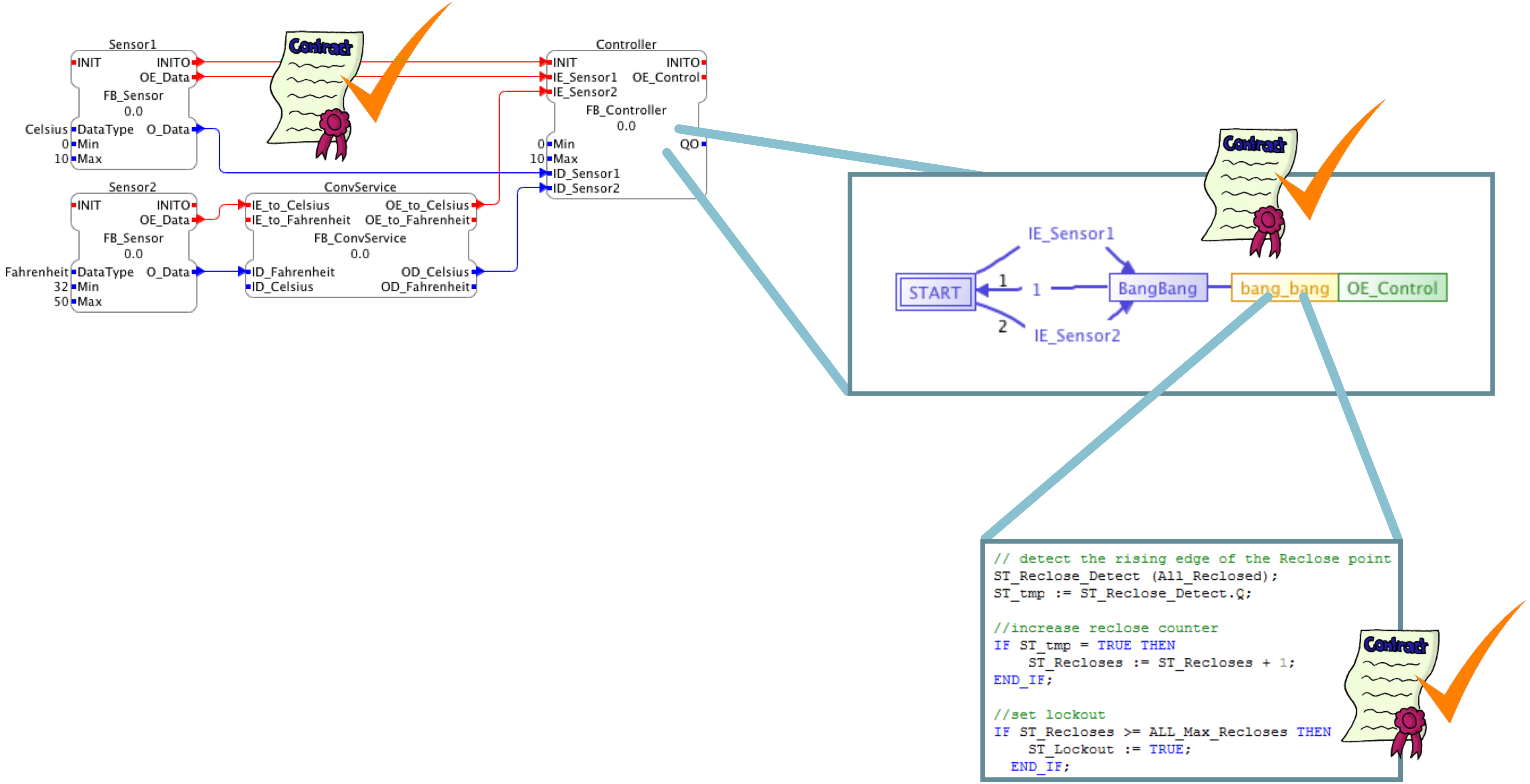
Example: system/FB composition



Verification of compositional soundness (w.r.t. the defined contracts)

```
module System
  clone import Sensor1 as S1
  clone import Sensor2 as S2

  (* should work in Celsius range 0 <= c < 10 *)
  predicate range (c:int) = 0 <= c < 10
  clone import ControlGen with predicate in_range = range

  let orchestration () =
    (* take readings from the sensors *)
    let s1_v = S1.read () in
    let s2_v = S2.read () in

    (* present readings to the controller *)
    let bang = control s1_v s2_v in

    (* make sure the controller meets our expectations *)
    assert {
      match bang with
        | True  -> to_Celsius s1_v <  to_Celsius s2_v
        | False -> to_Celsius s1_v >= to_Celsius s2_v
      end
    }
end
```

What’s more


Sensor1
INIT
INITO
OE_Data
FB_Sensor
0.0
Celsius
DataType
O_Data
0
Min
10
Max
Contract
Sensor2
INIT
INITO
OE_Data
FB_Sensor
0.0
Fahrenheit
DataType
O_Data
32
Min
50
Max
ConvService
IE_to_Celsius
OE_to_Celsius
IE_to_Fahrenheit
OE_to_Fahrenheit
FB_ConvService
0.0
ID_Fahrenheit
OD_Celsius
ID_Celsius
OD_Fahrenheit
Controller
INIT
INITO
IE_Sensor1
OE_Control
IE_Sensor2
FB_Controller
0.0
0
Min
QO
10
Max
ID_Sensor1
ID_Sensor2
Contract
IE_Sensor1
START
1
1
BangBang
bang_bang
OE_Control
2
IE_Sensor2
// detect the rising edge of the Reclose point
ST_Reclose_Detect (All_Reclosed);
ST_tmp := ST_Reclose_Detect.Q;
//increase reclose counter
IF ST_tmp = TRUE THEN
ST_Recloses := ST_Recloses + 1;
END_IF;
//set lockout
IF ST_Recloses >= ALL_Max_Recloses THEN
ST_Lockout := TRUE;
END_IF;
Contract

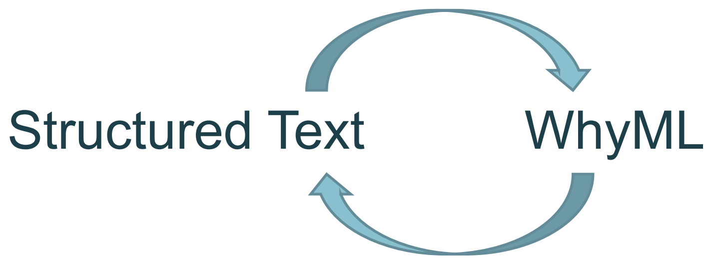
Future work towards verified implementations

LULEÅ UNIVERSITY OF TECHNOLOGY

- Currently: Why3 encoding of IEC 61499 programs made by hand (should be largely automatized)

- suitable representations for the IEC 61499/IEC 61131-3 data types defined in WhyML and deployed for automatic translation

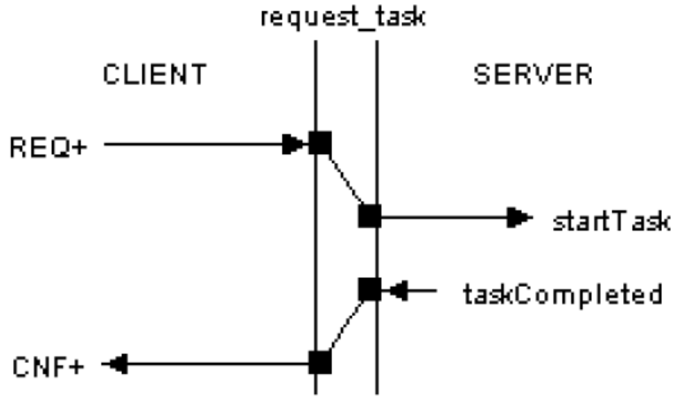
- Language translation:



- Tool integration: Why3 platform directly accessible through Function Block IDE *4DIAC*

ETFA 2016, WIP, Further Verification Possibilities

- Deductive verification approach in contrast to model checking approach
- Limited possibilities of IEC 61499 to express behavior at component level: service sequences (no notions of timing, data, and state)



- design by contract approach is proof enabling and allows for the extraction of verified implementations

ETFA 2016, WIP
Towards Certified Compilation of RTFM-core Applications

- RTFM-core : Concurrent tasks with **nested** critical sections
- Critical sections protected by named resources
- Deadlock free execution onto single core MCUs under the Stack Resource Policy
- ***Mapping from IEC 61499 to RTFM-core possible*** (C/C++ for algorithms)

- **CoreCert** Certified Compiler that
  - flattens nested structure into flat sequence of instructions
  - generates ceilings for named resources
  - proven correct to semantic model(s) in Why3
  - extracted to OCaml to generate executable


Towards Certified Compilation of RTFM-core Applications
Per Lindgren[1] Marcus Lindner[1] David Pereira[2] Luís Miguel Pinho[2]
[1]Luleå University of Technology Email:{per.lindgren, marcus.lindner}@ltu.se
[2]CISTER / INESC TEC, ISEP Email: {dmrpe, lmp}@isep.ipp.pt

Abstract
RTFM-core Language and Model of Computation
RTFM-core Grammar (Simplified)
RTFM-core Example
RTFM-kernel
RTFM-core Compiler
Why3 - a Platform for Deductive Program Verification
Why3 Data Types, for the RTFM-core compiler
Certified Compilation
Correct compilation of RTFMspec
Correct compilation of Resource Ceilings
Conclusions
References

Current Work
Extending RTFM-core (CoreCert) with “imp” language

- Subset of the IEC 61499 Structured Text

- variables, expressions
- statements
  - assignments
  - conditionals
  - loops

- proven Byte code generation
- proven VM (stack machine) for Byte code

- generates MIPS assembler
  - optimal register assignment
  - proofs on the way

```
// Example
// 1+2+3..6 = 21

WHILE a <= 5 DO
  a := a + 1;
  sum := sum + a
DONE
```

Future Work

- Extending front-end to larger subset of Structured Text
- Extending backend to ARM assembly
- Adopting the RTFM-4-FUN to generate RTFM-core-imp
  - ***May be proven as well, (formalising semantics of ST)***
- Integration to 4DIAC
  - Certified toolchain, provably correct implementation

Huge effort. Is it worth it?

- CompCert (C compiler) > 10 years in the making
- Commercial licence available
- We can already use CompCert with RTFM-4FUN
- Formalised semantics of C is difficult, “core-imp” simpler

Thank you for your attention!

Questions or comments