LAMP:
A new model processing language for AADL

ERTS 2020

Toulouse, 30 Jan 2020

P. Dissaux

Ellidiss Technologies, 24, quai de la douane, 29200 Brest, France

Ellidiss Technologies

www.ellidiss.com

Ellidiss Technologies
www.ellidiss.com

Model Processing 1/2

Model processing goals

- Observe: model exploration, queries and views
- Enrich semantics: add constraints not reflected by the syntax
- Verify static properties: rules checkers
- Perform model transformations:
  - Model to Model transformations:
    - Refinement: transform a descriptive models along the development life-cycle
    - Verification: transform a descriptive model into a verification model
  - Model to Text transformations
    - Code generation
    - Documentation generation

Model processing requirements

- Require a model processing language
- With a way to get access to model éléments (API)
- With strong and well defined semantics
  - Especially to support verification activities for critical systems (qualification)
- Better with an existing set of predefined rules and utilities (libraries)

Ellidiss Technologies
www.ellidiss.com

Model Processing 2/2

Model Processing Languages

- May be specific to the modeling language or development environment:
  - OMG world: OCL, ATL, QVT, MOFM2T
  - AADL world: REAL, LUTE, RESOLUTE, AGREE
- Or a general purpose programming language: Java, Ada, Python, Prolog
- Declarative style is appropriate to express rules

Model Processing Applications

- Off-line processing:
  - Part of a modeling/verification environment (tool plugins)
  - Under responsability of the tool editor
  - Independent from the actual model instance that will be processed
- On-line processing:
  - Part of the model to be processed (model extension)
  - Can be developed/customized by the end user
  - May access actual model instance information
  - Can also be used to prototype an off-line processing tool

Ellidiss Technologies
www.ellidiss.com

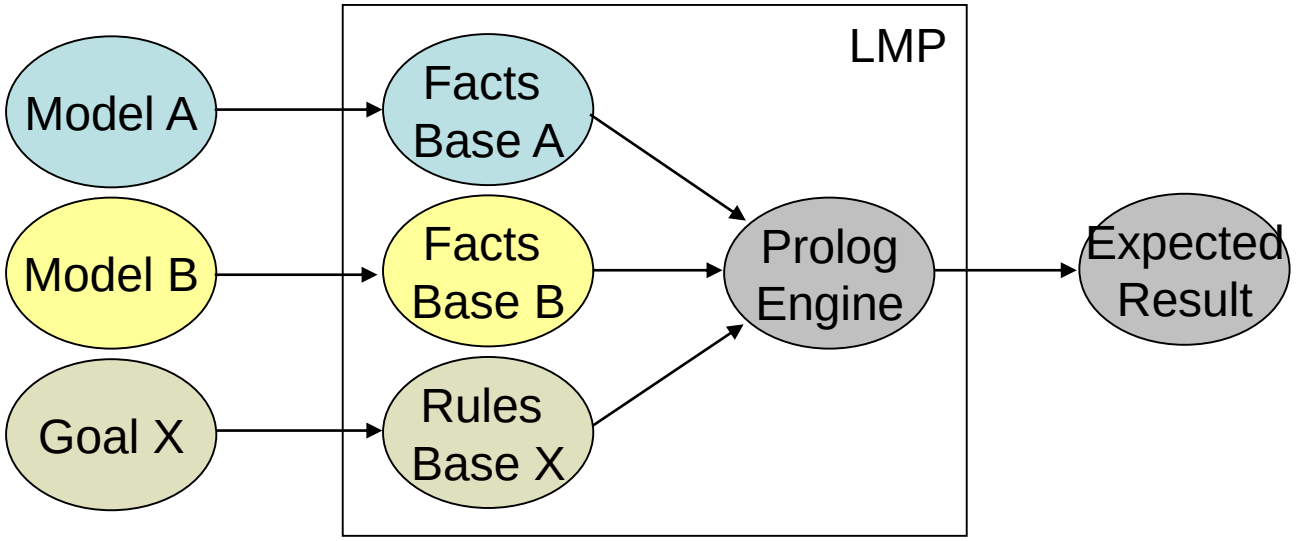
LMP:
a generic model processing approach

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:

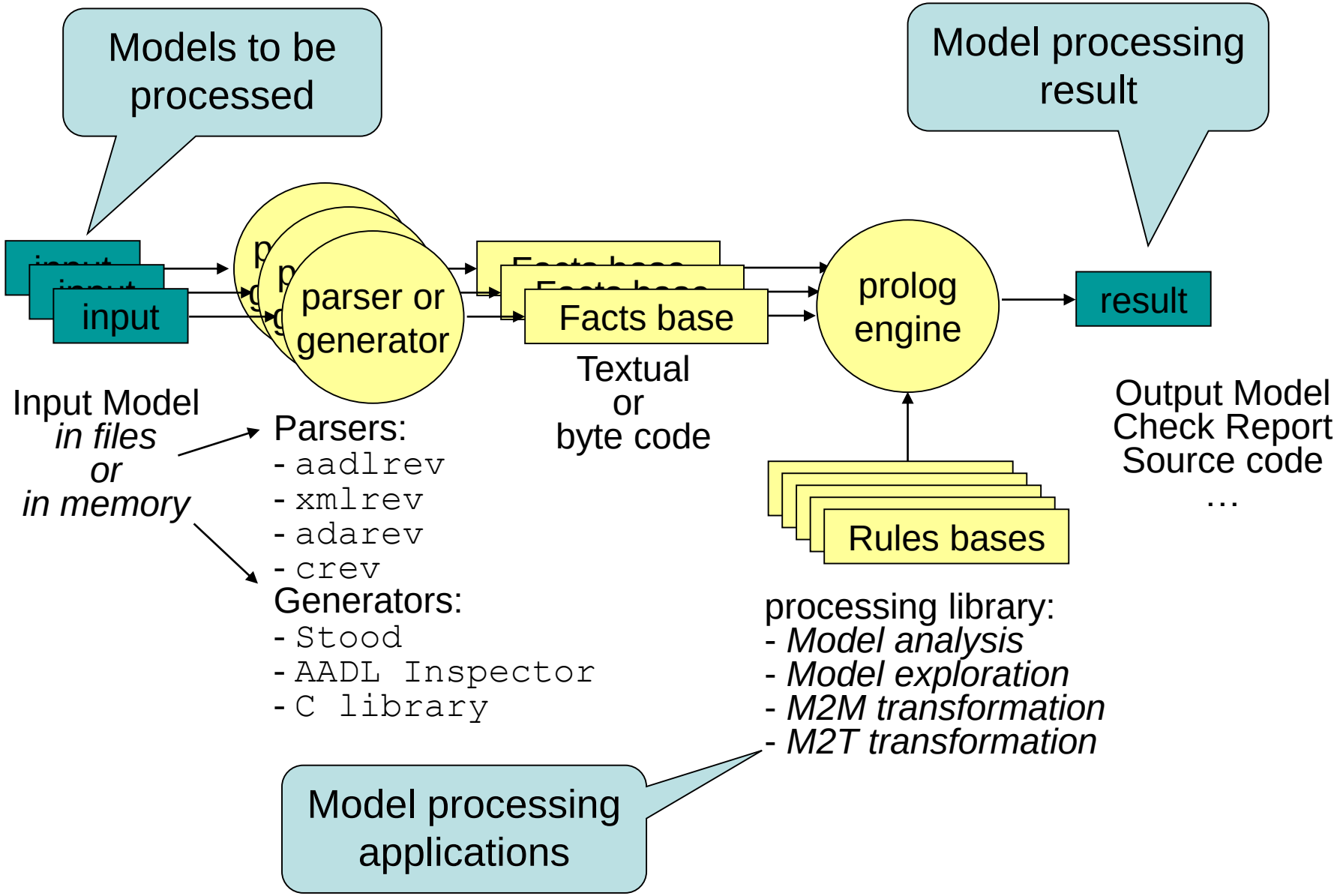
LMP development and runtime process:

- 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
www.ellidiss.com

LMP runtime process



Ellidiss Technologies
www.ellidiss.com

LMP for tag-based models (XMI/XML)

Fragment of the meta-model: XML Schema Definition (XSD)

```
<xsd:schema ... >
  <xsd:complexType name="SPEC-OBJECT">
    ...
    <xsd:attribute name="DESC" type="xsd:string" use="optional"/>
    <xsd:attribute name="IDENTIFIER" type="xsd:ID" use="required"/>
    <xsd:attribute name="LAST-CHANGE" type="xsd:dateTime" use="required"/>
    <xsd:attribute name="LONG-NAME" type="xsd:string" use="optional"/>
  </xsd:complexType>
  ...
</xsd:schema>
```

Corresponding prolog Fact Definition:

```
isSpecObject(Desc,Identifier,LastChange,LongName).
```

Ellidiss Technologies
www.ellidiss.com

LMP for token based models

Fragment of the meta-model: Backus-Naur Form (BNF)

```
…
component_type ::=
        component_category component_identifier
        { property }*
        end component_identifier;
property ::=
        property_name => property_value;
…
```

Corresponding prolog Facts definition:

```
isComponentType(ComponentCategory,ComponentIdentifier).
isProperty(ComponentIdentifier,PropertyName,PropertyValue).
```

Ellidiss Technologies
www.ellidiss.com

LMP: Merging and Processing

1. Merge together the two facts bases:

```
isSpecObject('','','','Temp_Lower_Bound').
isSpecObject('','','','Temp_Upper_Bound').
...
isComponentType('Thread','Thermostat').
isProperty('Thermostat','Coverage','Temp_Lower_Bound').
isProperty('Thermostat','Coverage','Temp_Upper_Bound').
```

model A (requirements)

model B (design)

2. Add the rules base:

check requirements coverage:

```
checkCoverage :-
  isSpecObject(_,_,_,R),
  isComponentType(_,C),
  not(isProperty(C,'Coverage',R)),
  writeErrorMessage(R).
```

rule to check

3. Run the Prolog engine

Ellidiss Technologies
www.ellidiss.com

Benefits of the LMP approach

- **Generic** solution for:
  - Model queries
  - Model constraints
  - Model transformations: M to M or M to T
  - Model exploration and architectural reasoning
- **Standard** prolog language (ISO/IEC 13211-1)
- **Independent**: compatible with the main meta-modelling formats (BNF, XSD, Ecore)
- **Declarative**: rules oriented approach
- **Modular**:
  - separate fact and rules bases
  - rules bases transitivity: e.g. Marte to Cheddar : Marte to AADL and AADL to Cheddar
- **Formal** (boolean logic): appropriate for tool qualification
- **Flexible**:
  - Supports heterogeneous models
  - Supports incomplete models (subsets)
  - Supports erroneous models (debugging)
- **Industrial** return of experience of many off-line processing tools:
  - Airbus: LMP applied for the verification of DO-178 certified projects (A380, A350)
  - European Space Agency: used in the TASTE tool-chain
  - Honeywell: architecture reasoning
  - Ellidiss: AADL Inspector model adaptors and Stood code generators
  - Commercial support

Ellidiss Technologies
www.ellidiss.com

AADL

- **Architecture Analysis and Design Language:**
  - Describes Systems with Hardware and Software components
  - Formal static and run-time semantics (real-time)
  - Textual and graphical notations

- **SAE aerospace division – AS-2C subcommittee:**
  - AADL 1.0 (AS 5506) 2004
  - AADL 2.0 (AS 5506A) 2009
  - AADL 2.1 (AS 5506B) 2012
  - AADL 2.2 (AS 5506C) 2017
  - AADL 3.0 : in preparation

SAE International

- **Annex documents**
  - Annex A: ARINC 653 Interface (AS 5506/1A) 2015
  - Annex B: Data Modelling (AS 5506/2) 2011
  - Annex C: Code Generation Annex (AS 5506/1A) 2015
  - Annex D: Behavior Annex (AS 5506/3) 2017
  - Annex E: Error Model Annex v2 (AS 5506/1A) 2015
- **Online resources**
  - https://www.sae.org/standards/content/as5506c
  - http://www.openaadl.org

Ellidiss Technologies
www.ellidiss.com

Off-line LMP plugins in AADL Inspector

LMP plugin	category
AADL semantic rules	model checker
AADL instance builder	model exploration
AADL ARINC 653 rules	model checker
UML MARTE to AADL	model transformation
SysML to AADL	model transformation
Capella to AADL	model transformation
AADL to Cheddar	model transformation
AADL to Marzhin	model transformation
AADL to OpenPSA	model transformation
AADL printer	model unparser
LAMP checker	model checker

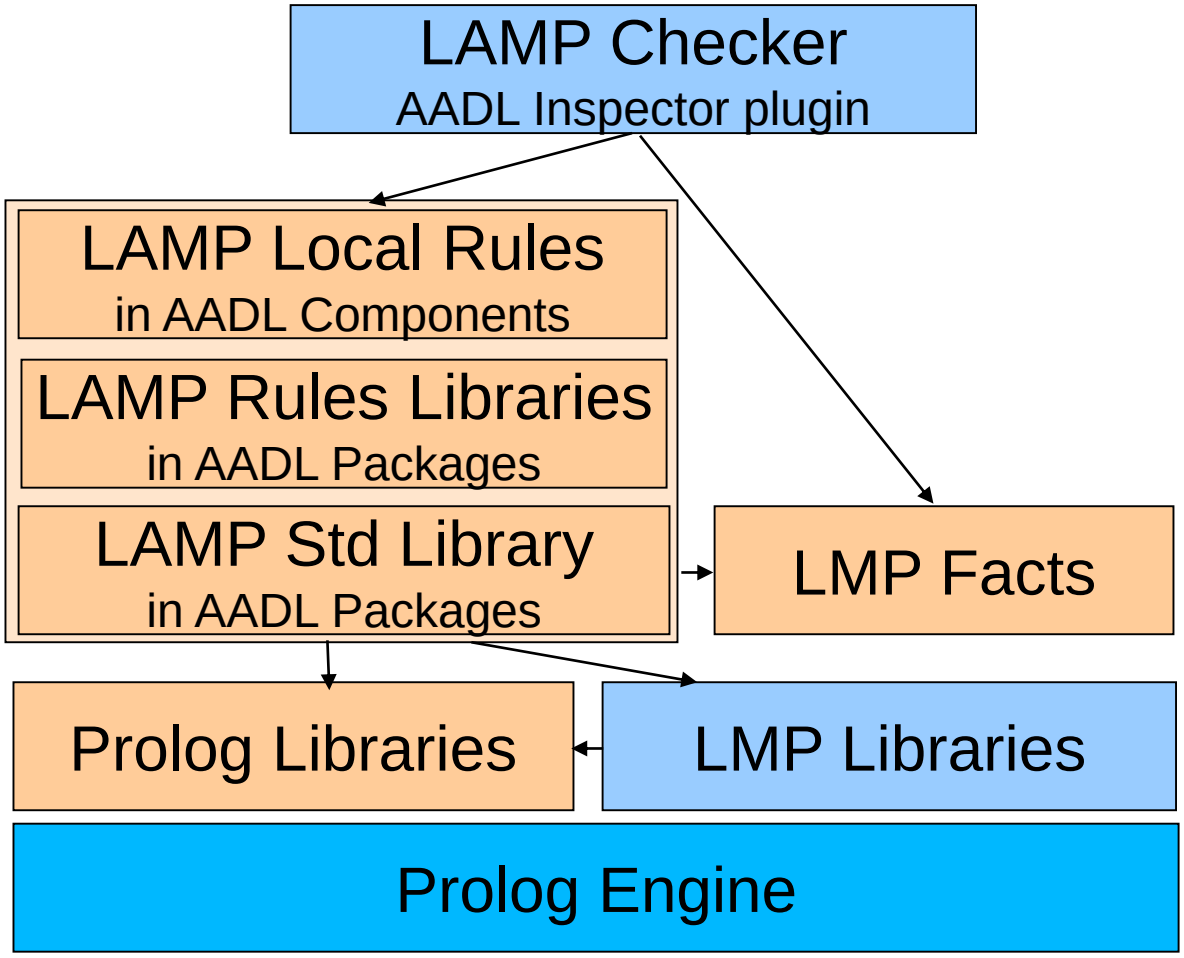
Ellidiss Technologies
www.ellidiss.com

On-line LMP programs for AADL: the LAMP annex

```
package Ellidiss::ERTS2020::paper26::e1
public

abstract A
-- a LAMP annex at component level
    annex LAMP {**
    /* standard prolog syntax */
    **};
end A;

-- a LAMP annex at package level
annex LAMP {**
/* standard prolog syntax */
**};

end Ellidiss::ERTS2020::paper26::e1;
```

Ellidiss Technologies
www.ellidiss.com

The LAMP stack



prolog byte code
prolog source code
uses

Ellidiss Technologies
www.ellidiss.com

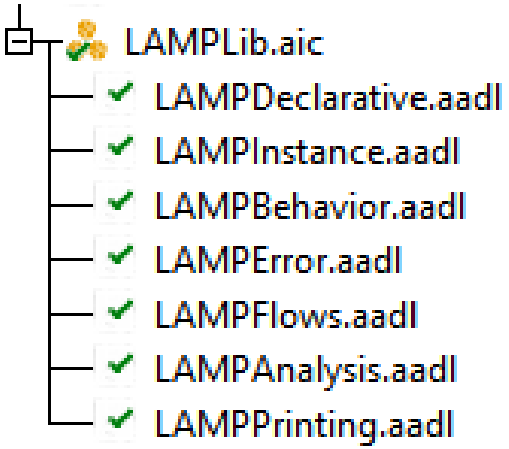
Inside LAMP

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 (all AADL model elements)
- Can also work on incorrect 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, …)



LAMP support inside AADL Inspector

- LAMP checker analysis plugin
- LAMPLib is pre-loaded within the AADL "environment"
- Available since A.I. 1.7 (http://www.ellidiss.com/downloads)

Ellidiss Technologies
www.ellidiss.com

LAMP Std Lib (1/3)

AADL core language accessors

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

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

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

```
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;
```

Ellidiss Technologies
www.ellidiss.com

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

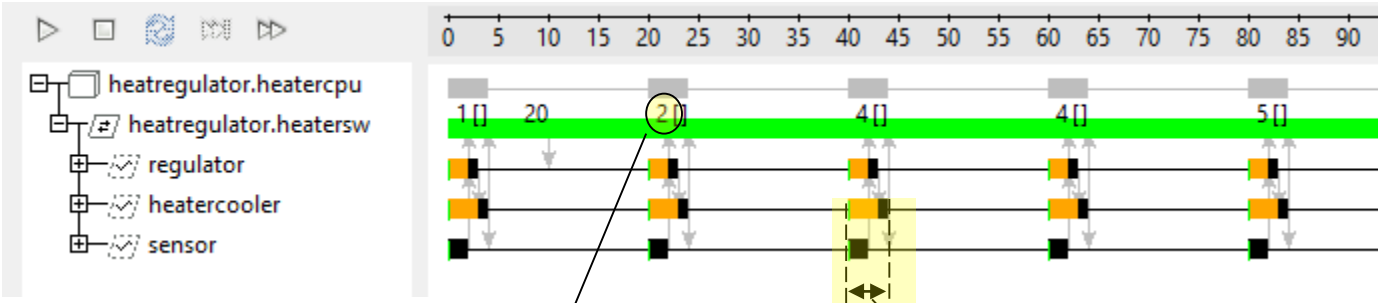
```
package Ellidiss::LAMP::EMV2 public
annex LAMP {**
  getErrorTypes(Class,Name,Ancestor) :- …
  getErrorStates(Class,Name,TypeSet,Kind) :- …
  … **};
end Ellidiss::LAMP::EMV2;
```

returns all the known error types from within a classifier

returns all the error states of a classifier

LAMP Std Lib (3/3)

Other analysis tools feedback



returns the computed response time of a thread

returns the computed response time of a thread

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

Ellidiss Technologies
www.ellidiss.com

LAMP Example (1/6)

```
PACKAGE p PUBLIC
SYSTEM IMPLEMENTATION s.i
SUBCOMPONENTS
  a : PROCESS a.i;
  c : PROCESSOR c;
PROPERTIES
  ACTUAL_PROCESSOR_BINDING => (REFERENCE(c)) APPLIES TO a;
  SCHEDULING_PROTOCOL => (RMS) APPLIES TO c;
ANNEX LAMP {**
  write('hello!'), nl,
  getInstances(X,'THREAD',C),
  checkPeriod(X,C),
  checkWCET(X,C),
  checkDeadline(X,C),
  checkOverflow(X,C)
**};
END s.i;
…
ANNEX LAMP {**
  checkPeriod(Id,Class) :- …
  checkWCET(Id,Class) :- …
  checkDeadline(Id,Class) :- …
  checkOverflow(Id,Class) :- …
**};
END p;
```

rules to be checked in component (goal)

local library rule in package (detailed in next slides)

Ellidiss Technologies
www.ellidiss.com

LAMP Example (2/6)

```
THREAD t
FEATURES
  i : IN DATA PORT d;
  o : OUT DATA PORT d;
PROPERTIES
  DISPATCH_PROTOCOL => Periodic;
  PERIOD => 15ms;
  DEADLINE => 8ms;
  COMPUTE_EXECUTION_TIME => 2ms..2ms;
ANNEX Behavior_Specification {**
  VARIABLES
    v : d;
  STATES
    s : INITIAL COMPLETE FINAL STATE;
  TRANSITIONS
    t : s -[ON DISPATCH]-> s { rand!(v); computation(10ms); o := v };
**};
END t;
```

AADL properties consistency

```
ANNEX LAMP {**
  checkPeriod(Id,Class) :-
    getProperties(Id,Class,'DISPATCH_PROTOCOL','PERIODIC'),
    not(getProperties(Id,Class,'PERIOD',_)),
    write('periodic thread '), write(Id),
    write(' has no period'), nl,
    fail.
  checkPeriod(_,_).
**}
```

LAMP Example (3/6)

```
THREAD t
FEATURES
  i : IN DATA PORT d;
  o : OUT DATA PORT d;
PROPERTIES
  DISPATCH_PROTOCOL => Periodic;
  PERIOD => 15ms;
  DEADLINE => 8ms;
  COMPUTE_EXECUTION_TIME => 2ms..2ms;
ANNEX Behavior_Specification {**
  VARIABLES v : d;
  STATES s : INITIAL COMPLETE FINAL STATE;
  TRANSITIONS t : s -[ON DISPATCH]-> s { rand!(v); computation(10ms); o := v };
**};
END t;
```

consistency between core property and sublanguage

```
ANNEX LAMP {**
  checkWCET(Id,Class) :-
    getProperties(Id,Class,'COMPUTE_EXECUTION_TIME',W),
    getMinMax(W,_,M),
    getBAComputation(Class,V),
    timeToInt(M,A), timeToInt(V,B), A \= B,
    write('WCET inconsistency for thread '),
    write(Id), nl,
    write('( '), write(A), write(' != '), write(B), write(' )'), nl,
    fail.
  checkWCET(_,_).
**}
```

Ellidiss Technologies
www.ellidiss.com

LAMP Example (4/6)

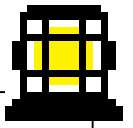
```
THREAD t
FEATURES
  i : IN DATA PORT d;
  o : OUT DATA PORT d;
PROPERTIES
  DISPATCH_PROTOCOL => Periodic;
  PERIOD => 15ms;
  DEADLINE => 8ms;
  COMPUTE_EXECUTION_TIME => 2ms..2ms;
ANNEX Behavior_Specification {**
  VARIABLES v : d;
  STATES s : INITIAL COMPLETE FINAL STATE;
  TRANSITIONS t : s -[ON DISPATCH]-> s { rand!(v); computation(10ms); o := v };
**};
END t;
```

consistency between property and simulation results

```
ANNEX LAMP {**
  checkDeadline(Id,Class) :-
    getProperties(Id,Class,'DEADLINE',D),
    timeToInt(D,A), strToNum(A,B),
    getResponseTime(Id,R),
    strToNum(R,C), C > B,
    write('deadline missed for thread '),
    write(Id), nl,
    write('( '), write(C), write(' > '), write(B), write(' )'), nl,
    fail.
  checkDeadline(Id,Class).
**}
```

Ellidiss Technologies
www.ellidiss.com

LAMP Example (5/6)



```
THREAD t
FEATURES
  i : IN DATA PORT d;
  o : OUT DATA PORT d;
PROPERTIES
  DISPATCH_PROTOCOL => Periodic;
  PERIOD => 15ms;
  DEADLINE => 8ms;
  COMPUTE_EXECUTION_TIME => 2ms..2ms;
  LAMPExample2_Prop::MAX_VALUE => 80 APPLIES TO o;
ANNEX Behavior_Specification {**
  VARIABLES v : d;
  STATES s : INITIAL COMPLETE FINAL STATE;
  TRANSITIONS t : s -[ON DISPATCH]-> s { rand!(v); computation(10ms); o := v };
**};
END t;
```

overflow detection from simulation traces

```
ANNEX LAMP {**
  checkOverflow(Id,Class) :-
    concat(Id,'.o',F),
    getProperties(F,Class,'LAMPExample2_Prop::MAX_VALUE',M),
    getPortValue(F,T,V),
    strToNum(V,W), strToNum(M,N), W > N, ,
    write('overflow of out data port '),
    write(F), sp, write(' at tick '), write(T), nl,
    write('( '), write(W), write(' > '), write(80), write(' )'), nl,
    fail.
  checkOverflow(Id,Class).
**}
```

Ellidiss Technologies
www.ellidiss.com

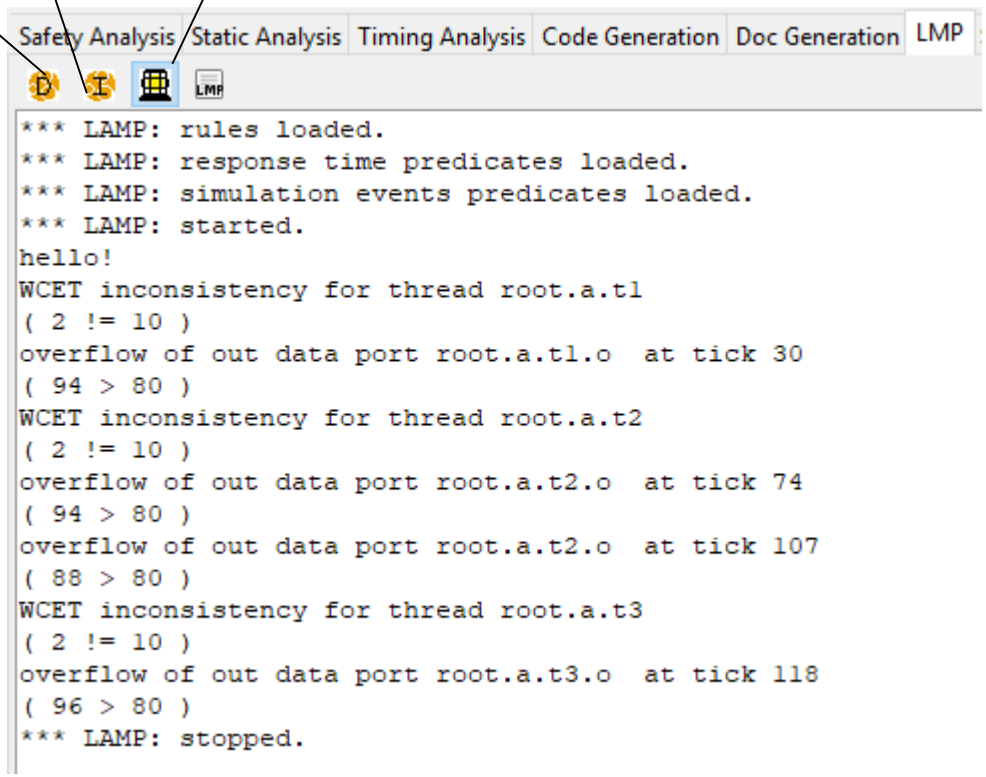
LAMP Example (6/6)

view the AADL instance model facts

view the AADL declarative model facts

launch the LAMP checker

```
PROCESS a
FEATURES
  i : IN DATA PORT d;
  o : OUT DATA PORT d;
END a;

PROCESS IMPLEMENTATION a.i
SUBCOMPONENTS
  t1 : THREAD t;
  t2 : THREAD t;
  t3 : THREAD t;
CONNECTIONS
  x : PORT i -> t1.i;
  y : PORT t1.o -> o;
END a.i;
```



Ellidiss Technologies
www.ellidiss.com

LAMP: Logical AADL Model Processing

- Use standard prolog language a processing language for AADL
  - No new dedicated language to define & learn &maintain
  - Declarative syntax and formal semantics (ISO/IEC 13211)
- Fully integrated within AADL declarative models (LAMP Annex)
- Enhanced by processing libraries
  - User defined in local AADL package
  - User defined in other AADL packages of the project
  - Standard LAMPLib
  - Prolog byte code can be used for IP libraries
- Complete access to model entities of AADL core and other annexes
- Can merge several input data sets (requirements, analysis results, …)
- In-line processing: can work on the actual model instance
- Appropriate to specify customized assurance cases

- Available since AADL Inspector 1.7 (LAMP Checker)