

<http://www.ellidiss.com>

LAMP:
to shed light on AADL models

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:

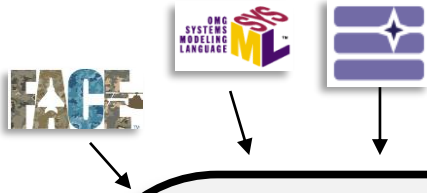
- Architectural SW Design: HOOD, AADL, SysML v2
- Early SW Verification: Real-Time, Safety, Security, Assurance Cases
- Logic Model Processing (LMP): Model Exploration, Verification, Transformation

Collaborative R&D projects:

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

Our AADL tools ecosystem (not osate-based!)

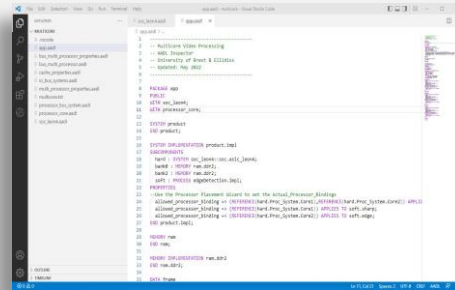
System design



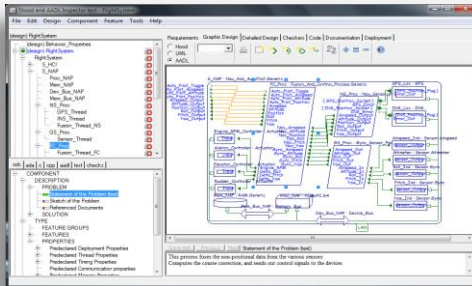
SW requirements

SW design

Textual design

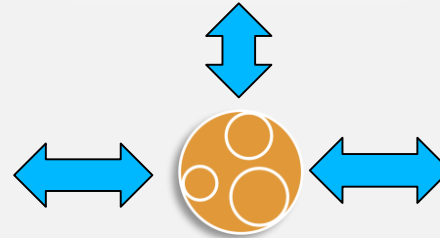


Graphical design



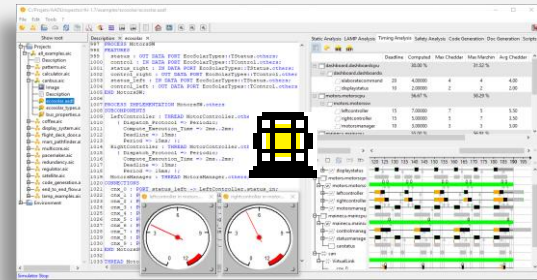
Stood for AADL

VSCoDe extension
for AADL



AADL files

Early verification



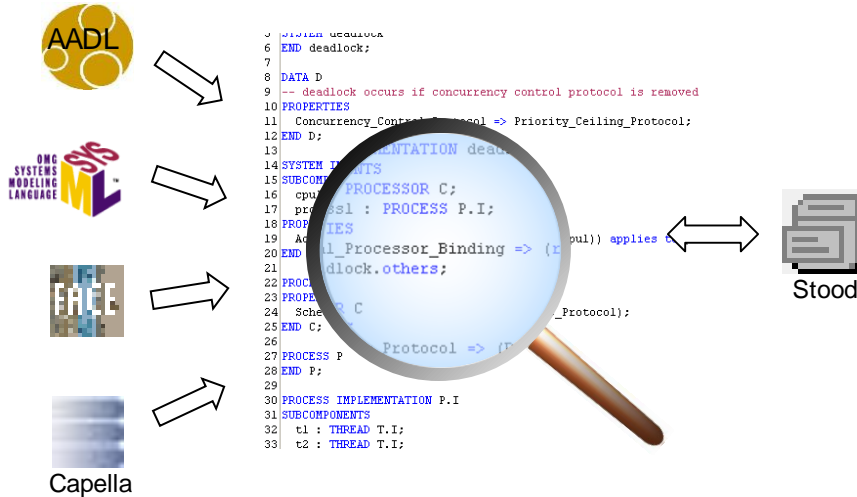
AADL Inspector

SW coding

AADL Inspector 1.9

model processing framework

new release



Acknowledgements:

- Cheddar is developed by the University of Brest
- Ocarina is developed by Telecom ParisTech, 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

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 Analyste
- 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

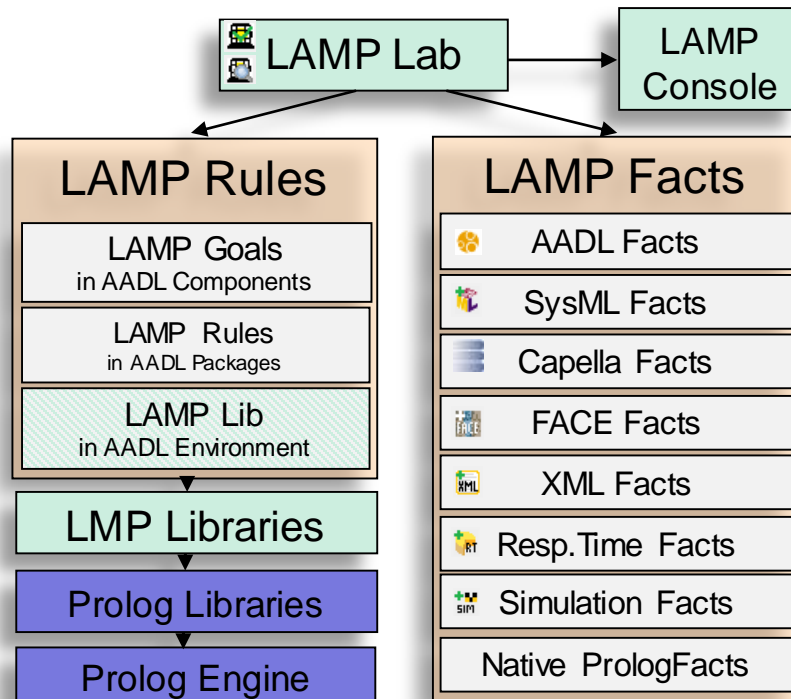
Contact: aadl@ellidiss.com





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 */

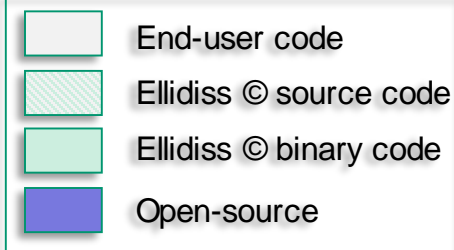
hello :-
    write('hello world!'), nl.

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

```

goal

rule



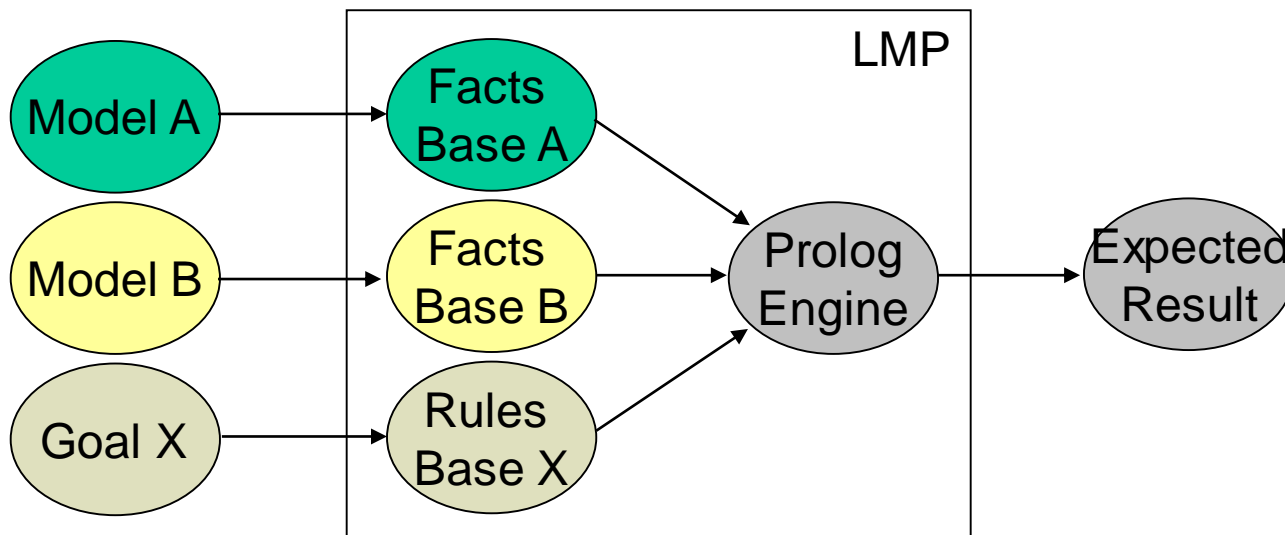
LMP: a generic model processing approach

LMP: Logic Model Processing

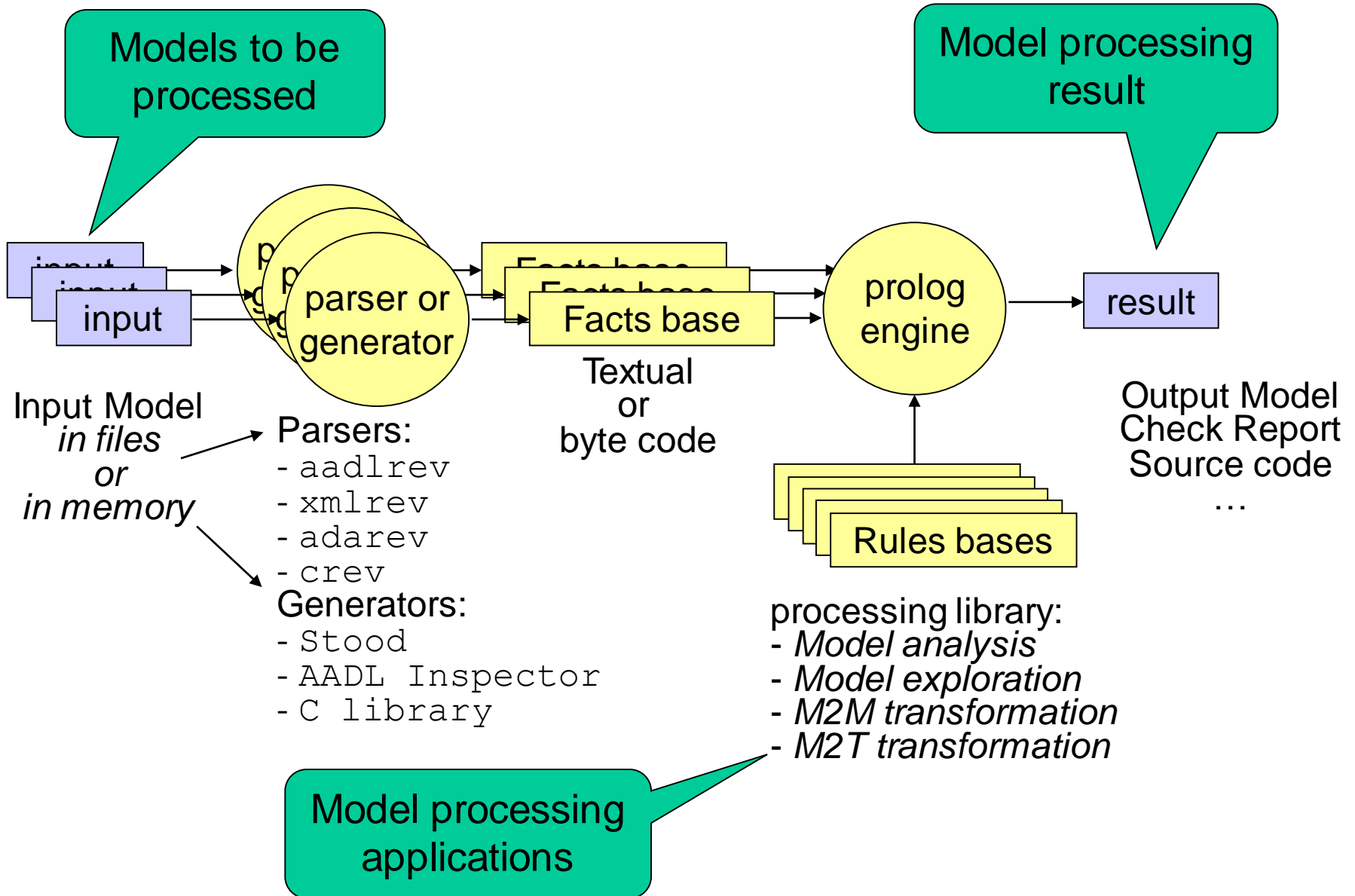
- Using the Prolog language as a Model Processing language (Declarative predicates, Boolean logics, unification and backtracking)
- Applying a dedicated development and runtime process:

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



LMP runtime process





LAMP AADL Annex subclause:

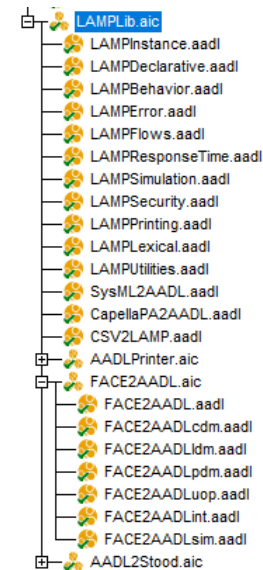
- Syntax: ANNEX LAMP {** /* standard prolog syntax */ **};
- LAMP user defined libraries in AADL Packages
- LAMP user defined local rules in AADL Components
- No new language to define and maintain
- Direct access to the LAMP low level API (including all AADL model elements)
- Can also work on incorrect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

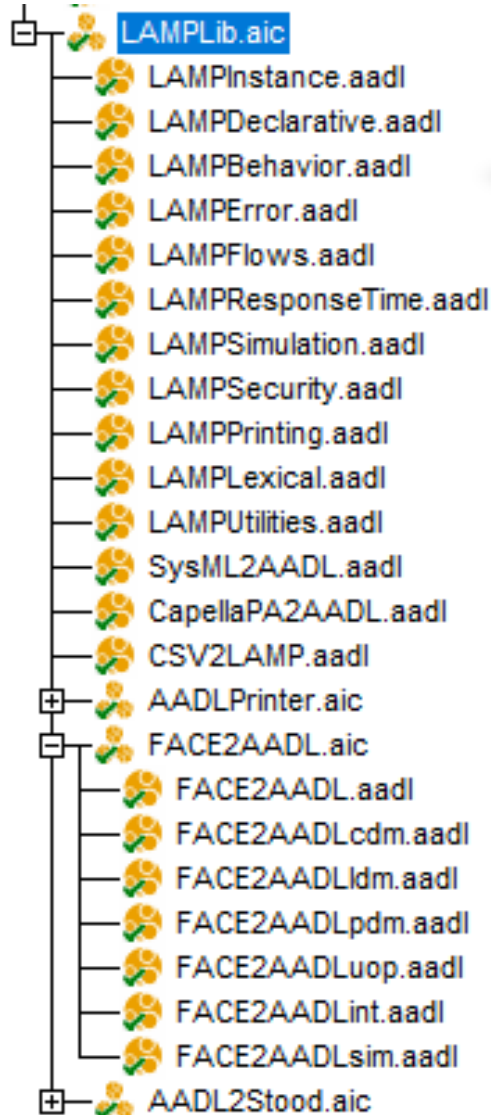
LAMP support inside AADL Inspector

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



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

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.)

```
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 the root classifier
(component implementation)

returns all the component instances

returns all the feature instances

returns all the properties for the
given instance element



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::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;
```

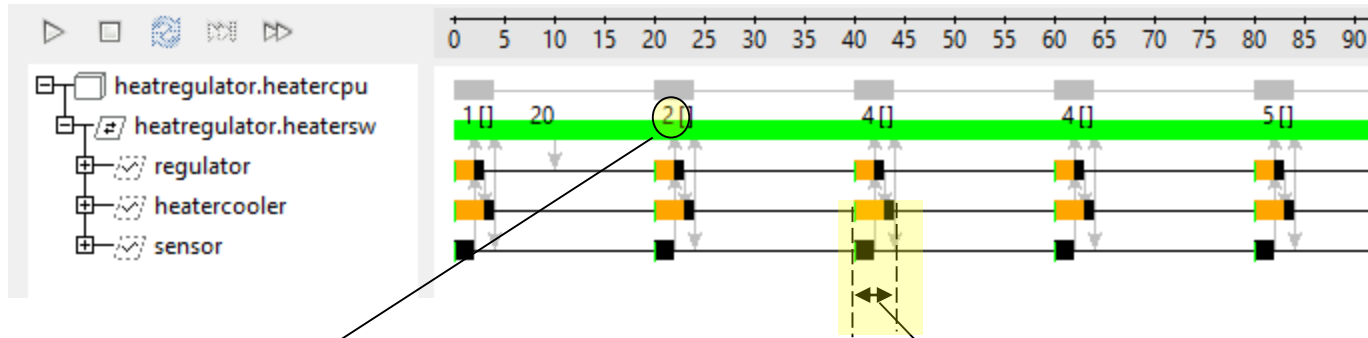
returns all the known error types
from within a classifier

returns all the error states of a classifier

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

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;
  
```

LAMP use cases:

Examples of LAMP usage:

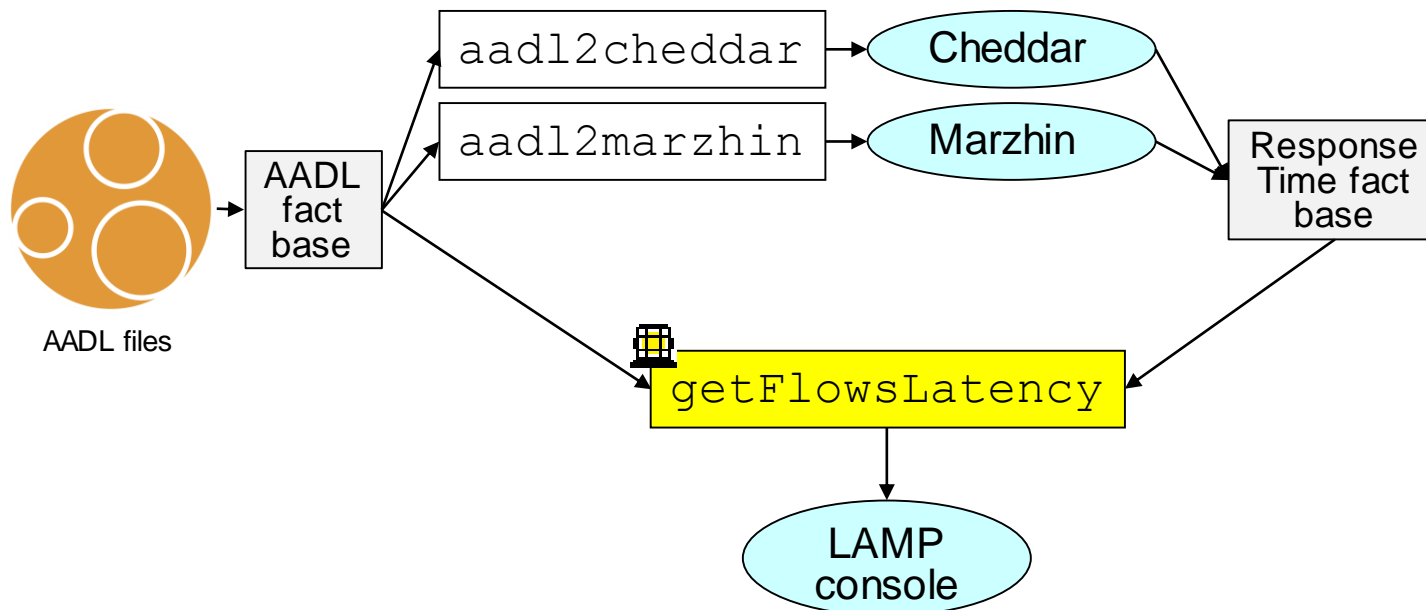
- Use case 1: Scheduling aware end to end flow latency analysis (SAFLA)
- Use case 2: System engineering models to AADL transformations
- Use case 3: AADL reverse engineering
- Use case 4: Cyber-security rules verification
- Use case 5: Customized user assurance cases

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



LAMP use case 1: SAFLA (cont.)



D:/Projets/AADLInspector/AI-1.9/environment/Ellidiss/LAMPLib/LAMPResponseTime.aadl

File Edit Tools ?

Show root

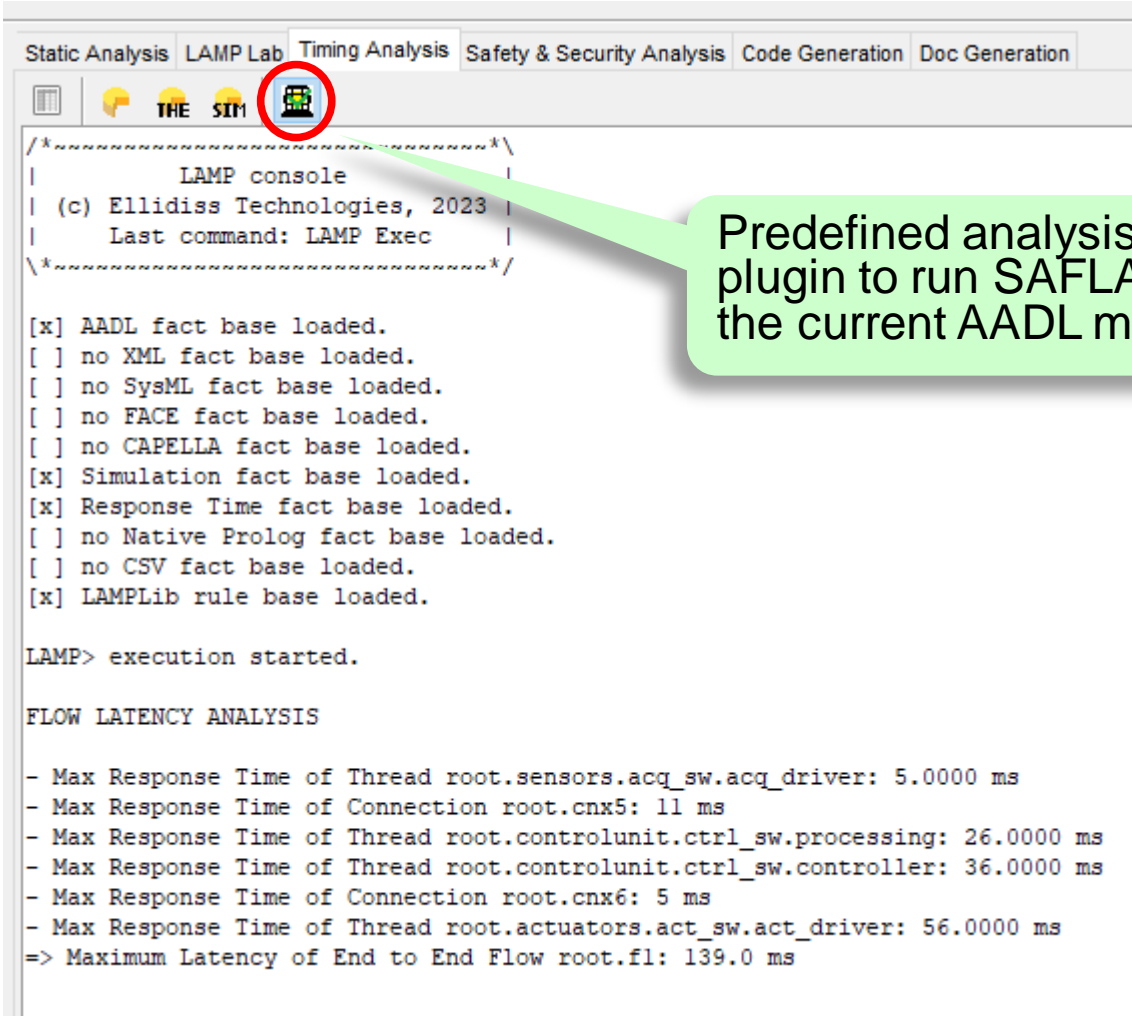
- Projects
 - all_examples.aic
 - Environment
 - Standard.aic
 - Ocarina.aic
 - Cheddar.aic
 - Ellidiss.aic
 - ai.aadl
 - gui.aadl
 - imp.aadl
 - math.aadl
 - stood.aadl
 - lamp.aadl
 - LAMPLib.aic
 - LAMPInstance.aadl
 - LAMPDeclarative.aadl
 - LAMPBehavior.aadl
 - LAMPError.aadl
 - LAMPFlows.aadl
 - LAMPResponseTime.aadl**
 - LAMPSimulation.aadl
 - LAMPSecurity.aadl
 - LAMPPrinting.aadl
 - LAMPLexical.aadl
 - LAMPUtilities.aadl
 - SysML2AADL.aadl
 - CapellaPA2AADL.aadl
 - CSV2LAMP.aadl
 - AADLPrinter.aic
 - FACE2AADL.aic
 - AADL2Stood.aic

```

29813 -----
29814 -- LAMP standard library v1.1
29815 -- (c) Ellidiss Technologies 2021
29816 -- Author: Pierre Dissaux
29817 -----
29818 -- ellidiss::lamp::responsetime
29819 -- SAFLA: Scheduling Aware Flow Latency Analysis
29820 -- Set of LAMP static analysis rules using response times
29821 -- computed by Cheddar and Marzhin timing analysis tools
29822 -- (29Oct20) introducing flow decomposition limit (cf.AI20005)
29823 -----
29824
29825 PACKAGE ellidiss::lamp::responsetime
29826 PUBLIC
29827 WITH ellidiss::lamp::flowsexploration;
29828
29829 ANNEX lamp {**
29830
29831 /*~~~~~*/
29832 | getFlowsLatency:
29833 | finds all the end to end flows in the current root system,
29834 | compute their maximum latency using Marzhin simulation, and
29835 | prints the result.
29836 | typical usage: specify following LAMP goal
29837 | -- abstract lamp_goal
29838 | -- annex lamp {** getFlowsLatency **};
29839 | -- end lamp_goal;
29840 /*~~~~~*/
29841
29842 getFlowsLatency :-
29843   not(isFlowImplementation('END TO END',_,_,_,_,_)),
29844   write('Warning: No end to end flow specification found in the AADL model'), nl, !.
29845
29846 getFlowsLatency :-
29847   not(getMaxResponseTime(_,_,3)),
29848   write('Warning: Flow Latency computation requires that both
29849     simulation and response Time facts bases have been loaded'), nl, !.
29850
29851 getFlowsLatency :-
29852   nl, write('FLOW LATENCY ANALYSIS'), nl, nl,
29853   getRoot(R), getClassifier(R,P,T,I),
29854   getAncestorRec(P,T,I,Q,U,J),
29855   isFlowImplementation('END TO END',Q,U,J,E,_,_,_),
29856   concat('root.',E,F),
29857   getFlowLatency(F),
29858   fail.
29859
29860 getFlowsLatency :- nl.
29861
  
```

LAMP use case 1:

SAFLA (cont.)



The screenshot shows the LAMP Lab software interface. The 'Timing Analysis' tab is active, and a red circle highlights the train icon in the toolbar. A green callout box points to the train icon with the text: 'Predefined analysis plugin to run SAFLA on the current AADL model'. The console window displays the following output:

```
/*.....*/
|           LAMP console           |
| (c) Ellidiss Technologies, 2023 |
|   Last command: LAMP Exec       |
/*.....*/

[x] AADL fact base loaded.
[ ] no XML fact base loaded.
[ ] no SysML fact base loaded.
[ ] no FACE fact base loaded.
[ ] no CAPELLA fact base loaded.
[x] Simulation fact base loaded.
[x] Response Time fact base loaded.
[ ] no Native Prolog fact base loaded.
[ ] no CSV fact base loaded.
[x] LAMPLib rule base loaded.

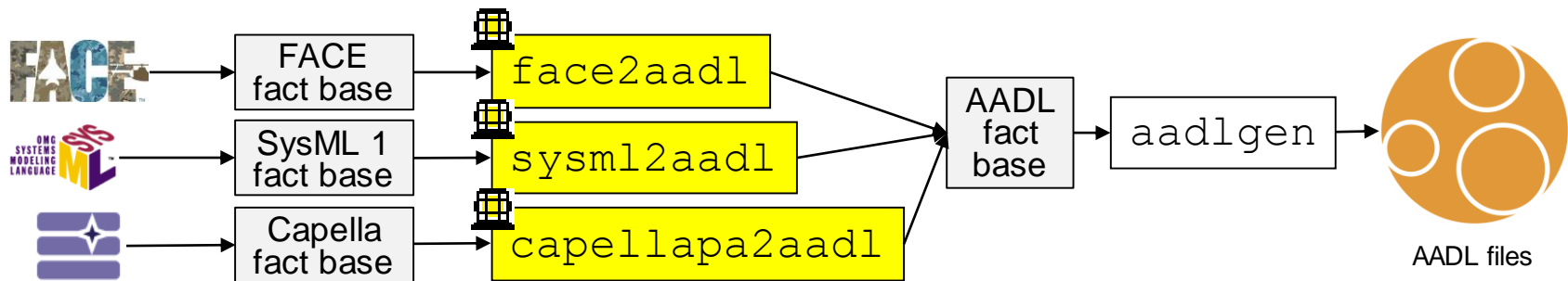
LAMP> execution started.

FLOW LATENCY ANALYSIS

- Max Response Time of Thread root.sensors.acq_sw.acq_driver: 5.0000 ms
- Max Response Time of Connection root.cnx5: 11 ms
- Max Response Time of Thread root.controlunit.ctrl_sw.processing: 26.0000 ms
- Max Response Time of Thread root.controlunit.ctrl_sw.controller: 36.0000 ms
- Max Response Time of Connection root.cnx6: 5 ms
- Max Response Time of Thread root.actuators.act_sw.act_driver: 56.0000 ms
=> Maximum Latency of End to End Flow root.fl: 139.0 ms
```


Convert model X into AADL

- Currently supported LAMP transformations:
 $X \in \{\text{FACE, SysMLv1, Capella}\}$
- Parse System Model (usually XMI or XML)
- Apply X to AADL mapping rules (Prolog source code)
- Unparse AADL model (AADL printer)



LAMP use case 2:

SysML2AADL



D:/Projets/AADLInspector/AI-1.9/environment/Ellidiss/LAMPLib/SysML2AADL.aadl

File Edit Tools ?

Show root

- Projects
 - all_examples.aic
 - Environment
 - Standard.aic
 - Ocarina.aic
 - Cheddar.aic
 - Ellidiss.aic
 - ai.aadl
 - gui.aadl
 - imp.aadl
 - math.aadl
 - stood.aadl
 - lamp.aadl
 - LAMPLib.aic
 - LAMPInstance.aadl
 - LAMPDeclarative.aadl
 - LAMPBehavior.aadl
 - LAMPError.aadl
 - LAMPFlows.aadl
 - LAMPResponseTime.aadl
 - LAMPSimulation.aadl
 - LAMPSecurity.aadl
 - LAMPPrinting.aadl
 - LAMPLexical.aadl
 - LAMPUtilities.aadl
 - SysML2AADL.aadl**
 - CapellaPA2AADL.aadl
 - CSV2LAMP.aadl
 - AADLPrinter.aic
 - FACE2AADL.aic
 - AADL2stood.aic

```

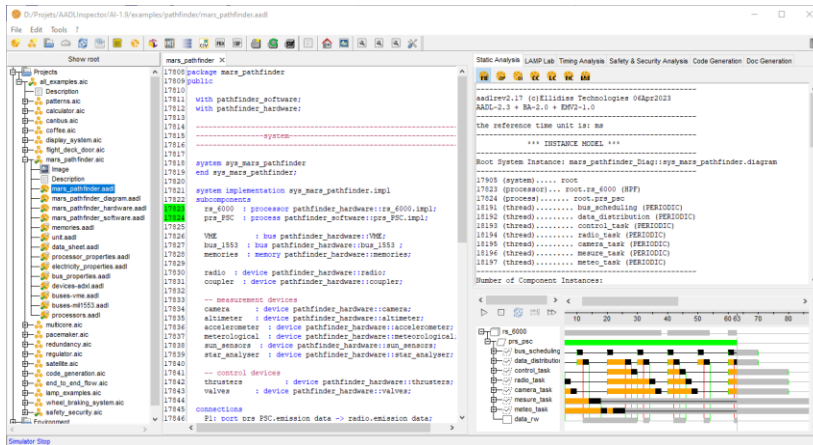
31125 sysml2aadl :-
31126   exploreSYSMLModel,
31127   createAADLModel.
31128
31129 /*~~~~~*/
31130 | exploreSYSMLModel:
31131 | creates an intermediate prolog representation from the sysml
31132 | and uml predicates of the input model:
31133 | - recursively creates package predicates (isX2APackage/2)
31134 | - for each package, adds component predicates (isX2AComponent/4)
31135 | - iterates for all isX2AComponent:
31136 |   - creates feature predicates (isX2AFeature/10)
31137 |   - creates subcomponent predicates (isX2ASubcomponent/8)
31138 |   - adds more annexes and properties when relevant
31139 |     (isX2AExpression/2, isX2ARequirements/3).
31140 | - iterates again for all isX2AComponent:
31141 |   - creates connection predicates (isX2AConnection/7).
31142 | - adds comments where relevant (isX2AComment/2).
31143 /*~~~~*/
31144
31145 exploreSYSMLModel :-
31146   initS2APackages(PId),
31147   initS2AComponents(PId),
31148   fail.
31149 exploreSYSMLModel :-
31150   isX2AComponent(PId,CId,_,_),
31151   cpt1Rst,
31152   initS2AFeatures(PId,CId),
31153   initS2ASubcomponents(PId,CId),
31154   cpt1Get(I), $assert(itemCount(CId,I)),
31155   initS2ARequirements(CId),
31156   initS2AExpressions(CId),
31157   fail.
31158 exploreSYSMLModel :-
31159   isX2AComponent(PId,CId,_,_),
31160   itemCount(CId,I), cpt1Put(I),
31161   initS2AConnections(PId,CId),
31162   fail.
31163 exploreSYSMLModel.
31164
  
```

LAMP use case 3: AADL Reverse Engineering

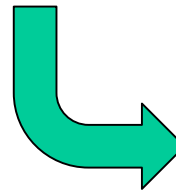


Convert an AADL Instance Model into a Stood Design

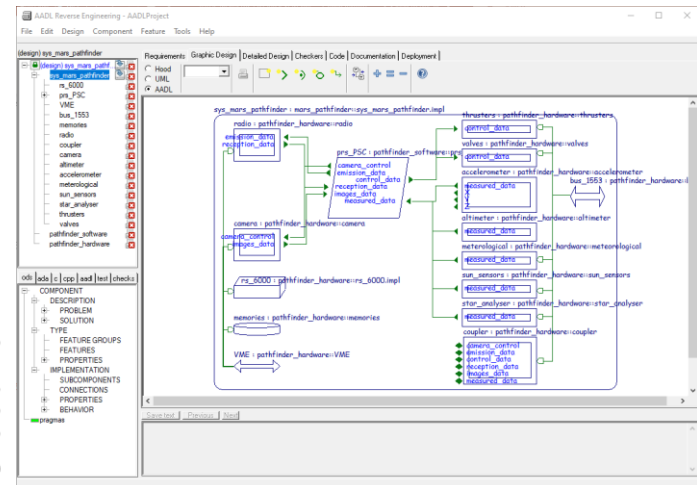
- Explore the AADL project and generate the instance model
- Generate an input file for the Stood design tool
- Launch Stood and load/update the AADL design
- Edit the AADL instance model diagram



AADL Inspector



Stood for AADL



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

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

D:/Projets/AADLInspector/AI-1.9/environment/Ellidiss/LAMPLib/LAMPSecurity.aadl

File Edit Tools ?

Show root

```

control_system x LAMPSecurity x
30083 -----
30084 -- LAMP standard library v1.2
30085 -- (c) Ellidiss Technologies 2022
30086 -- Author: Pierre Dissaux
30087 -----
30088 -- ellidiss::lamp::security
30089 -- Examples of LAMP security analysis rules
30090 -- This example was written prior to the publication of the
30091 -- AADL Security Annex that defines a standards list of
30092 -- properties for specifying security related information.
30093 -- Here, we use a simplistic user defined security model with
30094 -- a single property defining the security level associated
30095 -- with a Data classifier.
30096 -----
30097 -- changes log:
30098 -- 26Jun22: ensure that rule checkSecurityRules returns true only once (cf.AI_20058)
30099 -----
30100
30101 PROPERTY SET Lamp_Security_Model IS
30102
30103     Security_Level : ADLINTEGER 1 .. 5
30104     APPLIES TO (Data);
30105
30106 END Lamp_Security_Model;
30107
  
```

Projects

- all_examples.aic
- Environment
 - Standard.aic
 - Ocarina.aic
 - Cheddar.aic
 - Ellidiss.aic
 - ai.aadl
 - gui.aadl
 - imp.aadl
 - math.aadl
 - stood.aadl
 - lamp.aadl
 - LAMPLib.aic
 - LAMPInstance.aadl
 - LAMPDeclarative.aadl
 - LAMPBehavior.aadl
 - LAMPError.aadl
 - LAMPFlows.aadl
 - LAMPResponseTime.aadl
 - LAMPSimulation.aadl
 - LAMPSecurity.aadl
 - LAMPPrinting.aadl

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.

/*.....*/
| 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.

/*.....*/
| 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.

/*.....*/
| 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.

```

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

The screenshot displays the Ellidiss LAMP IDE interface. On the left, the 'File' menu is open, with 'Templates...' selected, showing a list of predefined templates including 'Multi read', 'Multi partition', 'Multi processor', 'Multi core (Partitioned Scheduling)', and 'LAMP model processing'. The main editor window shows AADL code for 'MyLAMP.aadl'. The code includes package declarations, annotations, and goal definitions. A green callout bubble points to the 'LAMP model processing' template in the menu. Another callout bubble points to the goal definitions in the code. A third callout bubble points to the 'ANNEX lamp' section where rules are defined. On the right, the 'LAMP Lab' console window shows the output of the LAMP checker, including a list of found component types: 'lamp_goal', 'Boolean', 'Integer', 'Integer_8', and 'Integer_16'. A green callout bubble points to the console output.

```
1  -- AADL Inspector Template
2  -- Model Process with LAMP Annotations
3
4
5
6  PACKAGE mylamp_pkg
7
8
9
10
11
12
13
14 ANNEX lamp {**
15   goal_1;
16   goal_2;
17   **};
18 END lamp_goal;
19
20 END mylamp_pkg;
21
22
23 PACKAGE mylamp_lamp_pkg
24 PUBLIC
25
26 -- Rules definition
27 -- May be split in several annexes if needed
28 ANNEX lamp {**
29   goal_1 :- write('list all found component types: ', nl,
30   **};
31
32 -- Rules definition
33 -- May be split in several annexes if needed
34 ANNEX lamp {**
35   goal_2 :- isComponentType(__,X,__,__,), write(X), nl,
36   **};
37
38 END mylamp_lamp_pkg;
39
```

```
Static Analysis: LAMP Lab | Timing Analysis | Safety & Security Analysis | Code Generator | Doc Generator
LAMP console
(c) Ellidiss Technologies, 2023
Last command: LAMP Checker

/*-----*\
LAMP console
(c) Ellidiss Technologies, 2023
Last command: LAMP Checker
\*-----*/

[x] AADL fact base loaded.
[ ] no XML fact base loaded.
[ ] no SysML fact base loaded.
[ ] no FACE fact base loaded.
[ ] no CAPELLA fact base loaded.
[ ] no Simulation fact base loaded.
[ ] no Response Time fact base loaded.
[ ] no Native Prolog fact base loaded.
[ ] no CSV fact base loaded.
[ ] no C fact base loaded.
[x] LAMP rules base loaded.
[x] LAMP queries loaded.


LAMP> execution started.

list all found component types:
lamp_goal
Boolean
Integer
Integer_8
Integer_16
```

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