Formal Verification of the PolyORB-HI Middleware

Jérôme Hugues, ISAE/DMIA

with support from Christophe Garion, ISAE/DMIA

and Gregory Essertel, ISAE/SUPAERO student

Objective of TASTE COO2

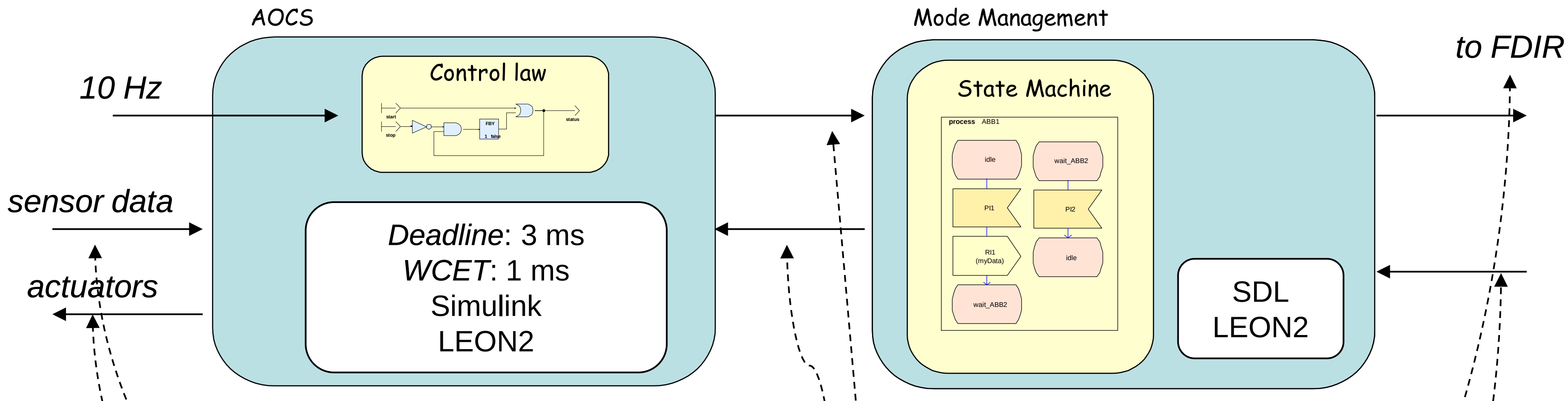
- As part of TASTE COO2 project, ISAE was tasked to explore usage of theorem proving tools, C/ACSL and SPARK2014
  - How can they be used to assess the quality of the code generated by the TASTE toolchains
  - Mostly exploratory, small workforce of 1 man.month funded by ESA, dedicated to review existing tools, technologies and see how far we can go

Short answer for the TL;DR folks: it works not so bad except for pointers management and tasking
Longer answer in the slides that follow ;-)

Outline

- Context & objective
- A little bit of theory
- Proof of the C runtime
- Proof of the Ada runtime
- Some lessons learnt

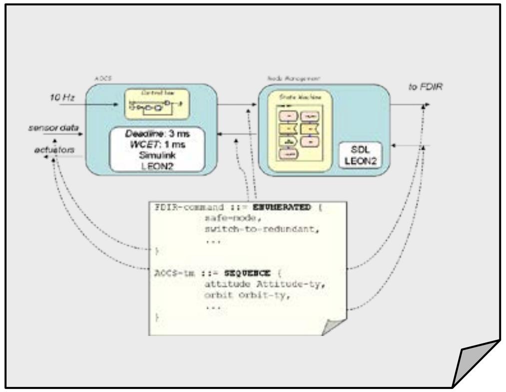
About TASTE and the TASTE process



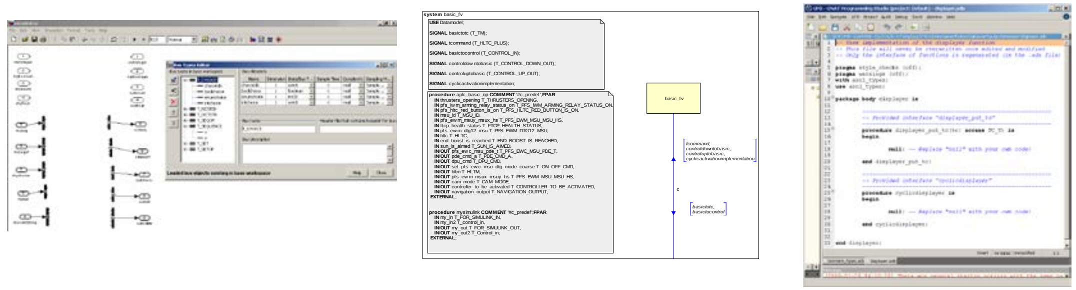

```
FDIR-command ::= ENUMERATED {
    safe-mode,
    switch-to-redundant,
    ...
}

AOCS-tm ::= SEQUENCE {
    attitude Attitude-ty,
    orbit Orbit-ty,
    ...
}
```

AADL and ASN.1
are combined to provide a formal, precise, and complete description of the system architecture and data.

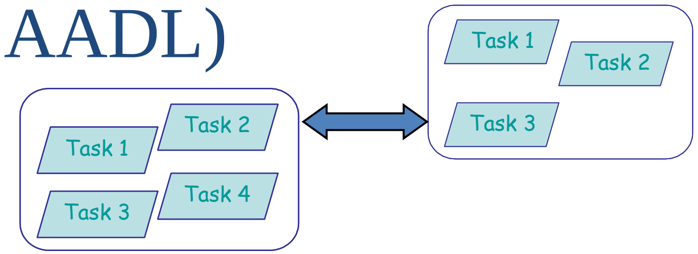
ASSERT “model compilation”



① Generate “application skeletons” in Simulink, SDL, C, and Ada



② Generate a software real-time architecture (in AADL)

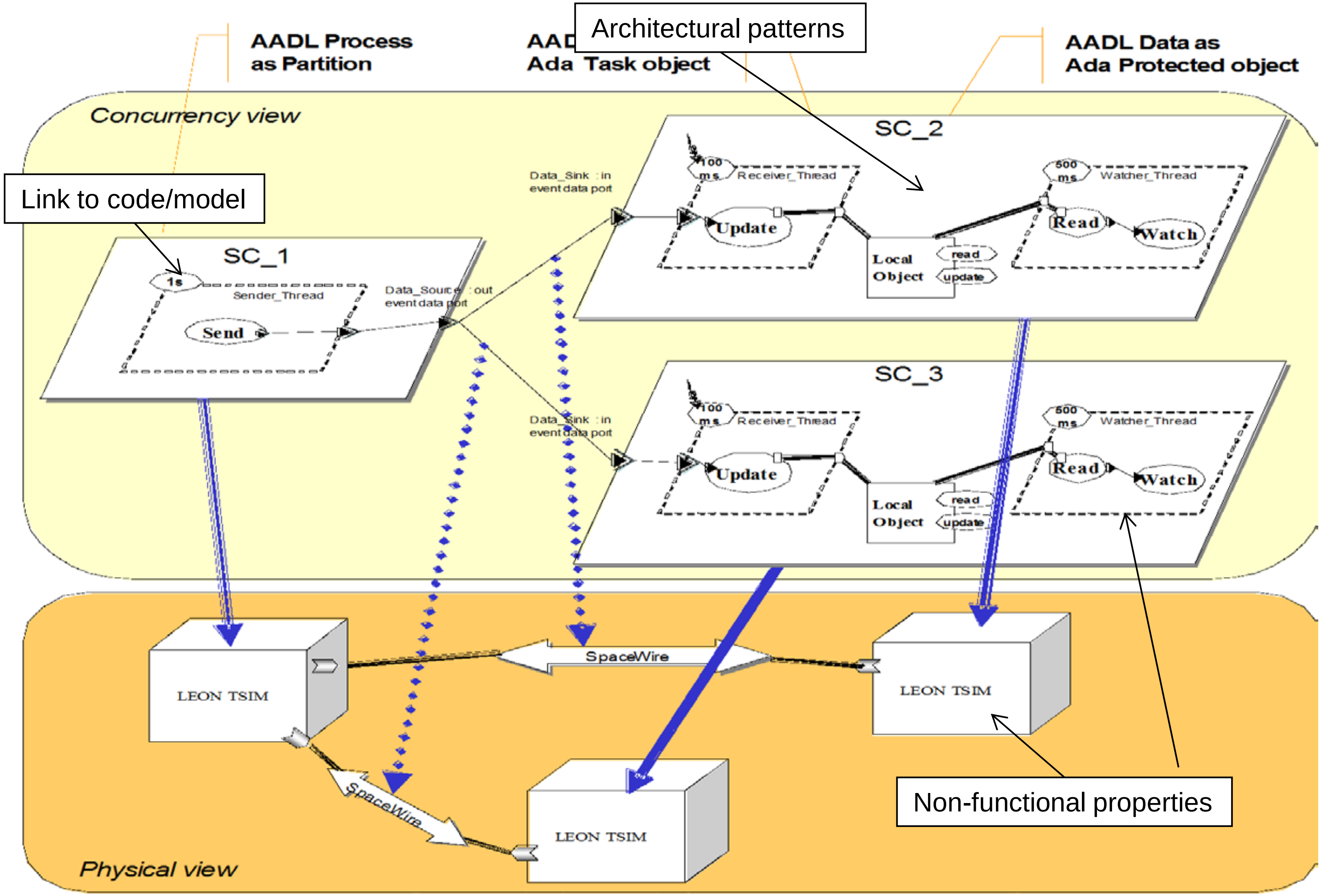


③ Generate glue code to put everything together on a real-time operating system

Behind the scenes

- TASTE relies on standardized languages :
  - ASN.1 and AADL to capture the software architecture and data
  - SDL, Simulink, SCADE, C, Ada, VHDL, … to capture the software behaviour
  - MSC and Python to test
- Combine graphical AND textual notations
  - If anything goes wrong, human can fix textual syntax
  - Diagrams for easier understanding
  - But some information is textual by nature
- Avoid languages with weak semantics or syntax

What you should to know about AADL @ ISAE



AADL covers many parts of the V cycle: model checking, scheduling, safety and reliability (ARP4754) and code generation

ISAE contributions to SAE AADL since 2009

Lead on the Ocarina toolset, used by ESA

Code generation : Ada, C (POSIX, ARINC653), RTOS

TRL 7 with ESA (ECSS E-40)

Scheduling: Cheddar, MAST

TRL 4 with ESA

Model checking: Petri Nets

TRL 2 (PhD contribution)

Architectural Constraints/Requirements checks

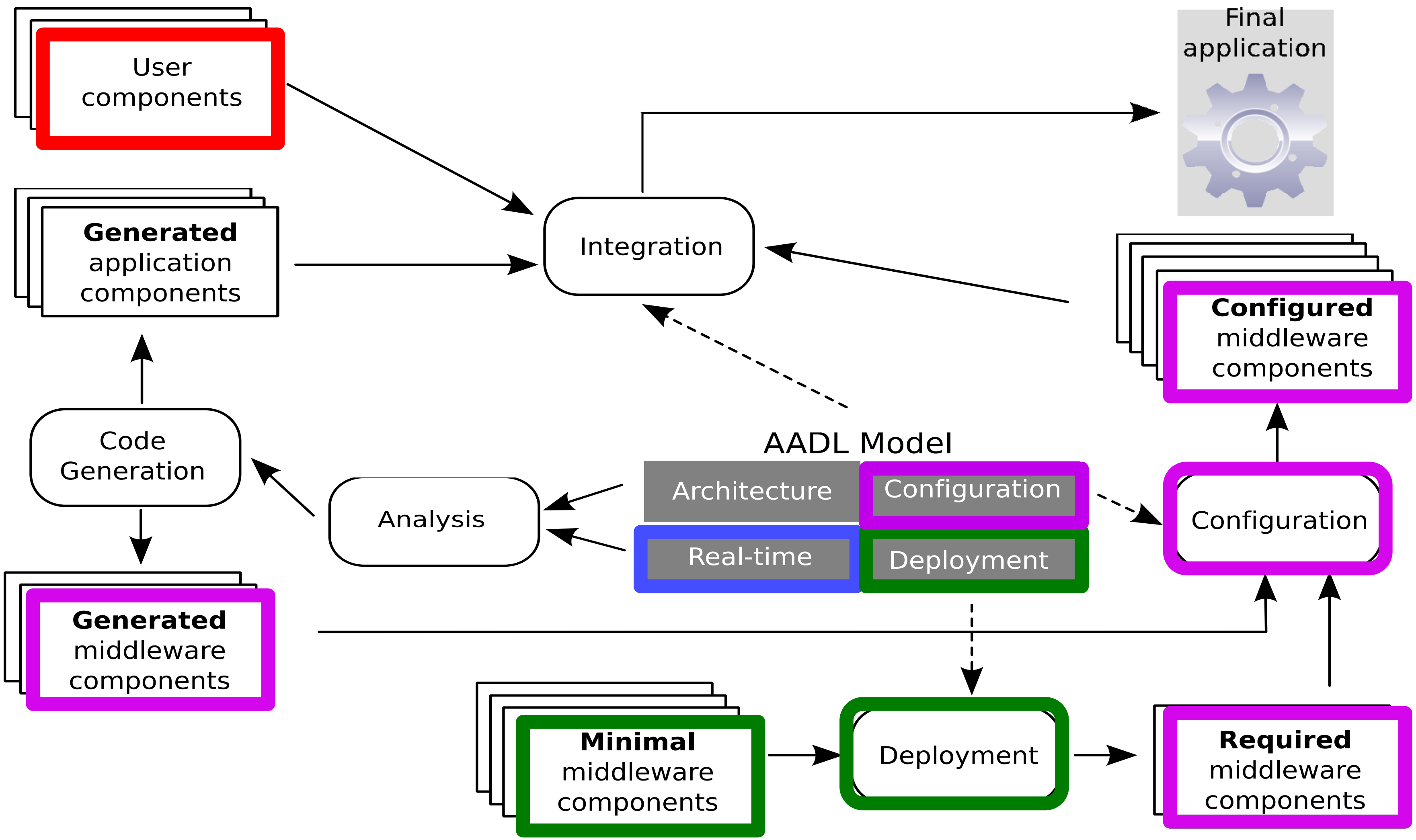
TRL 6 with SEI, being standardized

Link with SysML models

TRL 1, under evaluation

Code generation from AADL models

> Exploits AADL models information to generate application-tailored middleware



Ocarina: an AADL code generator
http://www.openaadl.org

- **Ocarina is a stand-alone tool for processing AADL models**
  - Command-line tool, a-la gcc
  - Can be integrated with third-party tools
    - OSATE (SEI), TASTE (ESA), Cheddar (UBO), MyCCM-HI (Thales)
    - Also emacs and vim modes
- **Code generation facilities target PolyORB-HI runtimes**
- **Two flavors**
  - Ada HI integrity profiles, with Ada native and bare board runtimes
  - C POSIX or RTEMS, for RTOS & Embedded
- **Generated code quality tested in various contexts**
  - For WCET exploration, support for device drivers, …
  - For various RTOS
- **Written to meet most High-Integrity requirements**
  - Follow Ravenscar model of computations, static configuration of all elements (memory, buffers, tasks, drivers, etc.).

TASTE COO2 roadmap

- Initial objective: demonstrate TASTE runtimes (PolyORB-HI/C and Ada) are free of runtime errors (RTE)
- The leading tool for asserting code is free of RTEs are
  - For C: ACSL framework, from CEA; combined with Why3 from Inria along with various analysis plug-ins
  - For Ada: forthcoming SPARK2014 from AdaCore and Altran Praxis

- Stakeholders agreed on the following roadmap
  - Adapt existing toolchains to process runtime entities
  - Perform annotations on source code to asses lack of RTEs

Outline

> Context & objective
> A little bit of theory
> Proof of the C runtime
> Proof of the Ada runtime
> Some lessons learnt

Some theoretical background

- We are interested in imperative programs: when executing a program P, we go from a state s_i to a state s_f
- Specifying P is characterizing s_i and s_f **with a Hoare triple {φ} P {ψ} where:**
  - » φ is a logical formula called the precondition of P
  - » ψ is a logical formula called the postcondition of P
- Intuitive meaning: starting from a state verifying the assertions **defined by φ, executing P should lead to a state verifying the assertions defined in ψ.**
- Thus, verifying the correctness of P given its specification as a **Hoare triple {φ} P {ψ} is to prove the Hoare triple {φ} P {ψ} in** the Floyd-Hoare formal system.

About the Floyd-Hoare formal system

> Four constructs: assignment, sequence, conditional, iteration
  » Enough to represent all sequential programs

$$\frac{}{\{\varphi[x/E]\}\ \ \mathsf{x := E}\ \ \{\varphi\}}\ (:=)$$

$$\frac{\{\varphi\}\ \ \mathsf{P}\ \ \{\gamma\} \qquad \{\gamma\}\ \ \mathsf{Q}\ \ \{\psi\}}{\{\varphi\}\ \ \mathsf{P;Q}\ \ \{\psi\}}\ (\mathsf{Seq})$$

$$\frac{\varphi \rightarrow \varphi' \qquad \{\varphi'\}\ \ \mathsf{P}\ \ \{\psi'\} \qquad \psi' \rightarrow \psi}{\{\varphi\}\ \ \mathsf{P}\ \ \{\psi\}}\ (\mathsf{Cons})$$

$$\frac{\{\varphi \wedge C\}\ \ \mathsf{P}\ \ \{\psi\} \qquad \{\varphi \wedge \neg C\}\ \ \mathsf{Q}\ \ \{\psi\}}{\{\varphi\}\ \ \mathsf{if\ \ C\ then\ P\ \ else\ \ Q\ \ fi}\ \ \{\psi\}}\ (\mathsf{Cond})$$

$$\frac{\{\varphi \wedge C \wedge v = V\}\ \ \mathsf{P}\ \ \{\varphi \wedge v \prec V\} \qquad \prec \mathsf{\ is\ wf}}{\{\varphi\}\ \ \mathsf{while\ \ C\ \ do\ P\ od}\ \ \{\varphi \wedge \neg C\}}\ (\mathsf{It})$$

The weakest-precondition calculus

> The weakest-precondition calculus is a particular semantics for imperative programs that can be viewed as a complete strategy to build deductions in FH logic

> **Main idea: given a specification {φ} P {ψ}, start from ψ, find the minimal (weakest) precondition allowing to deduce ψ and verify that φ implies this minimal precondition**

> Implies exploring a proof-tree, applying FH rules

» Combined with higher-order theories (for naturals, memory, …)

» And some specific strategies to speed up the process

$$\dfrac{\varphi \rightarrow I(0,X) \qquad \dfrac{\{I(0,X)\}\ \text{P1}\ \{I(Q,R)\} \qquad \{I(Q,R)\}\ \text{P2}\ \{I(Q,R) \wedge \neg(Y \leq R)\}}{\{I(0,X)\}\ \text{P}\ \{I(Q,R) \wedge \neg(Y \leq R)\}}}{\{\varphi\}\ \text{P}\ \{I(Q,R) \wedge \neg(Y \leq R)\}}$$

How does it relate to my source code ?

> The process is thus the following:
  - » a human expert writes specification in terms of preconditions, postconditions, loop invariants, loop variants etc. for a program
  - » those specifications are translated into verification conditions (VC) using weakest-precondition calculus. The VC are purely mathematical/logical statements
  - » the VC are then passed to a theorem prover, automated or interactive, to be discharged

```
/*@ ensures \result >= x && \result >= y;
    ensures \result == x || \result == y;
*/
int max (int x, int y) { return (x > y) ? x : y; }
```

Specification

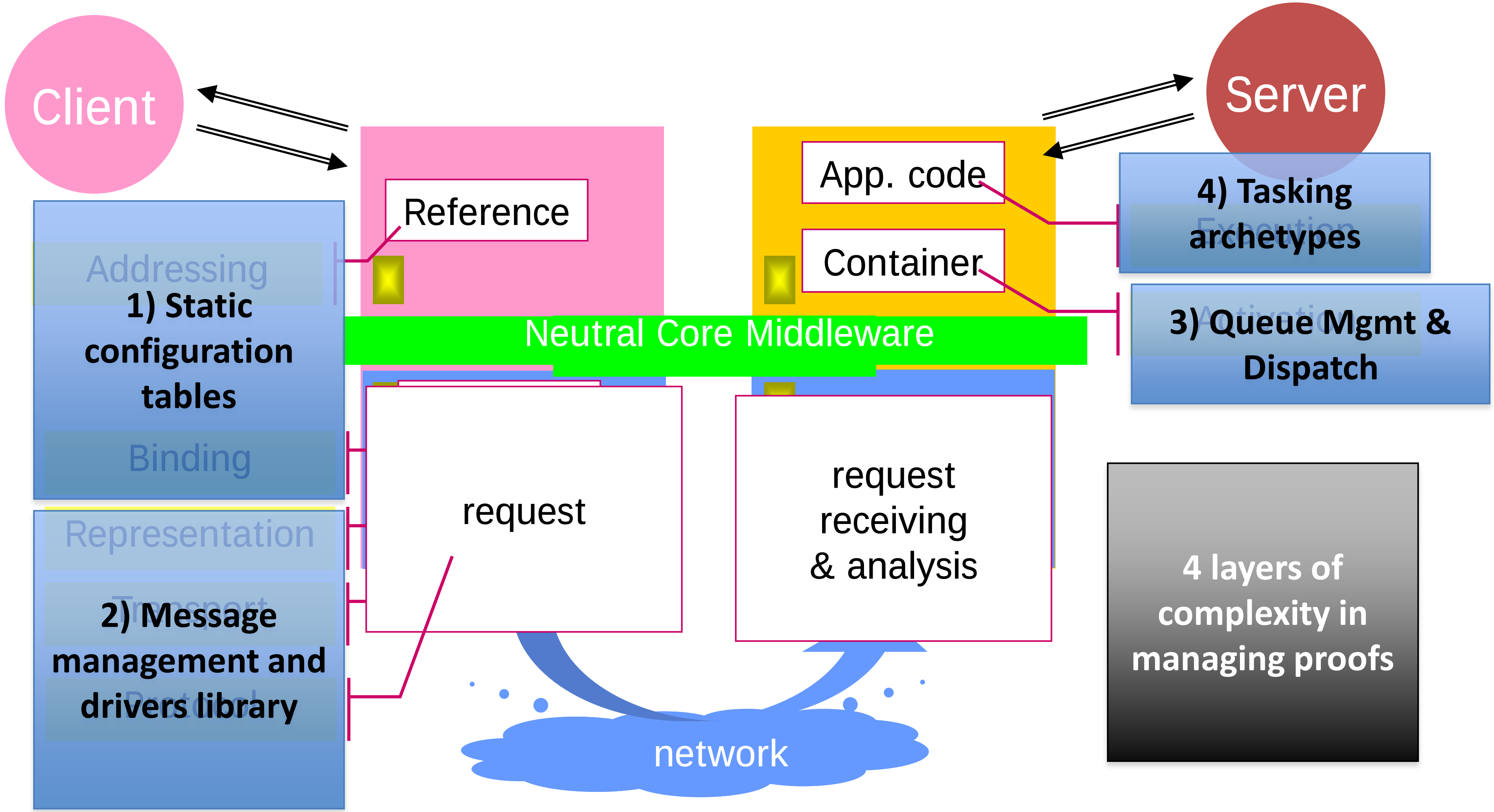
Implementation

Refining the roadmap

- The C and Ada PolyORB-HI runtimes share a common heritage
- Same global architecture, built on top of the “schizophrenic” middleware pattern
- Similar code design and patterns
  - Ravenscar archetypes for task constructs
  - Same queueing discipline for messages
  - Same code patterns generated from AADL description
- Yet, different implementation choices
  - Ada: rely on limited but rich High-Integrity subsets
  - C: must accommodate for an abstraction layers on top of OS for tasking, concurrency and time management

About the schizophrenic middleware architecture

> Inherited from the PolyORB middleware
> A generic definition of middleware architecture



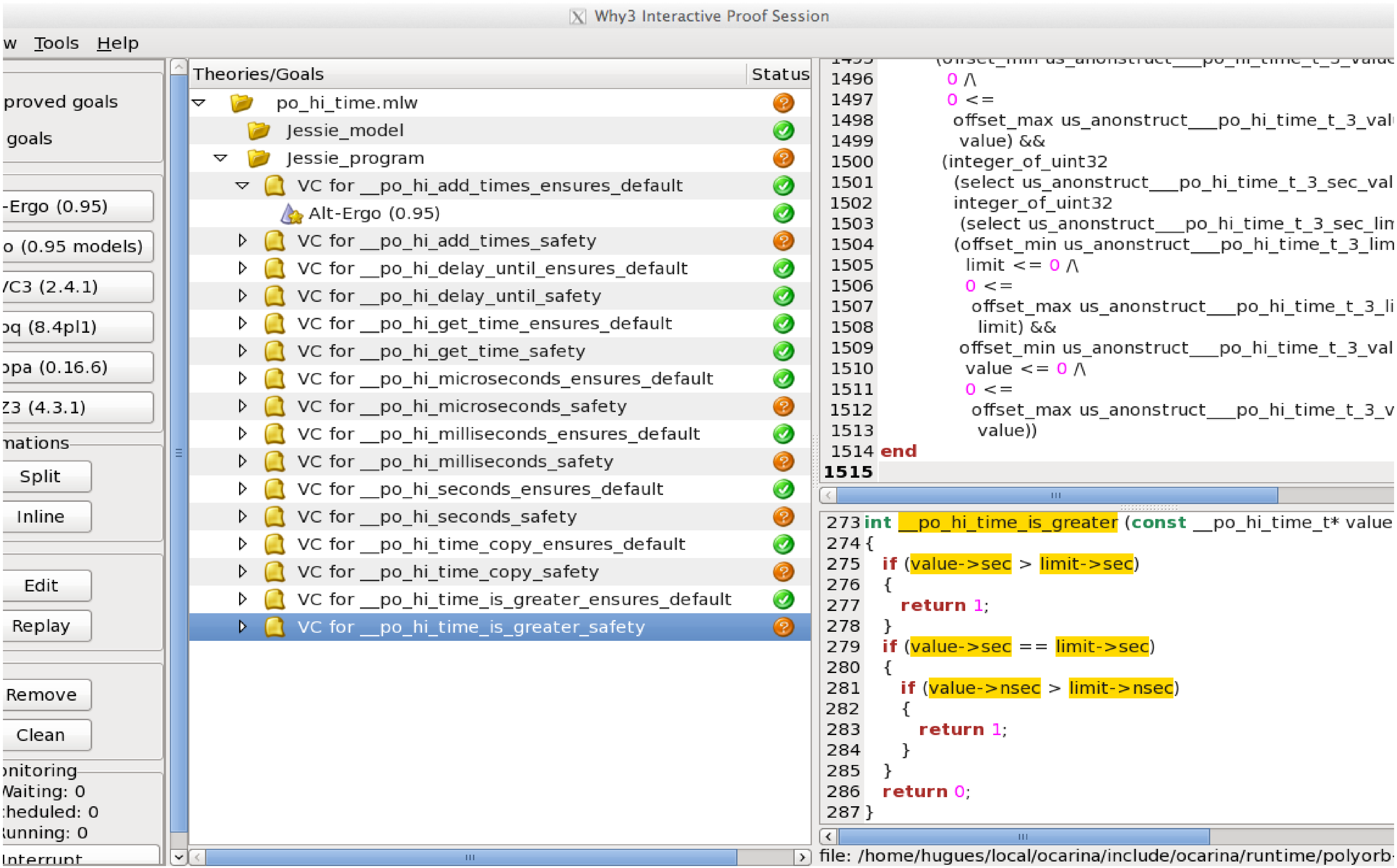
Outline

- Context & objective
- A little bit of theory
- Proof of the C runtime
- Proof of the Ada runtime
- Some lessons learnt

The Frama-C platform

> We have used the Frama-C platform to prove the runtime.
  - » the ACSL specification language allows to express assertions, preconditions, postconditions, loop variants and variants etc. as special comments in the C source code
  - » the RTE plugin has been used to generate additional assertions about possible runtime errors (signed integers overflow, invalid memory access, division by zero etc.)
  - » the WP plugin has been used as VC generator for the weakest-precondition calculus
  - » the Alt-Ergo SMT solver has been used to automatically prove the generated VCs

Frama-C/Why in action



Proving PolyORB-HI/C

> The following C compilation units must be considered
  - po_hi_types.h: definition of simple types and a function to copy arrays of bytes
  - po_hi_time.h: time management, with definition of a struct embedding time decomposed into seconds andnanoseconds. Several functions to add, initialize etc. such structures
  - po_hi_marshallers.h: conversion functions for marshalling/unmarshalling data
  - po_hi_messages.h: messages management functions (write message, append message, move part of a message etc.)
  - po_hi_main.h: defines functions for a synchronized start of the system
  - po_hi_protected.h: mutex management functions
  - po_hi_task.h: link to concurrency library
  - po_hi_gqueue.h: queue management functions
  - po_hi_transport.h: communications between tasks functions

About the complexity of annotations

> C memory model is a pain

```
/*@ requires \valid(((char *) dst)+(0..size-1));
  @ requires \valid(((char *) src)+(0..size-1));
  @ requires \separated(((char *) dst)+(0..size-1), ((char *) src)+(0..size-1));
  @ assigns ((char *) dst)[0..size-1] \from ((char *) src)[0..size-1];
  @ ensures \forall int i; 0 <= i < size ==> *(((char *) dst)+i) == *(((char *) src)+i);
  @*/
void __po_hi_copy_array (void* dst, void* src, __po_hi_uint32_t size);
```

> The specification reads as follow:

- » Pointer parameters are valid, at least for the length size specified as a parameter.
- » The memory regions of the two pointers do not overlap (`\separated` clause).
- » Only the dst pointer will be assigned, using the src pointer.
- » as the \from clause is an experimental feature, we have to specify the complete postcondition, i.e. that the size first bytes of src has been copied into dst.

About memory models

- All memory is statically allocated in the heap
  - As we deal with values living inside the heap, we have to use a memory model allowing to map each C value in the heap to logical expressions in the ACSL specification.
  - Of course, the more precise the memory model is, the more difficult the generated VC are to discharge.
- Limitations ☹
  - We have used the Typed memory model that allows reasoning with pointers with an efficient mixed memory model.
  - Unfortunately, the Typed memory model does not allow all possible casts between pointer types (for instance int $*$ to void $*$ is not allowed).

Abstracting OS primitives

- Use of external libraries require precising expected outputs
  - » Normal and error case, as this is propagated back to clients

```
/*@ behavior __tp_not_valid:
  @   assumes !\valid(__tp);
  @   assigns \nothing;
  @   ensures \result == EFAULT;
  @ behavior clock_not_valid:
  @   assumes !clock_valid(__clock_id);
  @   assigns \nothing;
  @   ensures \result == EINVAL;
  @ behavior normal:
  @   assumes \valid(__tp);
  @   assigns __tp->tv_sec;
  @   assigns __tp->tv_nsec;
  @   ensures \result == 0;
  @   ensures \valid(__tp);
  @   ensures __tp == \old(__tp);
  @   ensures __tp->tv_sec >= 0 && __tp->tv_sec <= UINT32_MAX;
  @   ensures __tp->tv_nsec < 1000000000 && __tp->tv_nsec >= 0;
  @*/
extern int clock_gettime (clockid_t __clock_id, struct timespec *__tp) __THROW;
```

Basic computations

- Some complex corner cases to evaluate even basic arithmetics
  - » Ex: byte swapping, shifting and integers do not mix very well

```
/□@
  @ ensures \result == ((value & 0x000000ff) << 24) +
  @                    ((value & 0x0000ff00) << 8)  +
  @                    ((value & 0x00ff0000) >> 8)  +
  @                    ((value & 0xff000000) >> 24);
  @□/
```

  - » But this can be proved

```
unsigned long __po_hi_swap_byte (unsigned long value)
{
    unsigned long v = 0;

    v |= (value % 256) □ 16777216;
    v |= ((value / 256) % 256) □ 65536;
    v |= ((value / 65536) % 256) □ 256;
    v |= (value / 16777216);

    return v;
}
```

    - Runtime penalty limited thanks to compiler optimizations

Global results

> Proving basic units, except queue management and tasking

VC	To be proved	Proved	Time (ms)	Qed	Alt-Ergo	Group
Qed	65	65	56			Tool
Alt-Ergo	156	152	28376			
Pre	16	16	668	12	4	Category
Post	41	41	12156	18	23	
RTE	92	90	9888	13	79	
Assigns	58	56	1732	16	42	
Loop	10	10	3940	6	4	
Other	4	4	48	0	4	
Total	221	217	28432			

> Still a lot to do at Frama-C/ACSL tool-support level

» No support for complex pointers, required for queues

» No support for concurrency constructs, impossible to demonstrate absence of interferences between task, or respect of Ravenscar!

> Recall this was a 3 man.month combined effort

Outline

- Context & objective
- A little bit of theory
- Proof of the C runtime
- Proof of the Ada runtime
- Some lessons learnt

About SPARK2014

- SPARK2014 leverages Ada2012 aspects to enable definition of contracts that can be either evaluated at run-time, or proved formally

```
procedure Push (R : in out Ring_Buffer; X : Integer)
    with Pre  => (not Is_Full (R)),
         Post => (R.Length = R.Length'Old + 1);
```

  - Annotation through valid Ada code + special attributes
  - Part of the compilation process

- GNATProve GPL2013 then GPL2014 used in this study
  - First a prototype, now supported by AdaCore and Altran Praxis
  - Requires adoption of Ada2012 to support all annotations

Proving PolyORB-HI/Ada

- The following packages must be considered
  - PolyORB-HI.Output: logging facilities
  - PolyORB-HI.Utils: helper functions
  - PolyORB-HI.Messages: message management
  - PolyORB-HI.Marshallers_G: marshalling functions
  - PolyORB-HI.Thread_Interrogators: message queues
  - PolyORB-HI.*_Task: task archetypes
- GNATProve allows one to control which package to analyse
  - E.g. do not consider drivers
- GNATProve generates VCs for code that lead to RTEs
  - Exclude safe code after compiler analysis

Adapting the Ada runtime

- Compared to C, annotations define pre/post conditions
  - Hypothesis on behavior shared with client
  - Mostly invariants on validity of data being exchanged
    - E.g. message well-formed, non-empty arrays, etc.
- No need to tell a lot about memory model
  - No need for pointers in High-Integrity Profile in Ada !
- No need to abstract OS services
  - These are part of Ada semantics
    - Copy of arrays, time management
- No need to precise bounds on types
  - Use subtype mechanism of Ada
    - Simplify many annotation when doing packet construction for instance

Global results

- Proving basic units, except queue management and tasking
  - Exact same subset as the C runtime
- 95 VCs to be discharged
  - 77 proved already
  - 18 unproved due to limitations in toolset
    - Mostly related to slicing and copies, need to adapt proof strategies
    - Easy to solve at tool level support
- Similar limitations to Frama-C/ACSL tool-support level
  - No support for concurrency constructs, impossible to demonstrate absence of interferences between task, or respect of Ravenscar!
  - Recall we have no need for pointers ☺

Outline

- Context & objective
- A little bit of theory
- Proof of the C runtime
- Proof of the Ada runtime
- Some lessons learnt

SPARK2014 vs ACSL

- SPARK2014
  - » Better language semantics
  - » Leverage Ada profiles
    - Restrictions enforced
  - » Annotations compilable
- ACSL
  - » Weak semantics of C
    - Annotations more complex
  - » No enforcement of profiles
  - » Annotations external
    - Risk of discrepancies
    - Compiled with E-ACSL
- Same backend technology: Why3, VCGen and Alt-Ergo
  - » Similar strategies applied
  - » Main difference in strategy to generate VC and management of input language semantics

PolyORB-HI/C & Ada

- Same subset of the runtime proved
  - Message marshalling, constructions and basic helper functions
  - Cover 50% of the code
- Limits in toolset to process complex queues of messages
  - To be addressed shortly, mostly a time issue
- Limits in theoretical framework to address concurrency
  - Must be abstracted away, through model of computation
  - Use of the Ravenscar model at must
- Which one to chose ? Portability of Ada helps !
  - 2 times more VCs to be discharged for the C variant!
  - Needs adaptation of the C runtime for every RTOS variants ☹

Future work

- Complete integration with TASTE toolset
  - Integrate new variants of the runtime to baseline
  - Provides script to automate proof as part of user-visible GUIs
- Extend proof also to user code
  - Needs better modeling artefacts at TASTE process-level
- Study evolution of theorem provers
  - A very active community!