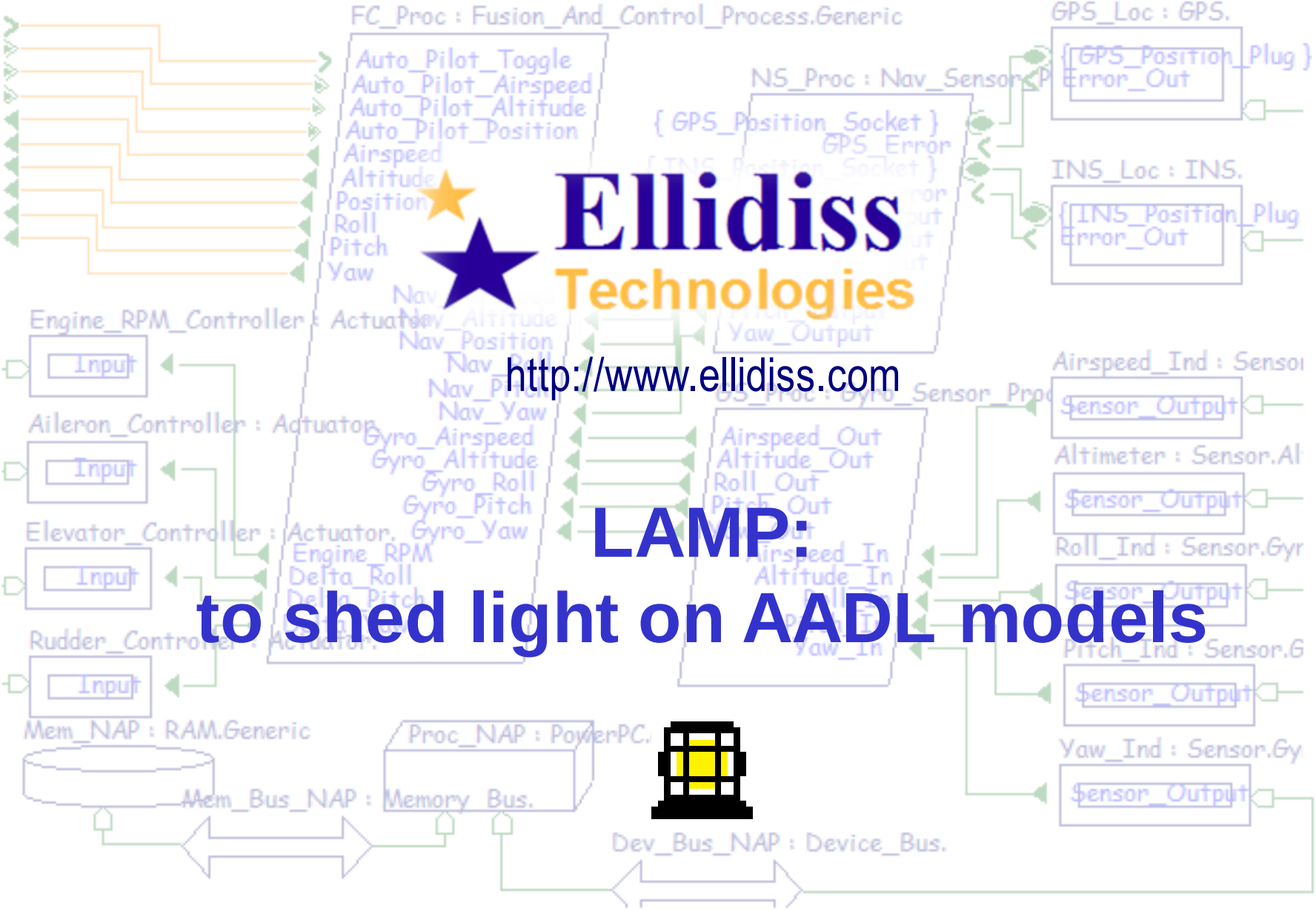


Ellidiss Technologies

http://www.ellidiss.com

LAMP:
to shed light on AADL models

Ellidiss Technologies

Activities and Products

Long-term support of industrial projects:

HOOD Software design tools for Ada and C

Eurofighter Typhoon

Tiger

Airbus A350

- CP-Hood
- Stood for Hood
- Stood for AADL
- AADL Inspector
- VS Code extensions for AADL & SysML2

Products

Innovation in new tools & technologies:

Model Driven Engineering **tools** for SW-intensive critical systems:

- Architectural SW Design: HOOD, AADL, SysML v2
- Early SW Verification: Real-Time, Safety, Security, Assurance Cases
- Logic Model Processing (LMP): Model Exploration, Verification, Transformation

Collaborative R&D projects:

- European Space Agency: TASTE
- European Commission: ERGO and MOSAR Projects (H2020)
- University of Brest: Cheddar

Ellidiss Technologies

Our AADL tools ecosystem

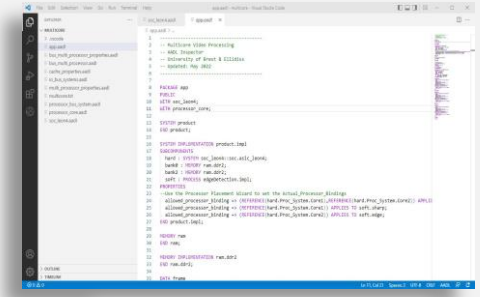
(not osate-based !)

System design

SW requirements

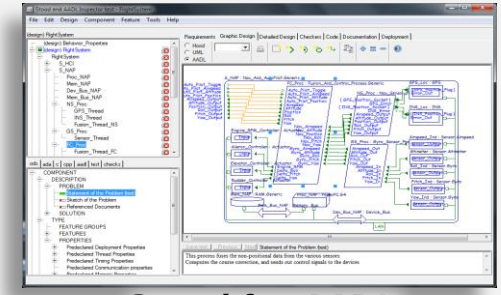
Textual design

SW design



VSCode extension for AADL

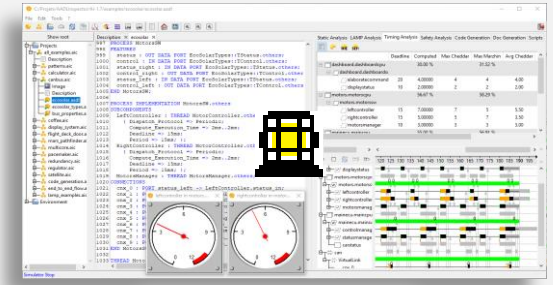
Graphical design



Stood for AADL

AADL files

Early verification



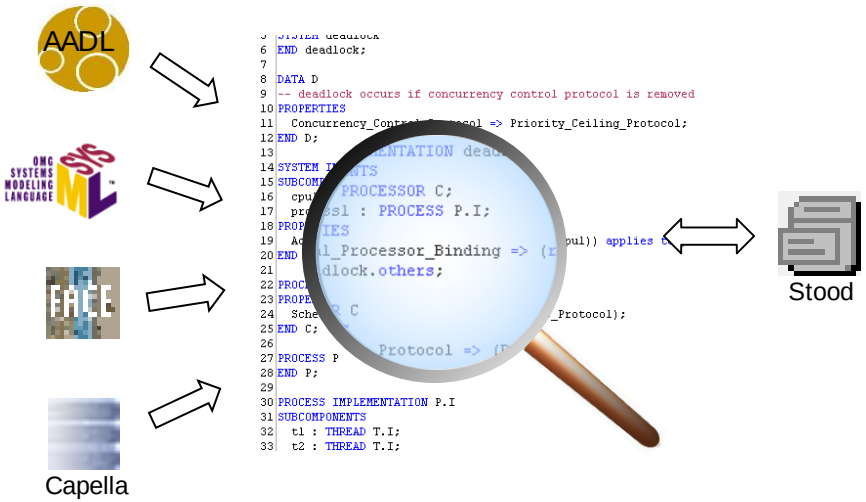
AADL Inspector

SW coding

AADL Inspector 1.9

model processing framework

new release



Ellidiss Technologies

www.ellidiss.com

***Features*:**

AADL support
- AADL Project browser
- Support of AADL 2.3 (SAE AS-5506D) (*New*)
- Support of the AADL Behavior Annex
- Support of the AADL Error Annex (EMV2)
- Support of the AADL ARINC 653 Annex
- Support of the AADL FACE Annex
- Automatic model instantiation

Models import/export
- SysML import (*Upgraded*)
- FACE import (*Upgraded*)
- Capella P.A. import (*New implementation*)
- Generic XML/XMI/CSV import (*Upgraded*)
- HOOD/SIF export (*New*)

Static Analysis
- AADL rules checkers
- Integration of the Ocarina AADL compiler

Real-time analysis and simulation
- Scheduling Analysis with Cheddar (*Upgraded*)
- AADL simulation with Marzhin (*Upgraded*)
- End to End Flow Latency Analysis

Safety & Security analysis
- EMV2 to OpenPSA transformation
- Fault Tree Analysis with Arbre Analyst
- Customizable Security rules checker

LAMP Laboratory and LAMP Libraries (*Upgraded*)
- Prolog rules embedded as AADL annexes
- Architectural reasoning
- Multi-languages cross processing
- User defined assurance cases

AADL reverse engineering (*New*)
- AADL instance model graphical generator
- Round-trip engineering with Stood for AADL

Acknowledgements:
- *Cheddar is developed by the University of Brest*
- *Ocarina is developed by TelecomParisTech, ISAE and ESA*
- *Marzhin is developed by Ellidiss Technologies and Virtualys*
- *Arbre Analyste is developed by Emmanuel Clément*
- *FACE is a trademark of the Open Group*

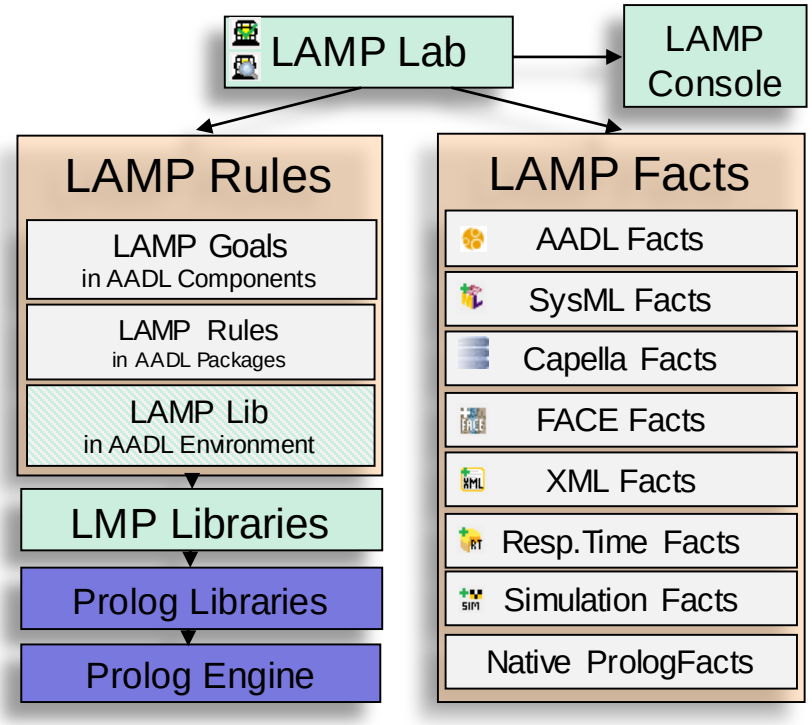
Contact: aadl@ellidiss.com

Ellidiss Technologies

LAMP outlines

LAMP: Logic AADL Model Processing

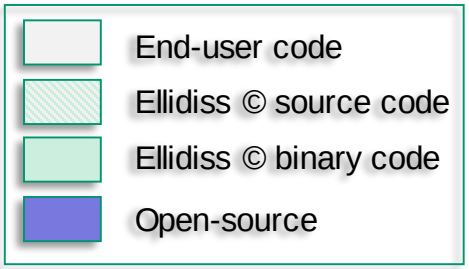
- *(symbolic)* AI inside AI !
- Prolog annex for AADL
- Embedded inference engine
- Facts bases manager (LAMPLab)
- Processing rules library (LAMPLib)




```
package ellidiss::lamp::hello
public

abstract say_hello
annex lamp {** hello **};
end say_hello;

annex lamp {**
/* plain Prolog code goes here */

hello :-
  write('hello world!'), nl.

**};
end ellidiss::lamp::hello;
```

goal

rule



Ellidiss Technologies

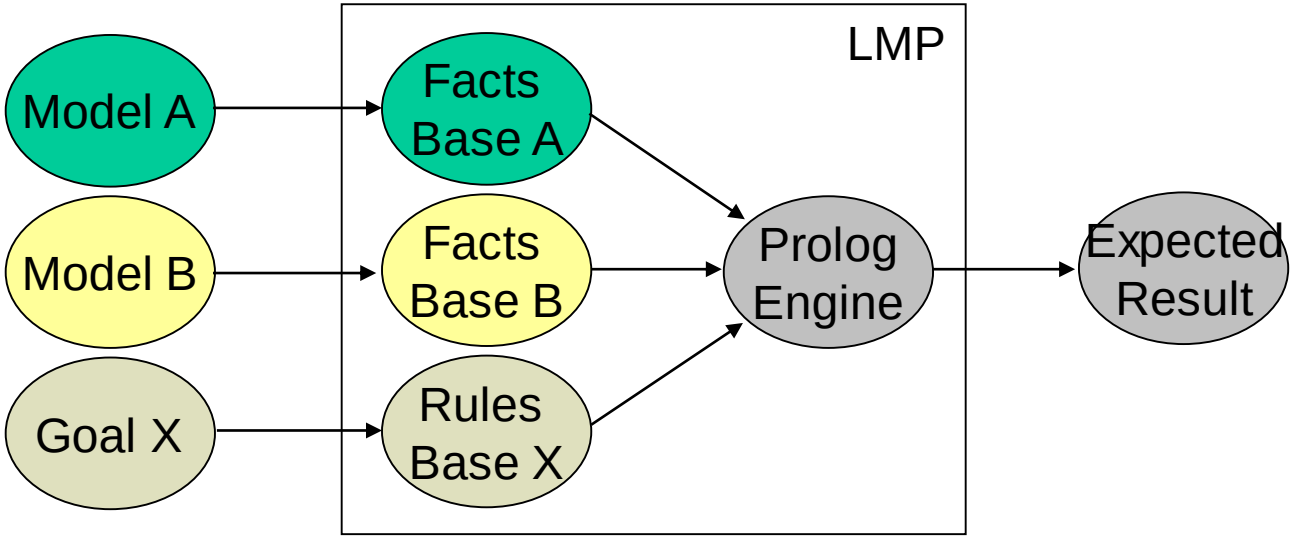
LMP:
a generic model processing approach

LMP: Logic Model Processing

- Using the Prolog language as a Model Processing language
  (Declarative predicates, Boolean logics, unification and backtracking)
- Applying a dedicated development and runtime process:

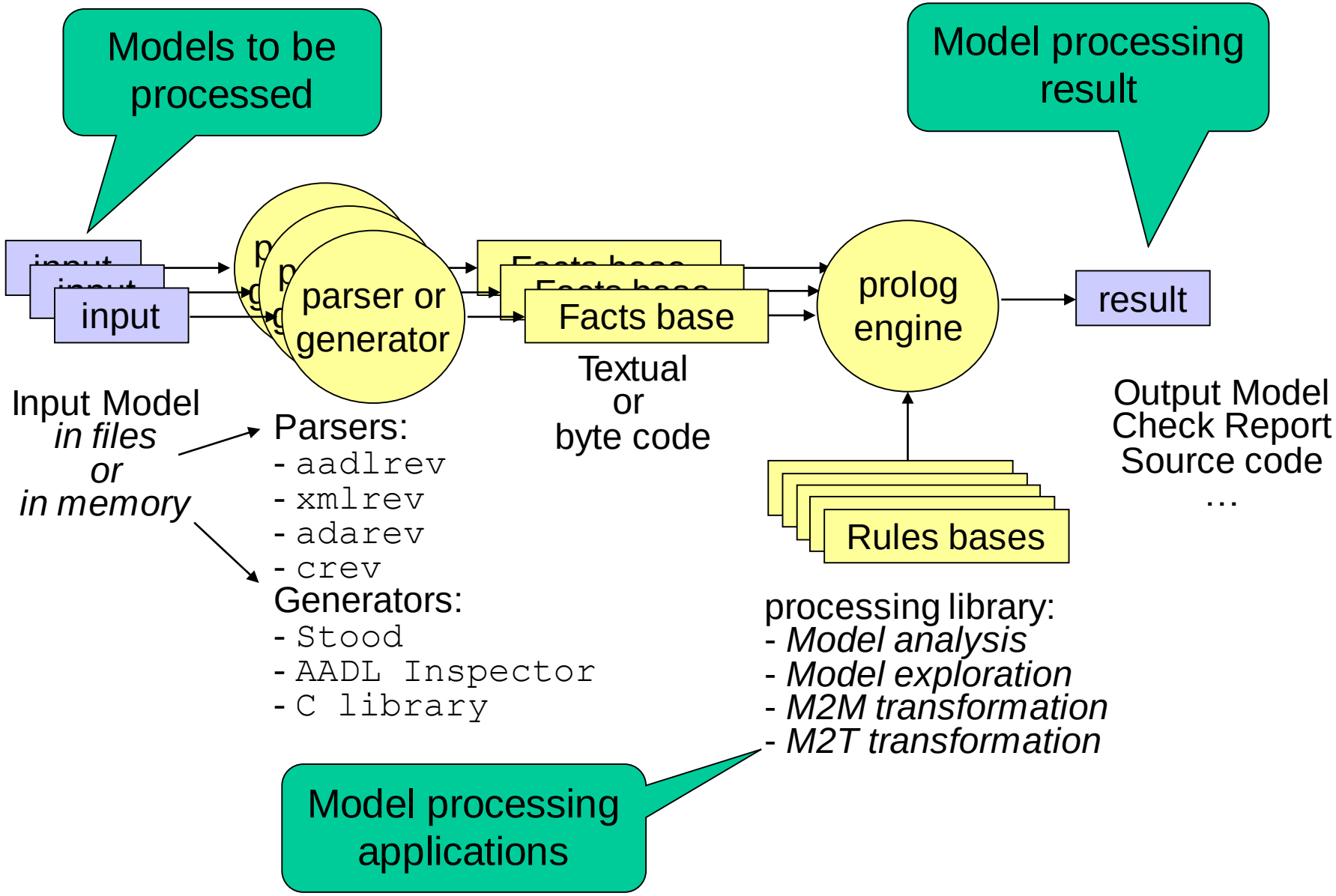
LMP development and runtime process:

- Any Meta-Model can be represented by a set of Prolog Fact Definitions
- Any Model can be represented by a populated Prolog Facts Base
- Any Model processing action (queries, constraints, transformations, …)
  can be represented by a Prolog Rules Base
- Prolog facts and rules base can be merged together to get the expected result



Ellidiss Technologies

LMP runtime process



Ellidiss Technologies

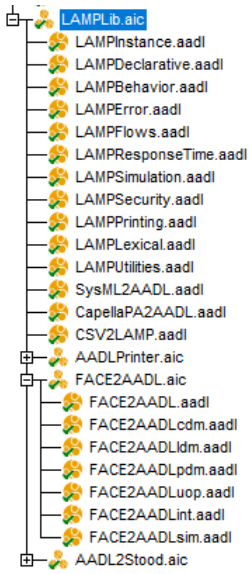
Inside LAMP

LAMP AADL Annex subclause:

- Syntax: `ANNEX LAMP {** /* standard prolog syntax */ **};`
- LAMP user defined libraries in AADL Packages
- LAMP user defined local rules in AADL Components
- No new langage to define and maintain
- Direct access to the LMP low level API (including all AADL model elements)
- Can also work on incorrect or incomplete models (debugging)

LAMP standard library: LAMPLib.aadl:

- High level API to the AADL declarative model
- High level API to the AADL instance model
- High level API to the Behavior and Error annexes
- API to analysis results (e.g. simulation traces)
- Utility rules (printing, …)
- Flow analysis and maodel transformations

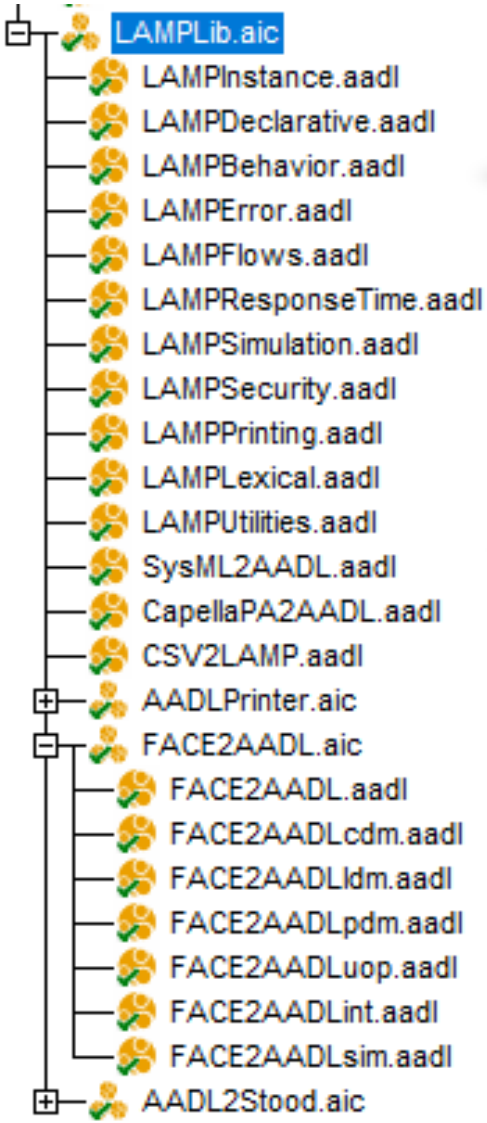


LAMP support inside AADL Inspector

- LAMP Lab: dashboard and console
- LAMP Lib is pre-loaded within the AADL «environment»
- Embedded Prolog inference engine

Ellidiss Technologies

LAMPLib 1.2
overview



AADL model accessors:
- Declarative model
- Instance model
- Behavior annex
- EMV2

AADL processing rules:
- Flows exploration
- Latency computation
- Security policies

Utility rules:
- AADL printer
- CSV parser
- Prolog libraries

Model transformations:
- FACE2AADL
- SysML2AADL
- Capella2AADL
- AADL2Stood

Ellidiss Technologies

LAMP Std Lib (1/3)

AADL core language accessors

returns all the features of a classifier (component type and ancestors)

```
package Ellidiss::LAMP::Declarative public
annex LAMP {**
  getClassFeatures(Class,Name,Categ,FClass) :-
  getClassSubcomponents(Class,Name,Categ,SClass) :- …
  getLocalProperties(Class,Name,Val,Owner) :- …
  … **};
end Ellidiss::LAMP::Declarative;
```

returns all the subcomponents of a classifier (component implementation and ancestors)

returns all the properties of a classifier (comp. type and impl. and ancestors.)

returns the root classifier (component implementation)

```
package Ellidiss::LAMP::Instance public
annex LAMP {**
  getRoot(Class) :-
  getInstances(Id,Categ,Class) :- … /* components */
  getInstances(Id,Categ,Class) :- … /* features */
  getProperties(Id,Class,Prop,Val) :- …
  … **};
end Ellidiss::LAMP::Instance;
```

returns all the component instances

returns all the feature instances

returns all the properties for the given instance element

Ellidiss Technologies

LAMP Std Lib (2/3)

AADL annexes accessors

```
package Ellidiss::LAMP::BA2 public
annex LAMP {**
  getBAVariables(Class,Name,VClass,Value) :- …
  getBAStates(Class,Name,Initial,Complete,Final):-
  getBADispatch(Class,Condition) :- …
  getBAComputation(Class,Duration) :- …
  …  **};
end Ellidiss::LAMP::BA2;
```

returns all the BA variables of a classifier

returns all the BA states of a classifier

returns all the BA dispatch conditions of a classifier

returns all the BA computation actions of a classifier

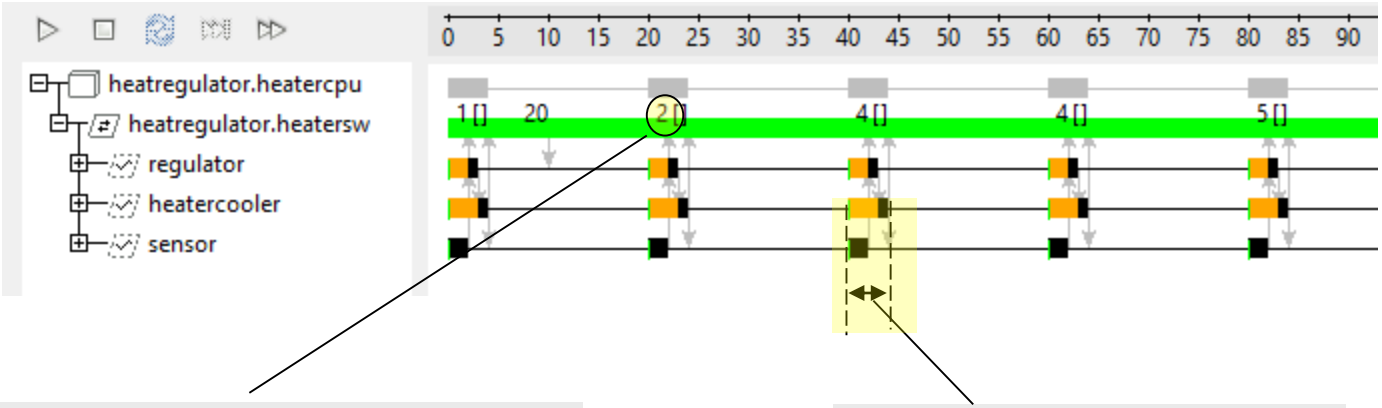
```
package Ellidiss::LAMP::EMV2 public
annex LAMP {**
  getErrorTypes(Class,Name,Ancestor) :- …
  getErrorStates(Class,Name,TypeSet,Kind) :- …
  … **};
end Ellidiss::LAMP::EMV2;
```

returns all the known error types from within a classifier

returns all the error states of a classifier

LAMP Std Lib (3/3)

Timing analysis tools feedback



returns the port value of a process or thread at output times

returns the computed response time of a thread

```
package Ellidiss::LAMP::Simulation
public
annex LAMP {**
  getPortValue(Id,Tick,Value) :- …
  …
**};
end Ellidiss::LAMP::Simulation;
```

```
package Ellidiss::LAMP::ResponseTime
public
annex LAMP {**
  getMaxResponseTime(Id,Duration,3) :- …
  …
**};
end Ellidiss::LAMP::ResponseTime;
```

Ellidiss Technologies

LAMP use cases: 

Examples of LAMP usage:

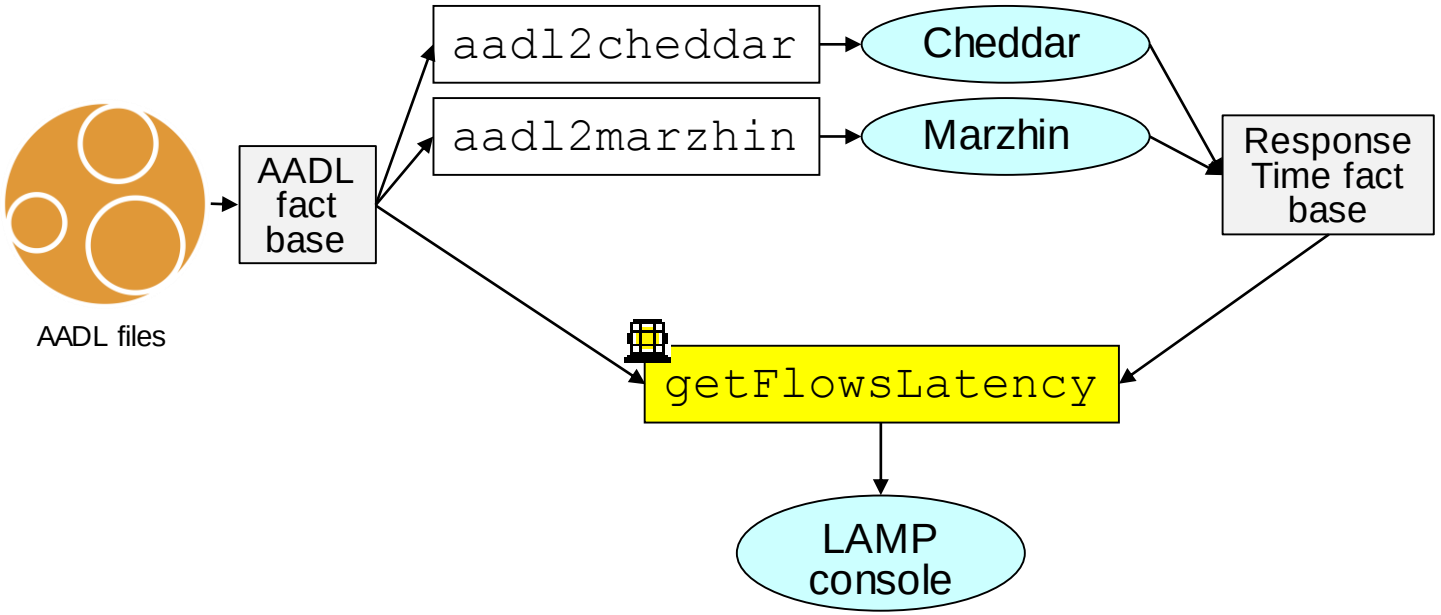
- Use case 1: Scheduling aware end to end flow latency analysis (SAFLA)
- Use case 2: System engineering models to AADL transformations
- Use case 3: AADL reverse engineering
- Use case 4: Cyber-security rules verification
- Use case 5: Customized user assurance cases

Ellidiss Technologies

LAMP use case 1:
SAFLA

Scheduling Aware Flow Latency Analysis

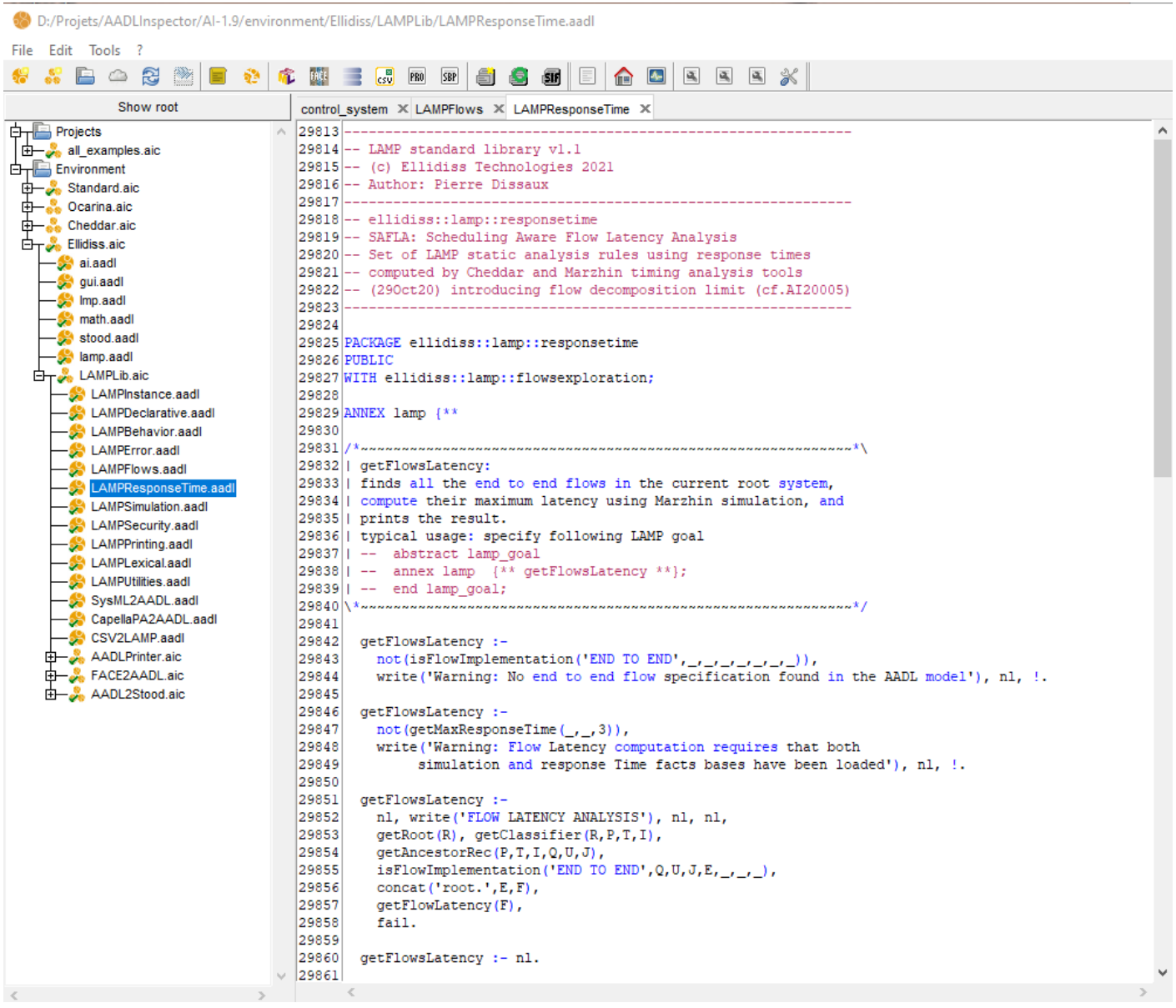
- Use scheduling analysis (Cheddar) and/or simulation (Marzhin) to estimate Threads and Bus messages Response Time
- Extract AADL Flows
- Compute End to End Flows latency based on Threads and Bus Messages estimate response time



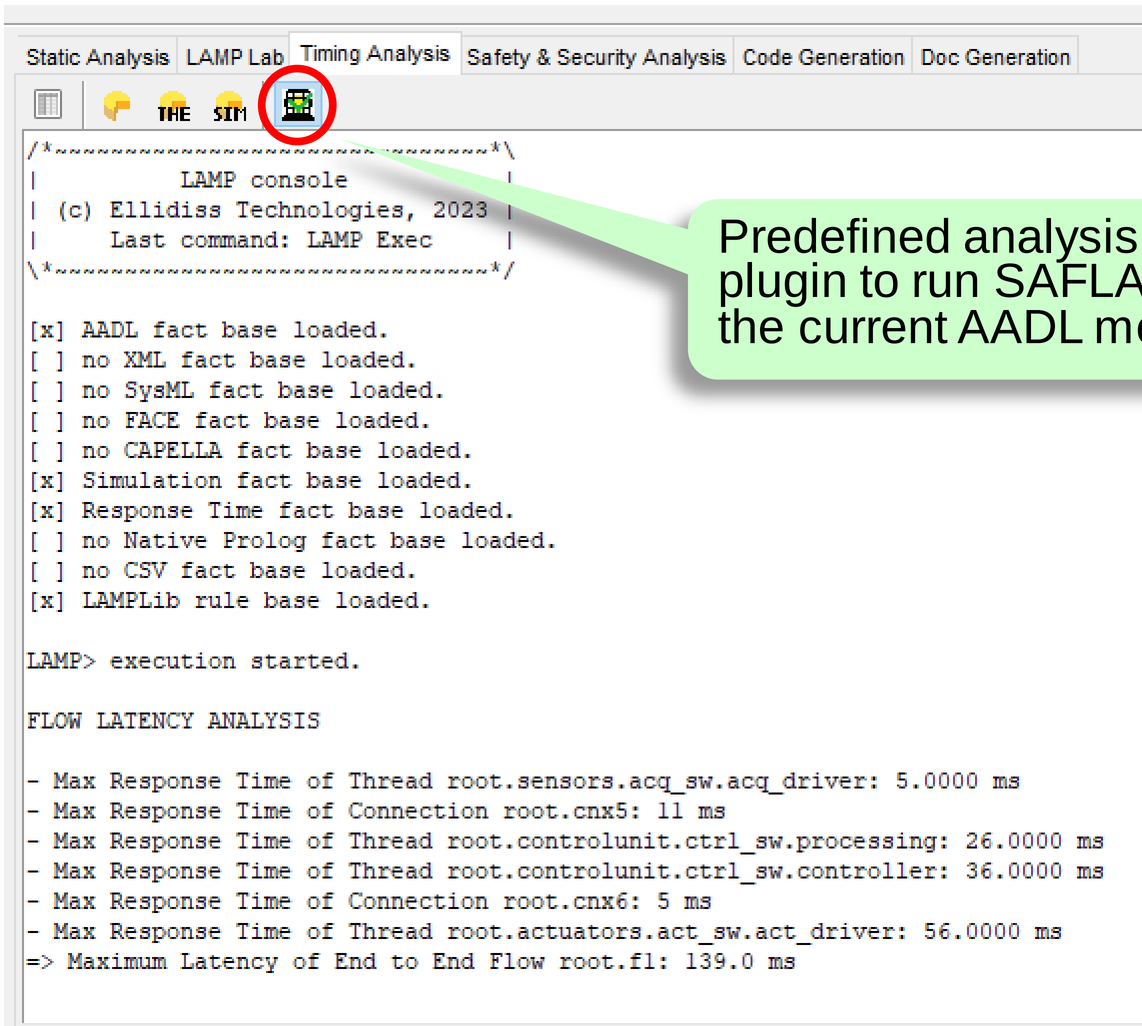
Ellidiss Technologies

LAMP use case 1:

SAFLA (cont.)



LAMP use case 1:
SAFLA (cont.)



Predefined analysis plugin to run SAFLA on the current AADL model

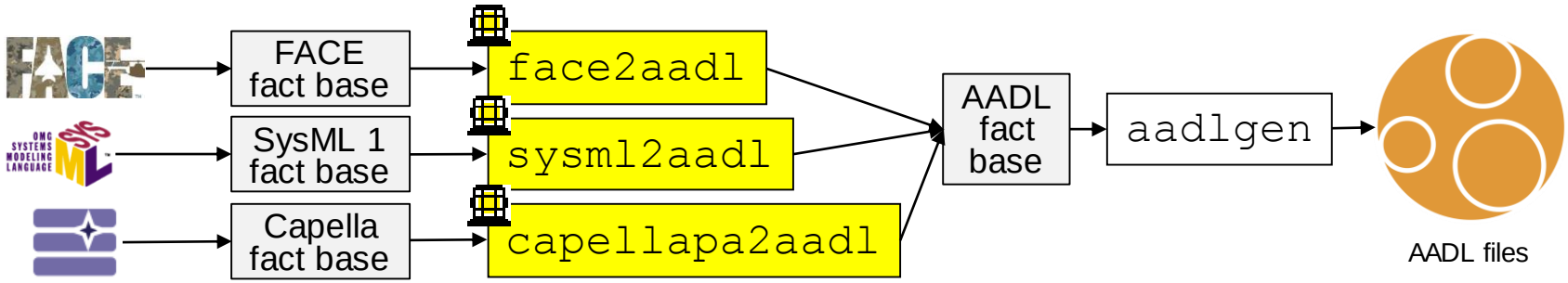
Ellidiss Technologies

LAMP use case 2:
X2AADL

Convert model X into AADL

- Currently supported LAMP transformations:

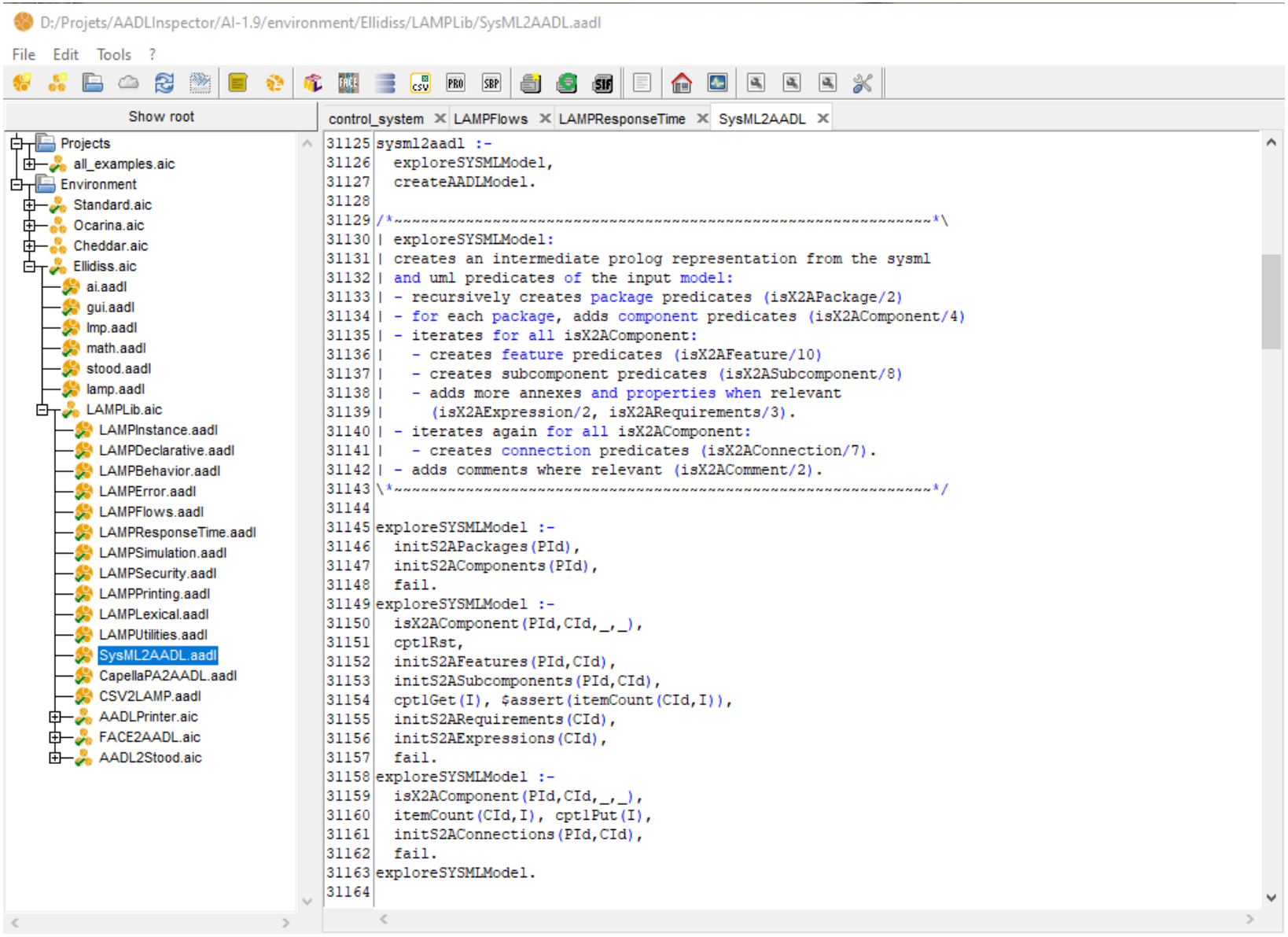
  X∈{FACE, SysMLv1, Capella}
- Parse System Model (usually XMI or XML)
- Apply X to AADL mapping rules (Prolog source code)
- Unparse AADL model (AADL printer)



Ellidiss Technologies

LAMP use case 2:

SysML2AADL

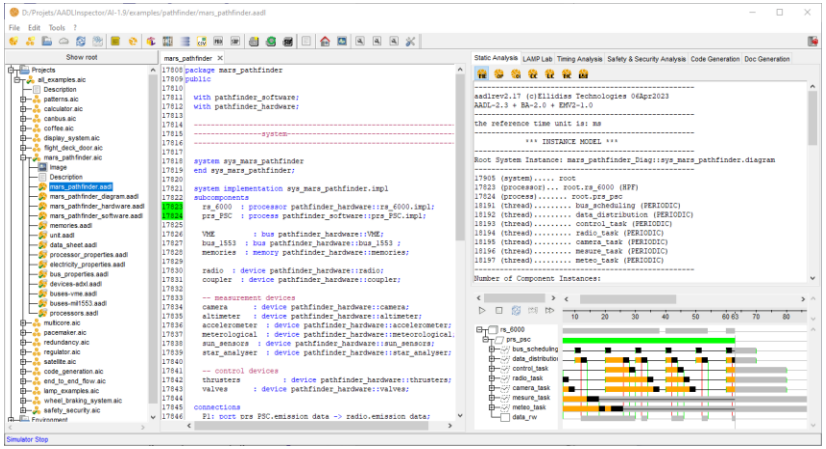


Ellidiss Technologies

LAMP use case 3:
AADL Reverse Engineering

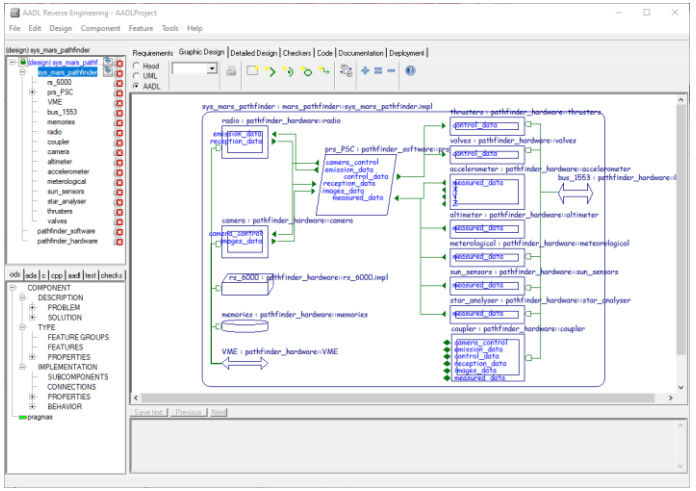
Convert an AADL Instance Model into a Stood Design

- Explore the AADL project and generate the instance model
- Generate an input file for the Stood design tool
- Launch Stood and load/update the AADL design
- Edit the AADL instance model diagram



AADL Inspector

Stood for AADL



Ellidiss Technologies

LAMP use case 3:
AADL Reverse Engineering (cont.)

```
property set stood is
  Box_Position : AADLSTRING
    applies to ( PACKAGE, SYSTEM, PROCESSOR, BUS, MEMORY, DEVICE,
                 VIRTUAL PROCESSOR, VIRTUAL BUS, ABSTRACT, MODE,
                 PROCESS, THREAD GROUP, THREAD, SUBPROGRAM, DATA );
  Link_Position : AADLSTRING
    applies to ( CONNECTION, MODE TRANSITION );
  Port_Position: AADLSTRING
    applies to ( FEATURE, FEATURE GROUP, PARAMETER );
end stood;
```

Stood diagram property set

```
system implementation FlightDeckDoor.others
…
end FlightDeckDoor.others;

system implementation FlightDeckDoor.diagram
extends FlightDeckDoor_Pkg::FlightDeckDoor.others
properties
-- Remove next line to ignore graphical extensions
  AI::root_system => "SELECTED";

-- The following properties set the diagram layout in Stood instance diagram editor
-- for the main system:
  Stood::Box_Position => "700 400 1200 800";
  Stood::Box_Position => "800 445 1100 650" applies to FDDControlSW;
  Stood::Link_Position => "1147 560 1208 560 1208 691 1270 691" applies to cnx_0;
  Stood::Link_Position => "584 366 680 366 680 511 789 511" applies to cnx_1;
  Stood::Port_Position => "left" applies to FDDControlSW.code;
  Stood::Port_Position => "left" applies to FDDControlSW.selector;
…
end FlightDeckDoor.diagram;
```

Root system

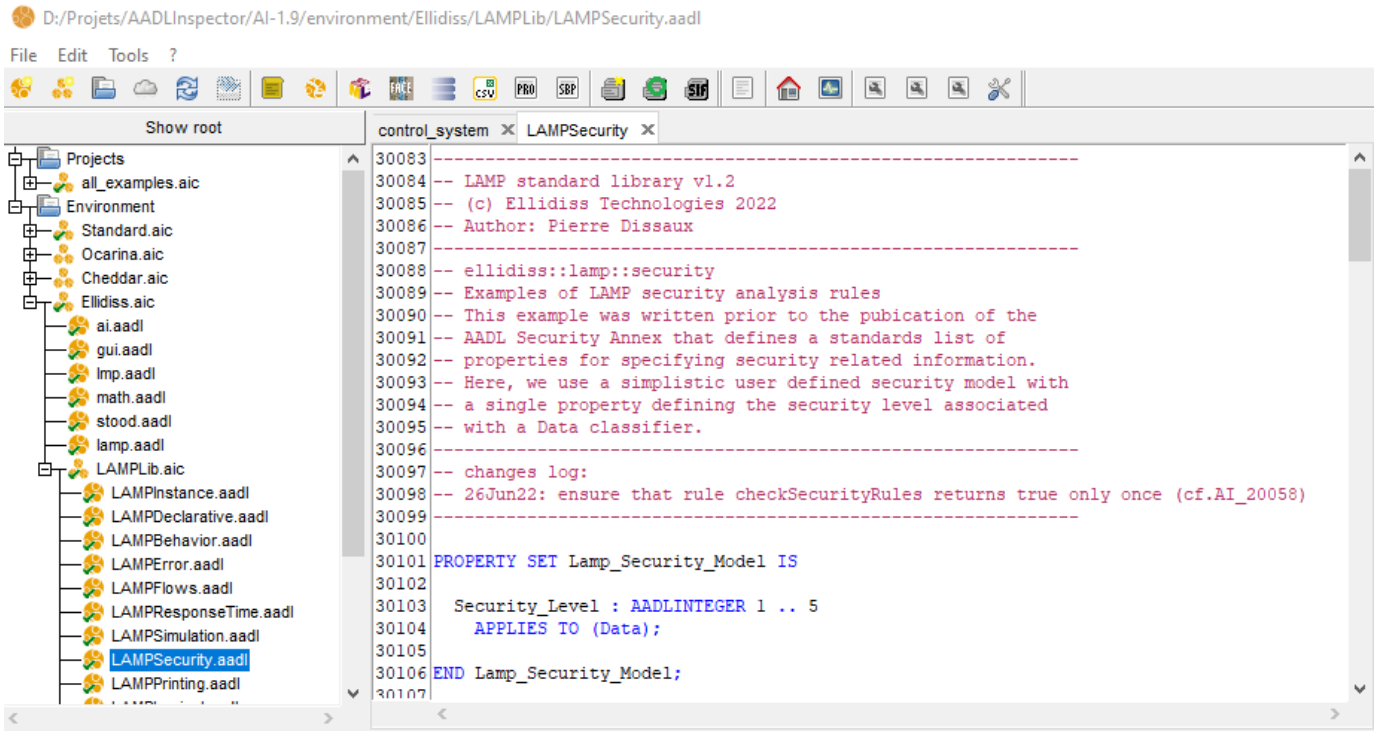
Root system extension with diagram properties

Ellidiss Technologies

LAMP use case 4:
Security policy checker

Template to implement security rules with LAMP

- AADL Property Set: LAMP_Security_Model
- Security policy specification: which security rules to check ?
- Security rules implementation
- Just a template to be customized for real cases



Ellidiss Technologies

LAMP use case 4:

Security policy checker (cont.)

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
| checkSecurityRules:
| check all security rules in sequence
| typical usage: specify following LAMP goal
| --  abstract lamp_goal
| --  annex lamp  {** checkSecurityRules **};
| --  end lamp_goal;
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

  checkSecurityRules :-
    ( isPropertyOpt(_,_,_,_,_,_,'LAMP_SECURITY_MODEL::SECURITY_LEVEL',_,_) ->
      ( write('Information: Security rules can be customized in file: '), nl,
        write('  environment/Ellidiss/LAMPLib/LAMPSecurity.aadl'), nl );
      ( write('Warning: No Security Level Property found in the AADL model'), nl, !) ),
    fail.
  checkSecurityRules :-
    nl, write('SECURITY ANALYSIS'), nl, nl,
    /* Sec_R1 */ checkFlowSecurity,
    /* Sec_R2 */ checkMaxSecurityLevel,
    /* Sec_R3 */ checkNoWriteDown,
    fail.
  checkSecurityRules :- nl.
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
| checkFlowSecurity:
| check security rule example Sec_R1 and print results
| Rule Sec_R1:
| "All components involved in a same end to end Flow
|  must be at the same security level."
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

  checkFlowSecurity :-
    getRoot(R), getClassifier(R,P,T,I),
    getAncestorRec(P,T,I,Q,U,J),
    isFlowImplementation('END TO END',Q,U,J,E,_,_,_),
    concat('root.',E,F),
    getEndToEndFlow('root',E,M),
    getFlowSecurityLevels(M,[],L,0,N), N > 1,
    printMessageSec_R1(F,L),
    fail.
  checkFlowSecurity :- nl.

  printMessageSec_R1(F,L) :-
    write('/!\ Security rule R1 (ERROR) : end to end flow '), write(F), nl,
    write('    has several several security levels: '),
    $member(A,L), write(A), sp, fail.
  printMessageSec_R1(_,_) :- nl.
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
| checkNoWriteDown:
| check security rule example Sec_R3 and print results
| Rule Sec_R3:
| "When two components are connected via a shared Bus,
|  they must comply with the No-Write-Down rule."
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

  checkNoWriteDown :-
    isAADLBusBinding(_,C,_),
    isAADLConnection(_,P,T,I,_,_,_,C,_,_,_,_),
    getConnectionEnds(P,T,I,C,Xs,Xd),
    getMaxSecurityLevel(Xs,Ls),
    getMaxSecurityLevel(Xd,Ld),
    Ls > Ld,
    printMessageSec_R3(C,Ls,Ld),
    fail.
  checkNoWriteDown :- nl.

  printMessageSec_R3(C,S,D) :-
    write('/!\ Security rule R3 (ERROR) : connection '), write(C), nl,
    write('    is between security levels: '),
    write(S), write(' and '), write(D), fail.
  printMessageSec_R3(_,_,_) :- nl.
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
| checkMaxSecurityLevel:
| check security rule example Sec_R2 and print results
| Rule Sec_R2:
| "The security level of a component is the higher security level
|  value associated with its Data ports."
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

  checkMaxSecurityLevel :-
    getMaxSecurityLevel(X,L),
    printMessageSec_R2(X,L),
    fail.
  checkMaxSecurityLevel :- nl.

  printMessageSec_R2(X,L) :-
    write('/!\ Security rule R2 (INFORMATION) : component '), write(X), nl,
    write('    is at security level: '), write(L), fail.
  printMessageSec_R2(_,_) :- nl.
```

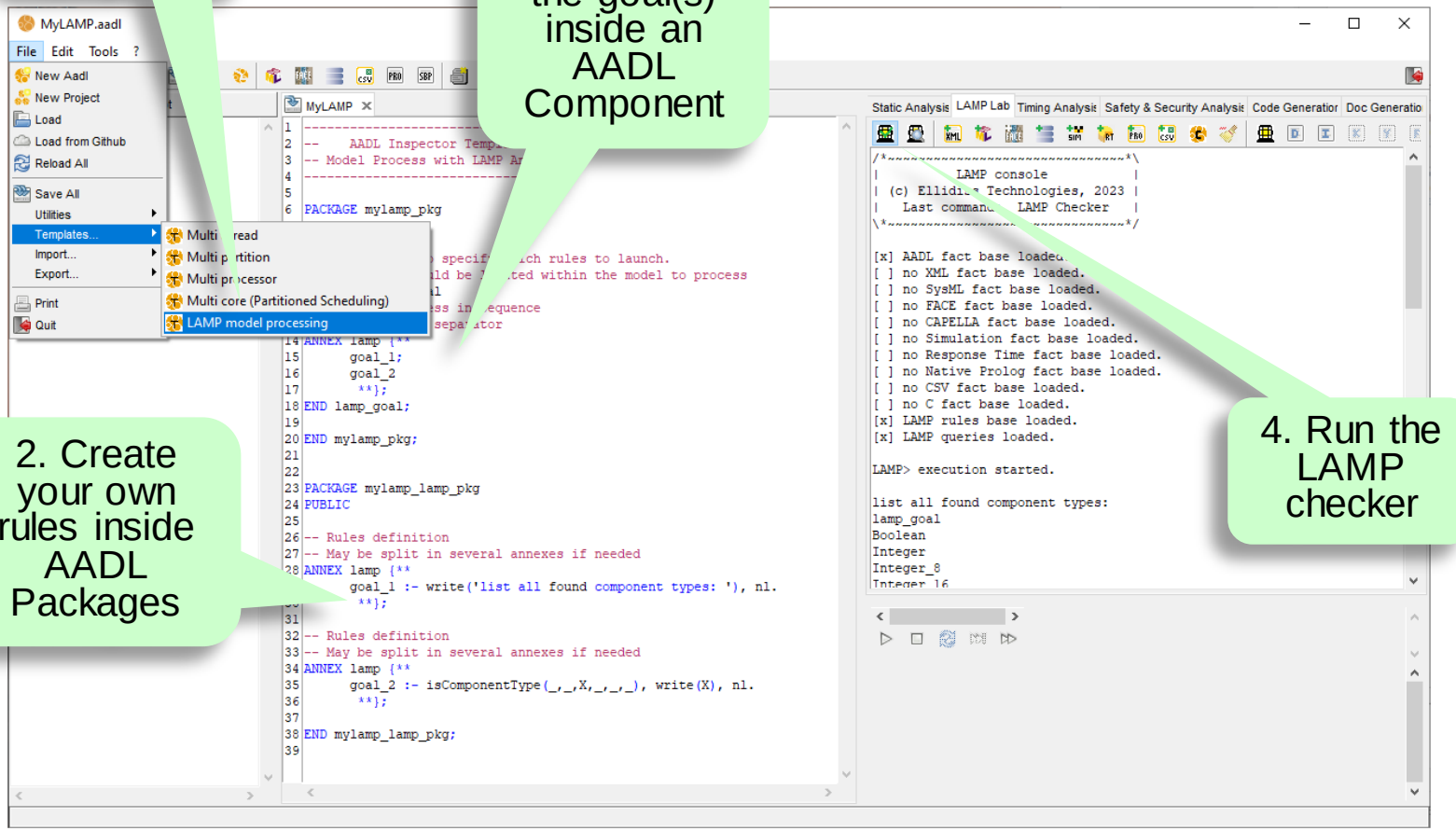
Ellidiss Technologies

LAMP use case 5:

Do it Yourself !

1. Use the predefined template

3. Specify the goal(s) inside an AADL Component



2. Create your own rules inside AADL Packages

4. Run the LAMP checker

Ellidiss Technologies

Tools availability

Current products availability

`https://www.ellidiss.com/downloads`

- Stood for AADL 5.5.1 (Linux)
- Stood for AADL 5.5.2 update 3 (Windows)
- AADL Inspector 1.9 (Windows & Linux) 

`https://marketplace.visualstudio.com`

- Visual Studio Code Extension for AADL 1.1.3
- Visual Studio Code Extension for SysMLv2 0.3.0

On-going work

- SysML v2 / AADL mapping and bridger
- AADL Inspector 2.0
  - New graphical front end
  - New plugins framework (MPT)
  - Enhanced LAMPLib