LAMP:
Logical AADL Model Processing

Pierre Dissaux, AADL committee, Pittsburgh, 15 May 2018

Ellidiss Technologies
www.ellidiss.fr

Introducing LAMP

LAMP = AADL + LMP

- AADL: I assume you know what it is…
- LMP: Logical Model Processing
  - The model processing technology used by Ellidiss for developing Stood and AADL Inspector plugins (offline model processing).
  - Based on standard prolog language
- LAMP: Logical AADL Model Processing
  - LMP based AADL annex (inline model processing)
  - AADL annex definition + libraries + run-time plugin (in AADL Inspector)

LAMP Objectives:

- Provide an inline, comprehensive, rigorous, flexible and user friendly solution to explore and analyse AADL models.
- Provide lighting to perform deeper analysis with AADL Inspector.

Note: in some way, may be seen as an alternate solution to REAL, LUTE, AGREE, RESOLUTE, the AADL Constraint Annex, …

Ellidiss Technologies
www.ellidiss.fr

LMP
main principles

1. Input data (model elements) are translated into prolog facts:

```
PACKAGE P PUBLIC THREAD t FEATURES i : IN DATA PORT d; END t; END p;
```

```
isComponentType('p','PUBLIC','t','THREAD','NIL',58).
isFeature('PORT','p','t','i','IN','DATA','d','NIL','NIL',60).
isComponentTypeEnd('p','PUBLIC','t','t',88).
isPackageEnd('p','p',146).
```

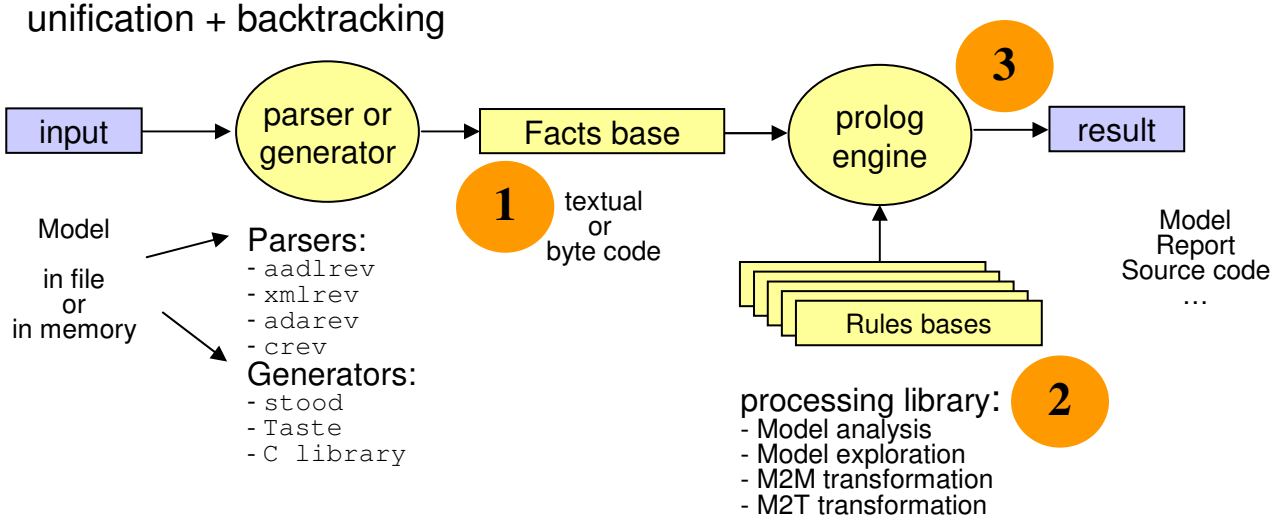
2. Processing rules and libraries are defined with standard prolog syntax:

```
logical AND: comma
logical OR: semicolon
logical NOT: not() operator
```

```
isComponentType(_,_,T,'THREAD',_,_), write(T).
```

3. Execution follows standard prolog semantics:

unification + backtracking



Ellidiss Technologies
www.ellidiss.fr

what's more with LAMP ?

The 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

The 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, …)
- …

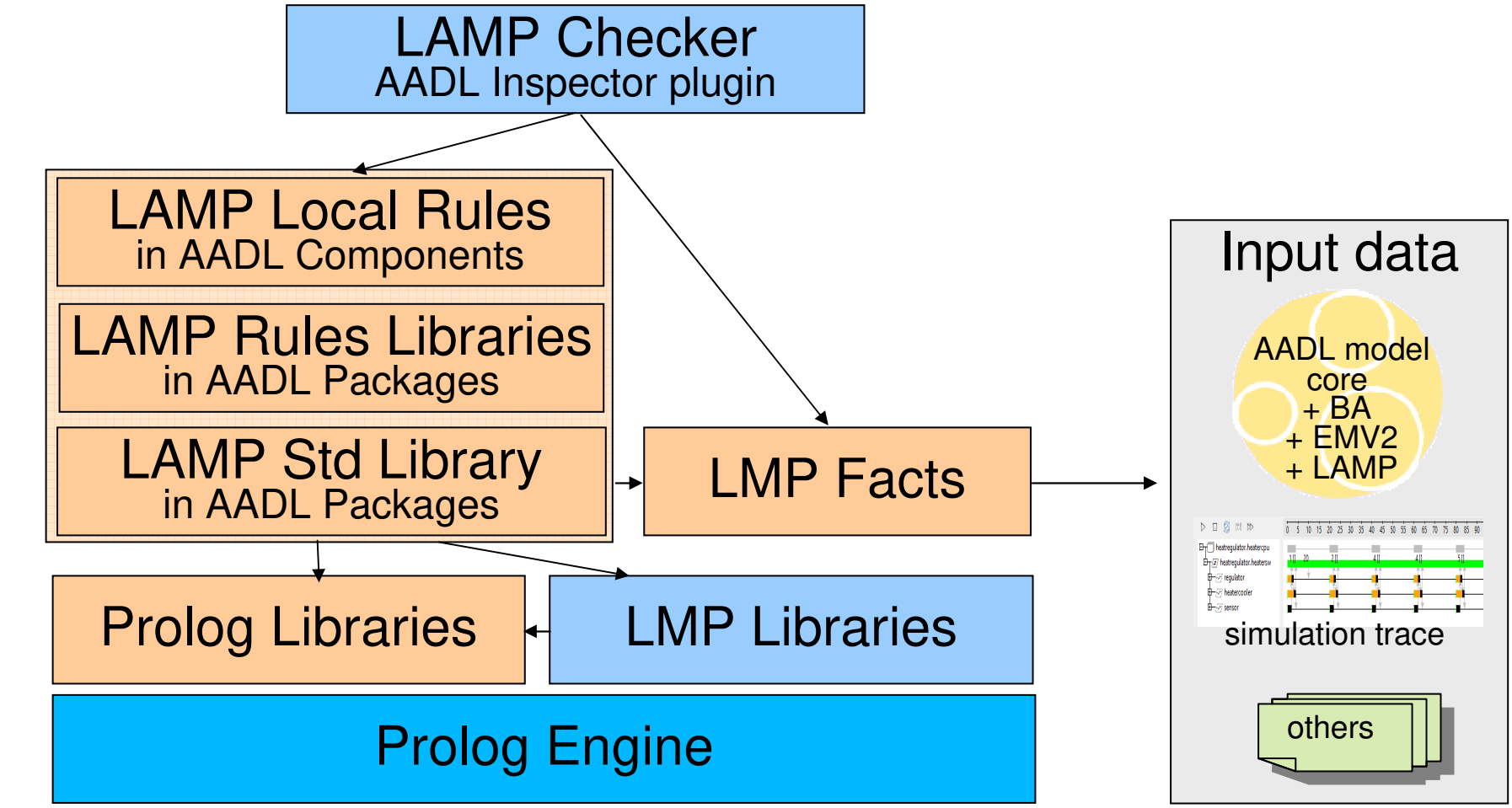
The LAMP plugin in AADL Inspector

- New LAMP Checker service in the existing LMP plugin
- Available in A.I. 1.7 (release planned end Q2 2018).

Note: the low level API (LMP) remains directly available

Ellidiss Technologies
www.ellidiss.fr

The LAMP stack



Ellidiss Technologies
www.ellidiss.fr

LAMP Std Lib (1/3)

1. 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;
```

LAMP Std Lib (2/3)

2. 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

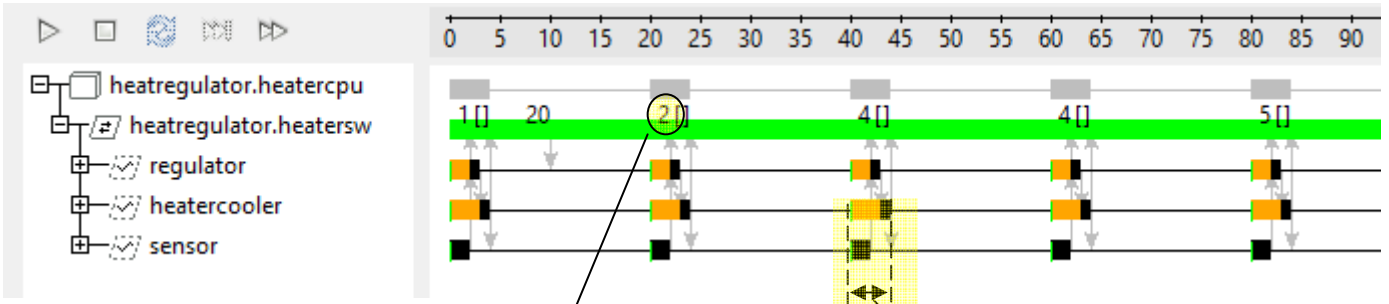
```
package Ellidiss::LAMP::EMV2 public
annex LAMP {**
  getErrorTypes(Class,Name,Ancestor) :- …
  getErrorStates(Class,Name,TypeSet,Kind) :- …
  … **};
end Ellidiss::LAMP::EMV2;
```

returns all the known error types from within a classifier

returns all the error states of a classifier

LAMP Std Lib (3/3)

3. 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;
```

Note: analysis tools must produce a prolog result facts base

Ellidiss Technologies
www.ellidiss.fr

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

local library rule in package (detailed in next slides)

Note: the LAMPLib package must be part of the project

Ellidiss Technologies
www.ellidiss.fr

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;
```

consistency between properties

```
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(_,_).
**}
```

Ellidiss Technologies
www.ellidiss.fr

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;
```

```
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(_,_).
**}
```

consistency between core property and BA action

Ellidiss Technologies
www.ellidiss.fr

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;
```

```
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).
**}
```

consistency between property and simulated response time

Ellidiss Technologies
www.ellidiss.fr

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;
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;
```

```
ANNEX LAMP {**
  checkOverflow(Id,Class) :-
    concat(Id,'.o',F),
    getPortValue(F,T,V),
    strToNum(V,W), W > 80,
    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).
**}
```

consistency between property and simulated port values

Ellidiss Technologies
www.ellidiss.fr

LAMP Example (6/6)

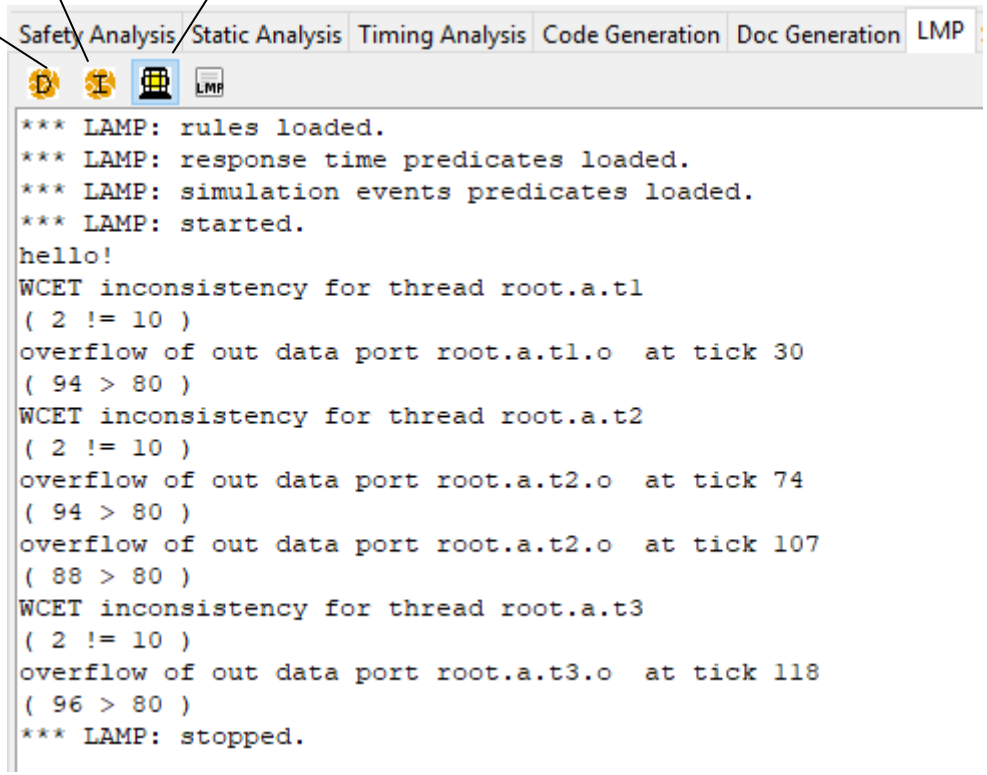
view the AADL instance model facts

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.fr

Conclusion

LAMP Summary:

- Standard prolog language:
  - No dedicated language to define & learn &maintain
  - Declarative syntax and formal semantics (ISO/IEC 13211)
- Enhanced by LMP libraries
  - Defined in the local package
  - Defined in other packages of the project
  - LAMPLib
  - prolog byte code can be used for IP libraries
- Fully integrated within AADL models (LAMP Annex)
- Full access to AADL core and other annexes entities
- Can be merged with other input data sets (requirements, analysis results, …)
- Available in AADL Inspector 1.7 (LAMP Checker)

Next steps:

- Enrich the LAMP Std Lib (LMP API can be used meanwhile)
- Enrich the facts base with result of other analysis plugins
- Experiment Safety and Security assurance cases
- …