ZIH
Zentrum für Informationsdienste und Hochleistungsrechnen

TECHNISCHE UNIVERSITÄT DRESDEN

Model checking of architectural descriptions: software & hardware

Chadlia JERAD

Dr.-Ing. in Electrical Engineering

Associate Professor at National School of Computer Sciences, Tunisia
Coordinator of the “Embedded software and systems” Specialization
Researcher at OASIS Laboratory/National Engineering School of Tunis, Tunisia

University of La Manouba

- 14 institutions
- National School of Computer Science (ENSI)
  - Graduate engineering programs

    3 Departments for 6 different specializations
  - Coordinator of the “Embedded Software and Systems” specialization
  - Research and doctoral studies

    PhD (1)

    University Habilitation (1)
- 845 students (*)
- 82 full-time professors (*)

ENSI

(*) non official numbers

University of Tunis El Manar

- 15 institutions
- National Engineering School of Tunis (ENIT)
  - Graduate engineering programs

    5 Departments for 9 different programs
  - Research and doctoral studies

    MS (6) + Professional MS (1)

    PhD (8)

    University Habilitations (7)
  - 1444 students
  - 211 full-time professors

OASIS Laboratory @ ENIT

- Staff
  - 50 members
  - 24 doctoral candidates
- Themes
  - Optimization/Supply chain
  - Computer science
- National collaborations
  - Tunisian laboratories
  - Ministry of health/Hospitals
  - Ministry of transportation
- International collaborations
  - France (ECP, ENSTA…)
  - Canada (Université Laval…)
  - USA (University of Minnesota)

- Diagnosability conditions of a class of DES
- Design of an ASIP for speaker recognition applications
- Accelerator design of stereovision applications

Motivations

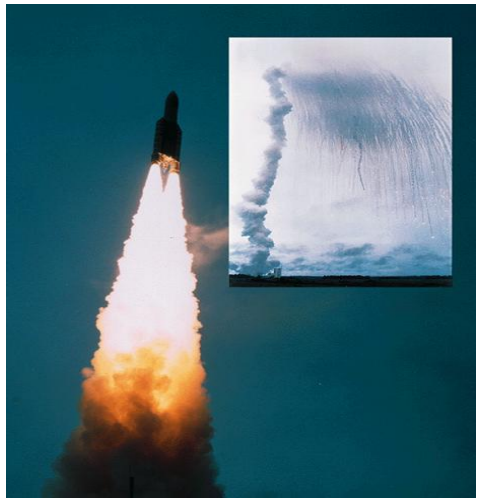
Increasing complexity: [1]

- SW requirements: 2x/10months [LOC SW/Chip]
- SW productivity: 2x/5years [LOC SW/Day]
- HW productivity: 1,6x/16months [Gates/Day]

Costs of failure

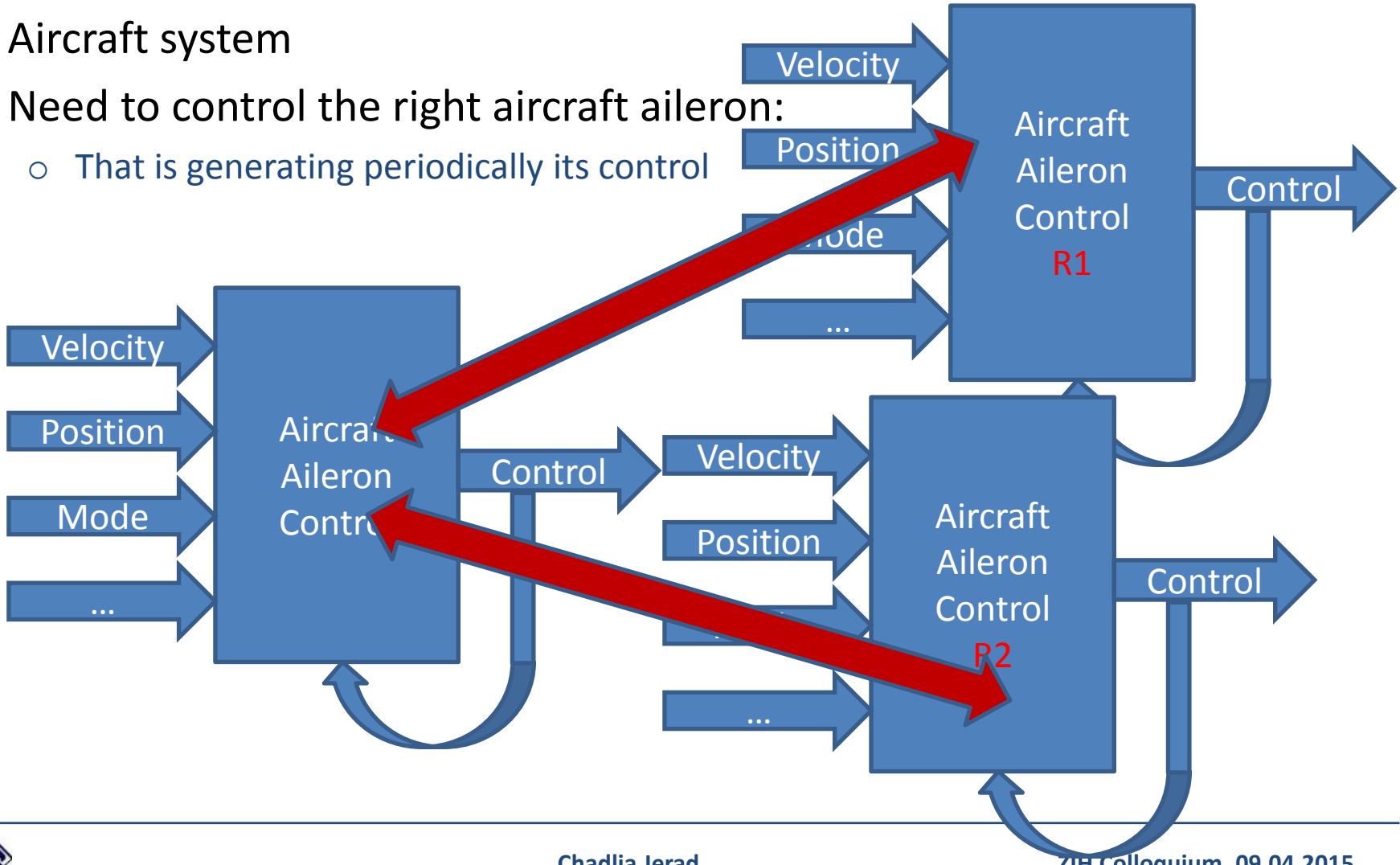
- Very depending on the system criticality
- Costs of failure are sometimes … dramatic!
  - Need to detect design faults
  - ~ 70% of project development cycle: design verification
  - Every approach to reduce development time

    → considerable influence on economy



Let's consider a simplified example

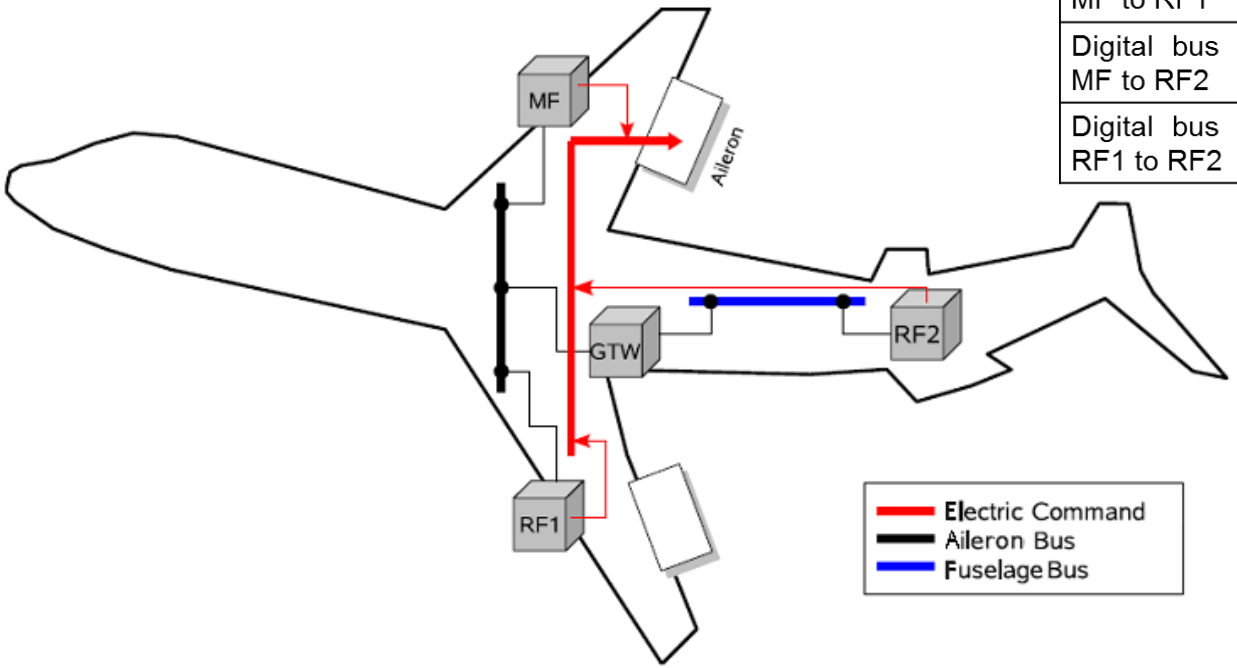
- Aircraft system
- Need to control the right aircraft aileron:
  - That is generating periodically its control



Need for abstraction

- Abstract the control computation details
  - Coarse grained vision
- Reason about the architecture

Bus	Minimal Latency	Maximal latency
Analogical buses connecting functions to the aileron	0 ms	0 ms
Digital bus connecting MF to RF1	0 ms	2 ms
Digital bus connecting MF to RF2	0 ms	4 ms
Digital bus connecting RF1 to RF2	0 ms	4 ms



Aerodynamics laws opinion

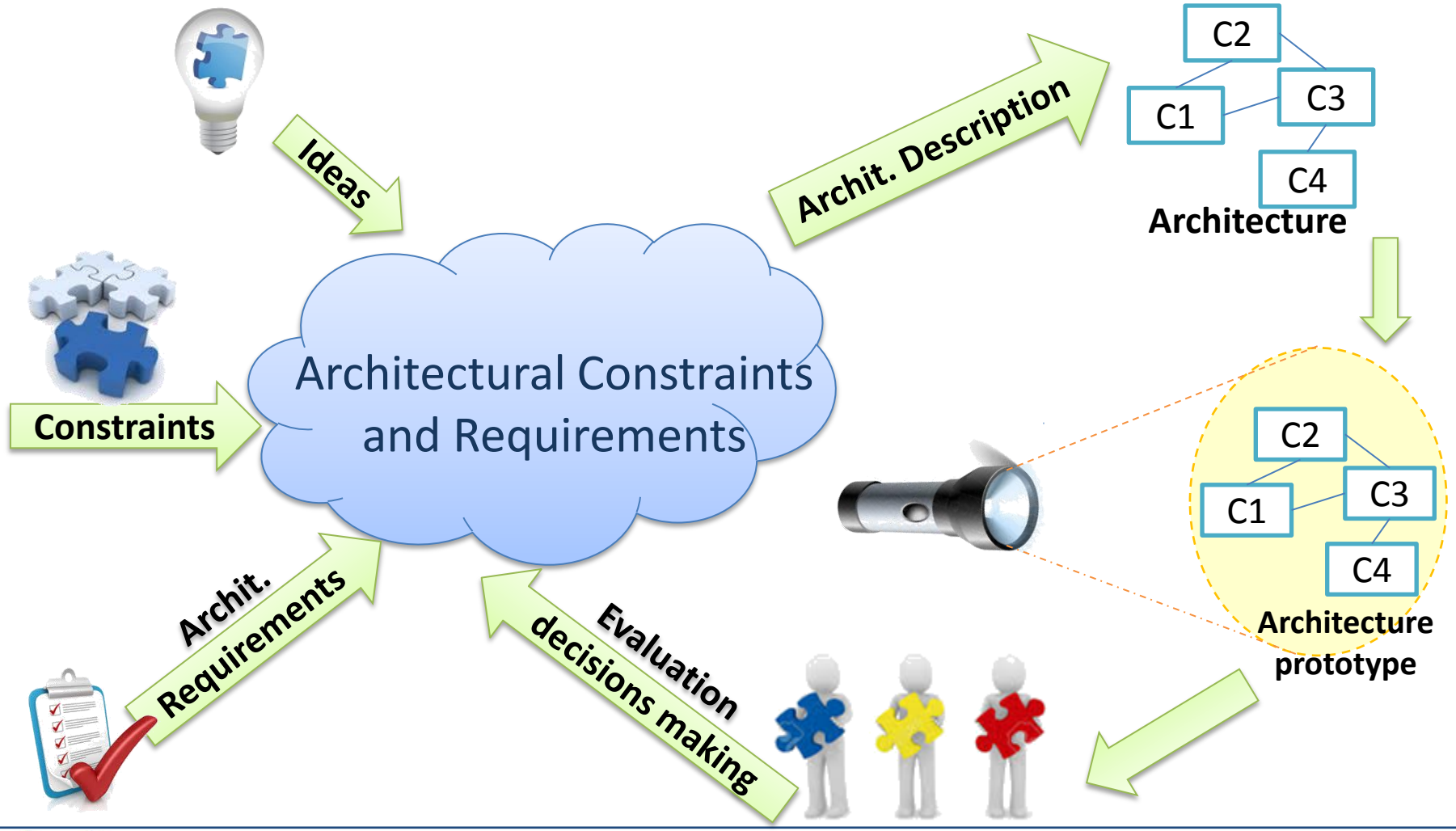
- Aircraft stability
- The aileron should not be non commended for more than 16ms

➔ **Need for verification, or even more formal verification**

Architecture description

- Very first created model of a system during the lifecycle
- Coarse grained vision
  - A complete description of a real system is impossible
- A set of architectural design decisions allowing to generate the artifact of an architecture
  - Abstract implementation details
  - Extract the reusable components
    - Why? Time-to-market
    - How? Modeling

General workflow



Need of a specification language

- Why do we need a specification language?
  - Need for accurate semantics
  - Ability to reason about:
    - Consistency,
    - Completeness,
    - Ambiguity,
    - Minimalism,
    - …
- Architecture description languages (ADL)

Architecture Description Languages (ADL) categories

- Semi-formal or formal Notation
- Textual and/or graphical syntax
- With/without design/validation tool support
- General purpose or domain specific
- Research prototypes or included in industrial process
- Modeling target:
  - Software architecture
    - Rapide, Wright
  - Hardware architecture
    - VHDL/Verilog | ADL for processor design: nML, LISA…
  - System architecture (hardware + software)
    - AADL

Software architecture description language

- SA milestones
  - 1993: SA is recognized as an independent research domain
  - 1997: first SADL prototypes
  - 2003: Architecture description languages
- Many attempts to classify software architecture description languages
- Medvidovic and Taylor's classification and comparison framework: the most complete

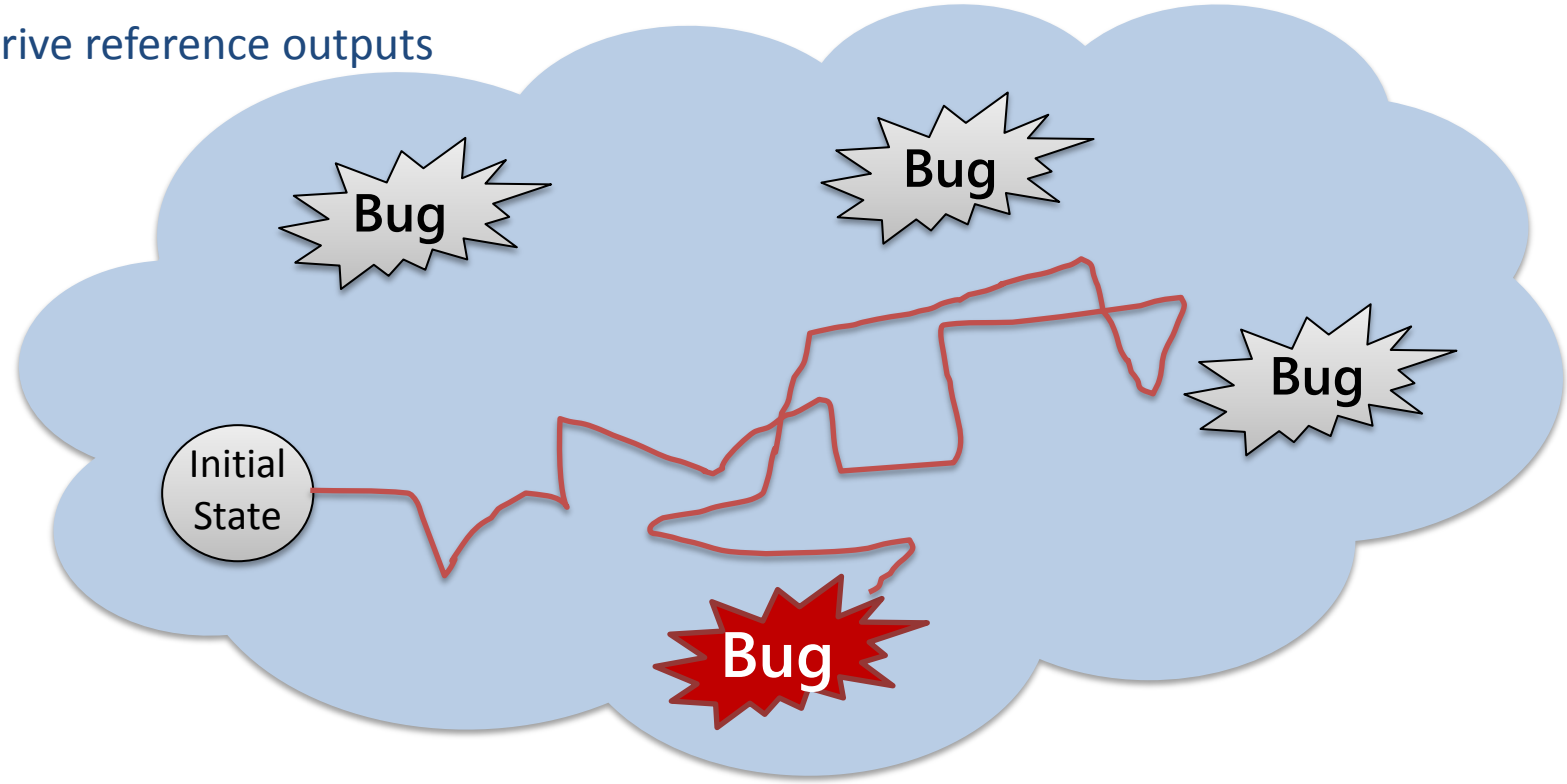
```
ADL
  Architecture modelling features
    Components
      Interface, Semantics, Constraints
      Types, Evolution,
      Non-functional properties
    Connectors
      Interface, Semantics, Constraints
      Types, Evolution,
      Non-functional properties
    Architectural configurations
      Understandability,
      Compositionality
      Heterogeneity,
      Refinement and traceability
      Scalability, Evolution,
      Dynamism, Constraints,
      Non-functional properties
  Tool support
    Active specification, Multiple views
    Analysis, Refinement
    Implementation generation, Dynamism
```

How to find bugs?

- At the early design stages
  - Simulation
  - Formal verification

Simulation

- Checks one output point at a time
- Input-driven
  - generate input vectors
  - derive reference outputs



Formal verification

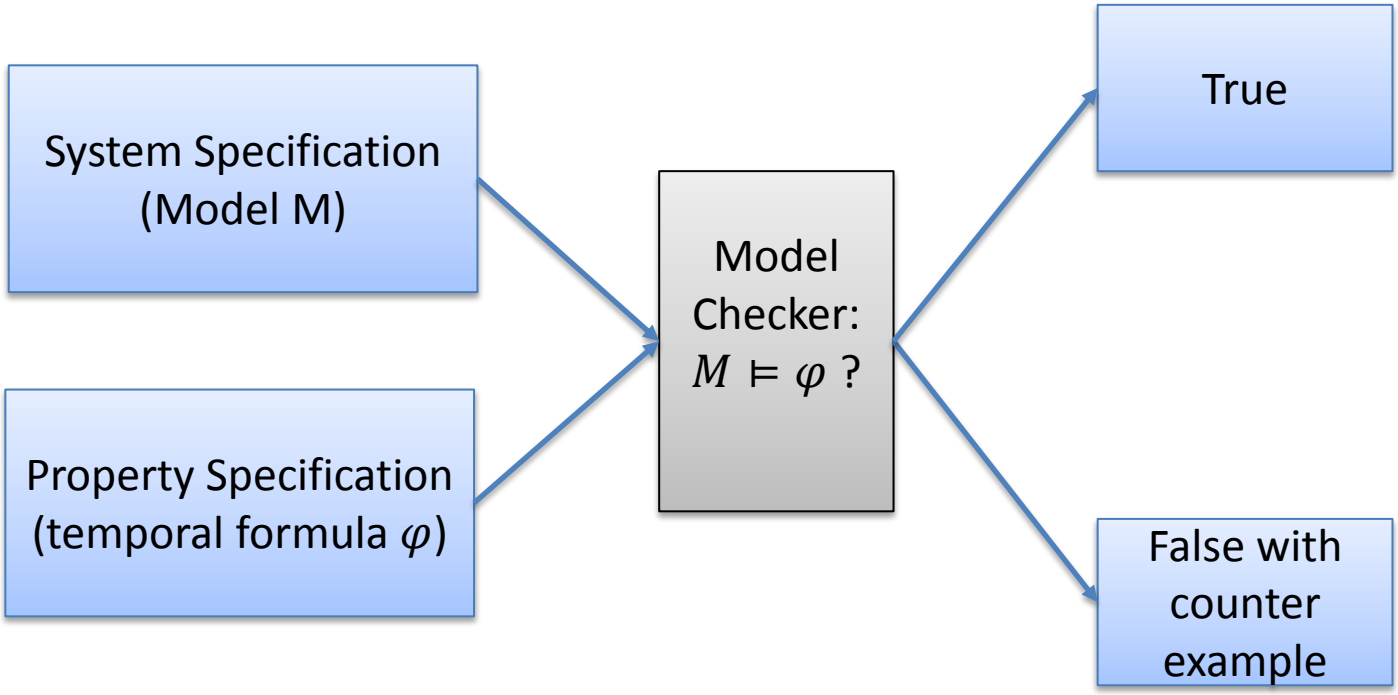
- Checks a group of output points at time
- Output-driven:
  - describe desirable output behavior
  - the formal checker approves or disapproves
- Uses extensive memory and long run time

Theorem proving	• Proves implementation is equivalent to specification in some formalism
Equivalence checking	• Compares synthesized model against original model
Model checking	• Checks if a model satisfies a given property

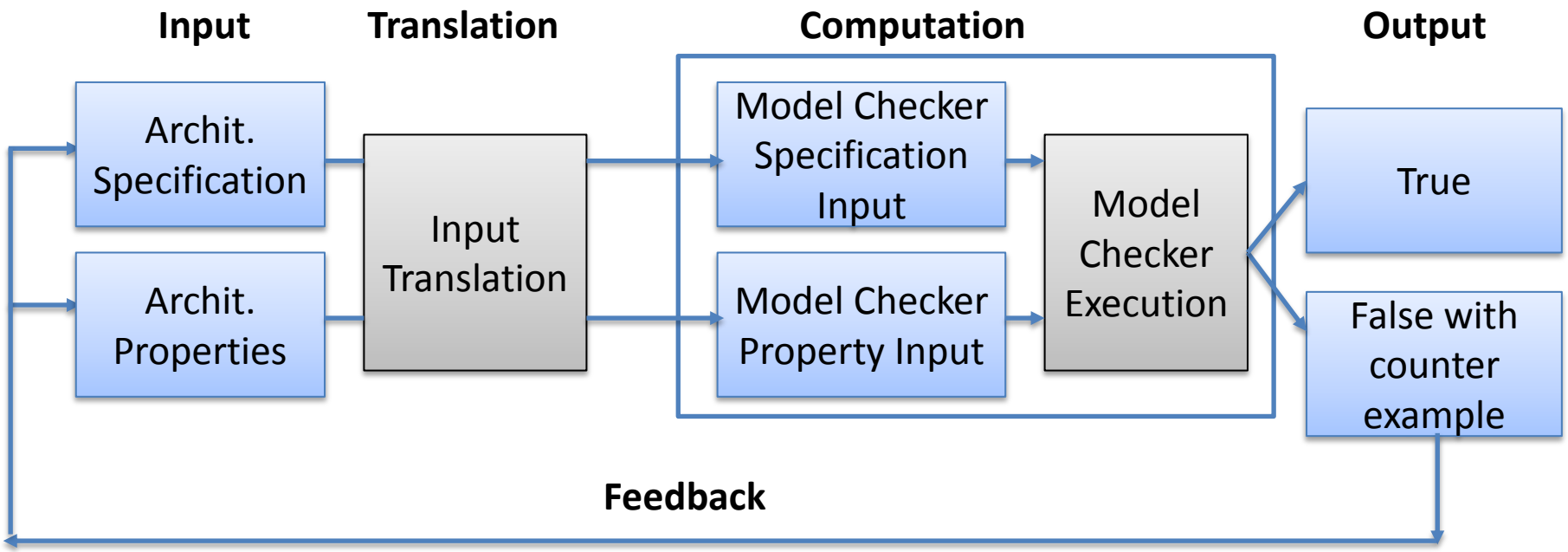
What is model checking

- Model Checking (Property Checking):
  - Automatic technique for verifying finite systems
  - Pioneered by Edmund Clarke, in 1981
- Exhaustive state space search
- Can uncover subtle design errors
- Currently the dominant formal verification tool.
- Major challenge:
  - To fight state-explosion problem
- Success stories:
  - RTL model debug prior to synthesis
  - Used concurrently with and/or prior to simulation
  - Cadence's Model Checking tool: FormalCheck

Model checking process



Model checking architectural designs process



Classification

MC Engine Name	Used by	Input Languages	Supported Logics	Types of Property	Specific Features	Type of Checking
SPIN	Bose, Arcade, Zanolin et al. CHARMY	PROMELA (CSP-like)	LTL or Buchi automaton	Safety and liveness	Distributed Systems	Model Checking
SMV	SAM	SMV languages	CTL	Safety, liveness and fairness	Hardware and embedded system	Model Checking
NuSMV	Aemilia, AutoFocus		CTL or LTL		Customizable and extensible	Model Checking
Cadence SMV	Garlan et al., AutoFocus		CTL, LTL or SMV		Refinement verification	Model Checking and Equivalence Checking
Maude	Cbabel, Lfp	Maude	LTL	Safety and liveness	Based on multiset rewriting system	Model Checking
UPPAAL	Fujaba	Timed-automata	TCTL	Safety and timed liveness	Real-time	Model Checking
…						

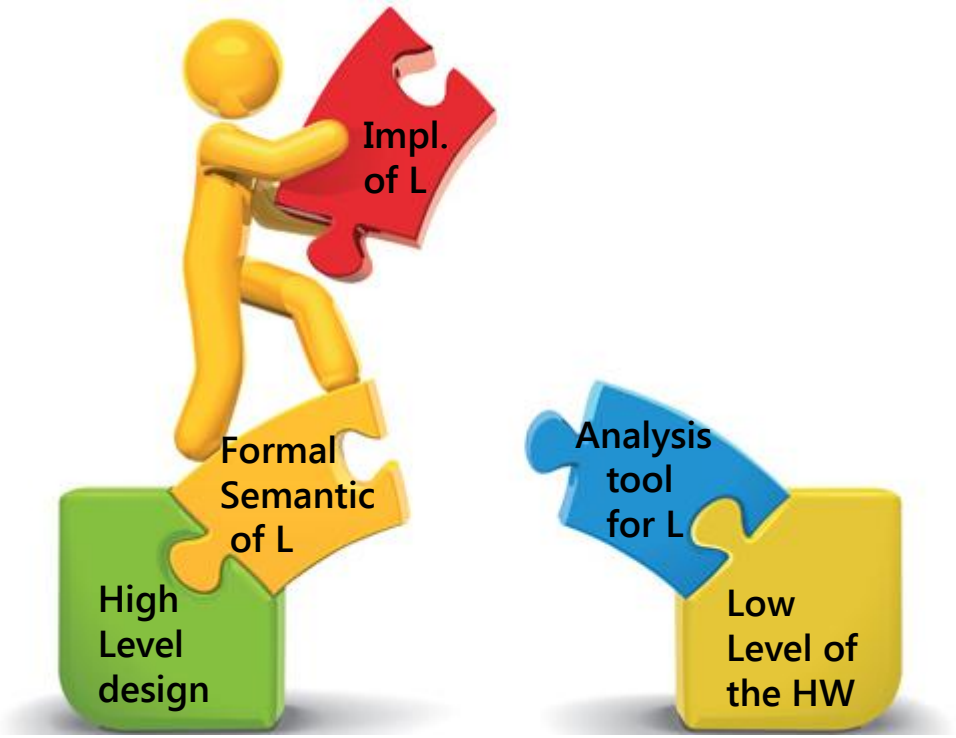
Rewriting logic formalism for architecture description and analysis

Why “Rewriting logic”?

- Semantic framework for concurrency formalisms
  - Executable semantics
- It can be used to give both, operational and denotational semantics to programming languages
- The logic of concurrent action and change
- Based on simple deduction rules
- A logical framework in which other logics can be represented

Rewriting logic semantics project

- "The broad goal of the project is to develop a tool-supported computational logic framework for modular programming language design, semantics, formal analysis and implementation, based on rewriting logic."



Definition

- A labelled rewrite specification is a 4-uplet $\mathcal{R} = (\Sigma, E, \mathrm{L}, R)$, where:
  - Σ is a ranked alphabet of function symbols,
  - E is a set of Σ-equations,
  - L is a set of labels
  - and R is a set of elements ; $R \subseteq L \times (T_{\Sigma,E}(X))^2$
- The rewrite signature of $\mathcal{R}$ is the equational theory (Σ, E)
- The elements in R are called rewrite rules and are often denoted by expressions of the form $r: [t] \rightarrow [t']$

$$State \leftrightarrow Term \leftrightarrow Proposition$$

$$Transition \leftrightarrow Rewriting \leftrightarrow Deduction$$

$$Distibruted\ Structure \leftrightarrow Algebraic\ Structure \leftrightarrow Propositional\ Structure$$

Rewriting rules

- Given a rewrite specification $\mathcal{R}$, $\mathcal{R} \vdash [t] \rightarrow [t']$ iff $[t] \rightarrow [t']$ can be obtained by finite application of the following deduction rules:
  - Reflexivity: $\forall\, [t] \in T_{\Sigma,E}(X), \frac{}{[t] \rightarrow [t]}$
  - Congruence: $\forall\, f \in \Sigma_n, n \in \mathbb{N}, \frac{[t_1] \rightarrow [t'_1] \ldots [t_n] \rightarrow [t'_n]}{[f(t_1,\ldots,t_n)] \rightarrow [f(t'_1,\ldots,t'_n)]}$
  - Replacement: $\forall\, r : [t(x_1, \ldots x_n)] \rightarrow, [t'(x_1, \ldots x_n)] \in R$
    $\frac{[\omega_1] \rightarrow [\omega'_1] \ldots [\omega_n] \rightarrow [\omega'_n]}{[t(\bar{\omega}/\bar{x})] \rightarrow [t'(\overline{\omega'}/\overline{x'})]}$
  - Transitivity: $\frac{[t_1] \rightarrow [t_2] [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$

Applications area

Models of concurrent computation	Distributed architectures and components	Reference Model for Open Distributed Processing	Operational semantics of languages
• Equational programming • Lambda calculi • Petri nets • CCS and p-calculus • Actors	• UML diagrams and metamodels • Middleware architecture for composable services	• Validation of OCL properties • Model management and model transformations	• Structural operational semantics (SOS) • Agent languages • Active networks languages • Mobile Maude • Hardware description languages

Specification and analysis of communication protocols	Modeling and analysis of security protocols	Real-time, biological, probabilistic systems	Logical framework and meta-tool
• Active networks • Wireless sensor networks • FireWire leader election protocol	• Cryptographic protocol specication language CAPSL • MSR security specication formalism • Maude-NPA	• Real-Time Maude Tool • Pathway Logic • Pmaude	• Linear logic • Translations between HOL and Nuprl theorem provers • Pure type systems • Open calculus of constructions • Tile logic

Maude system

- High-level language and high-performance system
- Developed a SRI
- Features
  - Executability
  - High performance engine
  - Modularity and parameterization
  - Built in - booleans, number hierarchy, strings
  - Reflection - using descent and ascent functions
  - Search and model-checking

http://maude.cs.uiuc.edu

Maude system

- Modules
  - Functional modules: equational logic
  - System modules: specify general rewrite theories
- Maude provides a range of efficient analysis commands:
  - Rewriting for simulation/prototyping
  - Search
  - …
- Temporal logic model checking: check whether all possible behaviors from one initial state satisfies a temporal logic formula
  - only when reachable state space finite

Maude's meta-level & meta-programming

- Rewriting logic is reflective
- The functional module META-LEVEL:
  - Maude terms are redefined
  - Maude modules are redefined
  - Moving between reflection levels: operations: upModule, upTerm, downTerm…
- Reducing a term to canonical form: metaReduce
- Rewriting a term in a system module: metaRewrite and metaFrewrite
- Maude versions:
  - Core Maude
  - Full Maude

Full-Maude

- Full Maude is an extension of Maude
- Written in Maude itself
- Special syntax for object-oriented modules supporting object-oriented
  - Concepts such as objects, messages, classes, and multiple class inheritance.
  - Class declarations:

```
class Person | age : Nat , status : Status .
```

  - An object can be represented as a term

```
< "Peter" : Person | age : 35 , status : single >
```

- Full Maude itself can be used as a basis for further extensions, by adding new functionality
  - Declarative debuggers for Maude, for wrong and missing answers
  - Real-Time Maude tool for specifying and analyzing real-time systems

Real-Time Maude

- Particularly suitable to specify object oriented real-time systems
- Two types of rewrite rules:
  - ordinary rewrite rules
  - and tick rewrite rules,
- Real-Time rewrite theories
  - Timed modules
  - Or object oriented timed modules
- Analysis techniques
  - timed rewriting
  - untimed and time-bound search for states that are reachable from the initial state
  - time-bound linear temporal logic model checking

Architecture description in Real-Time Maude

- Main idea:
  - Topology → Static aspect
    - Components
    - Connectors
    - Configuration
    - The interfaces
  - Behaviour → Dynamic aspect

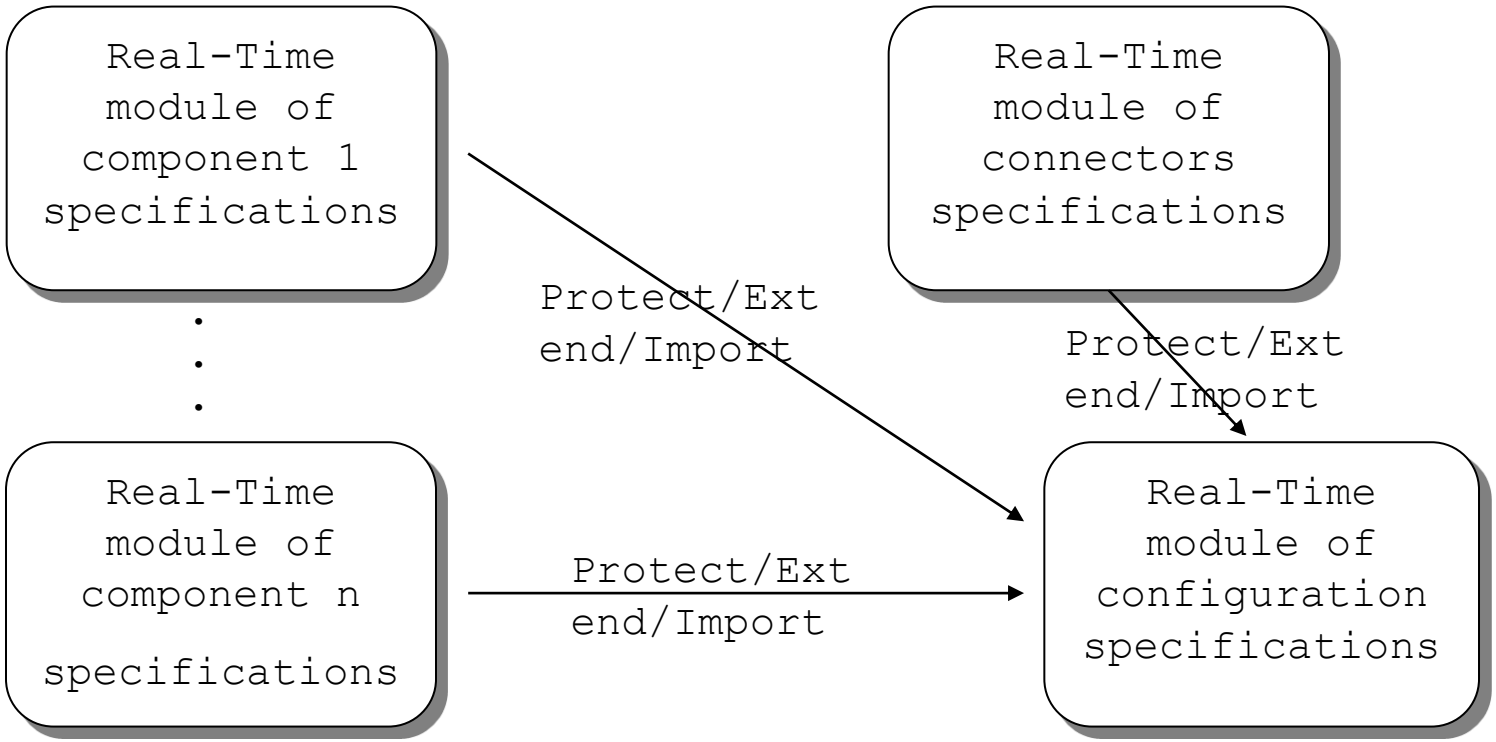
Architecture description in Real-Time Maude

- Main idea:

Software architecture concepts	Real-Time Maude concepts
Component	Class
Component interface	A set of terms having the sort Service on top
Component computation	A set of rewrite rules
Connector	A set of rewrite rules
Types	Sorts
Communication events	Messages exchange
Configuration	A term having the sort System on top
Compositionality	Sub-class relationship

Architecture description in Real-Time Maude (3)

- Modular description



Model checking with Maude

- Two levels of specification:
  - a system specification level,
  - a property specification level
- Temporal logic allows specification of properties such as
  - safety properties
  - and liveness properties
- Maude 2 includes a model checker to prove properties expressed in Linear temporal logic (LTL)

Linear Temporal Logic

- Temporal operators:
  - Eventually: $\Diamond\varphi = \top\,\mathcal{U}\,\varphi$
  - Henceforth: $\Box\varphi = \neg\Diamond\neg\varphi$
  - Release: $\varphi\,\mathcal{R}\,\psi = \neg((\neg\varphi)\,\mathcal{U}\,(\neg\psi))$
  - Unless: $\varphi\,\mathcal{W}\,\psi = (\varphi\,\mathcal{U}\,\psi) \vee (\Box\varphi)$
  - Leads-to: $\varphi \rightsquigarrow \psi = \Box(\varphi \rightarrow (\Diamond\psi))$
  - Strong implication: $\varphi \Rightarrow \psi = \Box(\varphi \rightarrow \psi)$
  - Strong equivalence: $\varphi \Leftrightarrow \psi = \Box(\varphi \leftrightarrow \psi)$
- Timed LTL
  - Time-bounded linear temporal logic model checking
  - The untimed linear temporal logic model checking

Modelling components

- Example: MF class

```
class MF | clock : Time, dly : Time,
                   computation : String, mode : String .

op CmdService RF1Service RF2Service
            : -> Service [ctor] .
```

Master function behaviour

MF behaviour	Real-Time Maude specification

Compute position	`rl [MF-Compute-position] :`
	`  < O : MF | mode : "cmd", computation : "Start" >`
	`=> < O : MF | mode : "cmd", computation : "CmpPos" > .`

Sending controls	`rl [MF-Sending] :`
	`  < O : MF | computation : "CmpPos" >`
	`=> < O : MF | computation : "Send" >`
	`  provide("CmdService") provide("RF1Service")`
	`  provide("RF2Service") .`

Period	`rl [MF-Period] :`
	`  < O : MF | clock : R, dly : 0, computation : "Send" >`
	`=> if R == 2 then`
	`  < O : MF | clock : 0, dly : 0, computation : "Start" >`
	`  else`
	`  < O : MF | clock : R, dly : step, computation : "Send" >`
	`  fi .`

MF failiure	`rl [MF-Failiure] :`
	`  < O : MF | >`
	`=> none .`

RF1 behaviour

RF1 behaviour	Real-Time Maude specification

Receive a message	`crl [RF1-Receive-require-message] :`
	`  < O : RF1 | clock : R, dly : R' > require("RF1Service")`
	` => < O : RF1 | clock : 0, dly : 0 >`
	`if R <= 4 .`

Advance time if no message is received	`crl [RF1-No-message-advance-time] :`
	`  {< O : RF1 | clock : R, dly : 0 > Conf}`
	` => {< O : RF1 | clock : R, dly : step > Conf}`
	`if R <= 4 and not(existe(require("RF1Service"), Conf)) .`

Move to cmd mode	`crl [RF1-Move-to-cmd-mode] :`
	`  < O : RF1 | clock : R, dly : R', mode : "off", computation : "Start" >`
	` => < O : RF1 | clock : 0, dly : 0, mode : "cmd", computation : "Cmd" >`
	`if R > 4 .`

Compute position	`rl [RF1-Compute-position] :`
	`  < O : RF1 | mode : "cmd", computation : "Cmd" >`
	` => < O : RF1 | mode : "cmd", computation : "CmpPos" > .`

Sending controls	`rl [RF1-Sending] :`
	`  < O : RF1 | computation : "CmpPos" >`
	` => < O : RF1 | computation : "Send" > provide("CmdService")`
	`  provide("RF1Service") provide("RF2Service") .`

Period	`rl [RF1-Period] :`
	`  < O : RF1 | clock : R, dly : 0, computation : "Send" >`
	` => if R == 2 then < O : RF1 | clock : 0, dly : 0, computation : "Cmd" >`
	`  else < O : RF1 | clock : R, dly : step, computation : "Send" > fi .`

RF1 failiure	`rl [RF1-Failiure] : < O : RF1 | mode : "Cmd" > => none .`

Modelling connections

Connectors behaviour	Real-Time Maude specification

Analogical bus	`rl [Analogical-bus] :`
	`  provide("CmdService")`
	`  => require("CmdService") .`

Digital bus with maximal latency 2	`rl [Digital-bus-2] :`
	`  provide("RF1Service")`
	`  => tempRequire("RF1Service", 2, 0, 0) .`
	`crl [tempRequire-to-require] :`
	`  tempRequire("RF1Service", R, R', R'')`
	`  => require("RF1Service")`
	`if R'' >= R' and R'' <= R .`

Digital bus with maximal latency 4	`rl [Digital-bus-4] :`
	`  provide("RF2Service")`
	`  => tempRequire("RF2Service", 4, 0, 0) .`
	`crl [tempRequire-to-require] :`
	`  tempRequire("RF2Service", R, R', R'')`
	`  => require("RF2Service")`
	`if R'' >= R' and R'' <= R .`

Advancing time

- A synchronous rule that increases all clock attribute values:

```
crl [tick] :
 {C:Configuration} => {delta(C:Configuration, R)} in time R
if mte(C:Configuration) == true .
```

- `mte` : operation that describes advancing time condition
- `delta` : operation that models the effect of time elapse on the system

System verification

- The system's initial state:

```
ops AileronInst MFInst RF1Inst RF2Inst : -> Oid [ctor] .

op initState : -> System [ctor] .

eq initState = { < AileronInst : Aileron | clock : 0, dly : 0,
        computation : "Start" >

        < MFInst : MF | clock : 0, dly : 0,  computation : "Start",
        mode : "Cmd" >

        < RF1Inst : RF1 | clock : 0, dly : 0, computation : "Start",
        mode : "Off" >

        < RF2Inst : RF2 | clock : 0, dly : 0, computation : "Start",
        mode : "Off" >  } .
```

System verification

- First property:
  - The aileron should not remain without control more than 16 ms (AileronCMD property).

```
op AileronCMD : -> Prop [ctor] .

eq {< O : Aileron | clock : R', dly : R'',
       computation : S >
    Rest:Configuration} |= AileronCMD = (R' > 16) .
```

```
Maude > (mc {initState} |=u [] ~ AileronCMD .)
   rewrites: 51475 in 116ms cpu (114ms real) (443723
   rewrites/second)
   Untimed model check {initState}  |=u []~ AileronCMD in
   AIRCRAFT-CHECK      with mode default time increase 1
   Result Bool :
     true
```

System verification

- Second property
  - There must be only one function controlling the aileron at a time (`NbrFCmd` property).

```
op NbrFCmd : -> Prop [ctor] .
eq {< MFInst : MF | mode : "Cmd" > < RF1Inst : RF1 |
        mode : " Cmd" >
        Rest:Configuration} |= AileronCMD = true .
...
```

```
Maude > (mc {initState} |=u [] ~ NbrFCmd .)
   rewrites: 54617 in 96ms cpu (96ms real) (568891
   rewrites/second)
   Untimed model check {initState}  |=u []~ NbrFCmd in AIRCRAFT-
   CHECK        with mode default time increase 1
   Result Bool :
     true
```

System verification

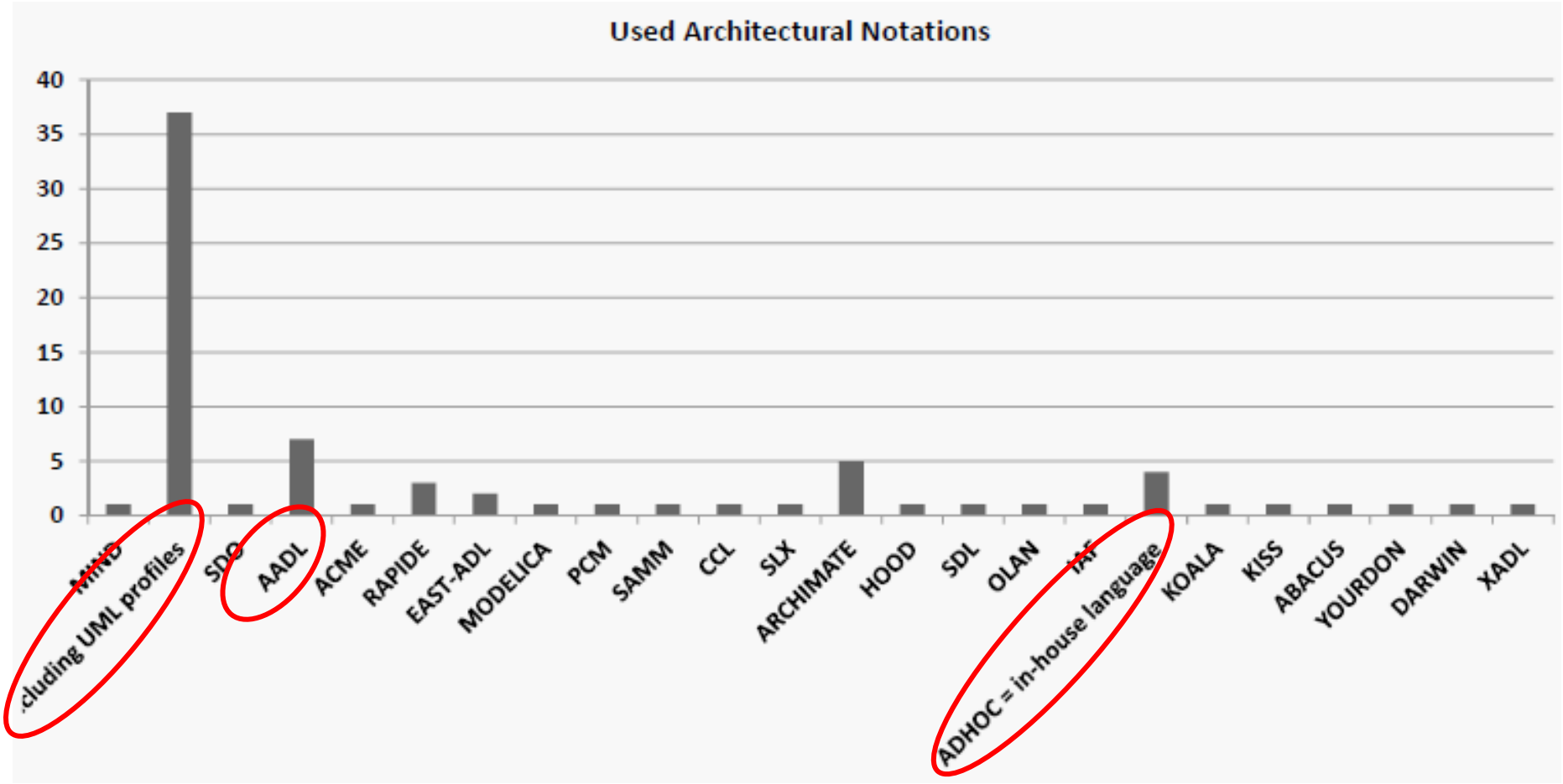
- Deadlock freeness:

```
Maude > (utsearch {initState} =>! GS:GlobalSystem .)
  rewrites: 49226 in 88ms cpu (88ms real) (559348 rewrites/second)
  Untimed search in AIRCRAFT-CHECK {initState} =>! GS:GlobalSystem
              with mode default time increase 1 :
  No solution
```

Current situation

- Why is there so many ADL?
  - Express different needs
  - Different domains
  - Different analysis
  - Some of them are mostly similar
  - Some are only research prototypes

Current situation



Problems to solve

- From ADL side
  - High degree of formalization
  - → Difficult to integrate within industral life-cycle
  - Limited number of analysis tools
  - Limited industrial support

- From formal methods side
  - State explosion problem

Thank you for your attention

References

- Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.
- Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. 2013. What Industry Needs from Architectural Languages: A Survey. *IEEE Trans. Softw. Eng.* 39, 6 (June 2013), 869-891.
- Jernimo Castrilln Mazo and Rainer Leupers. 2013. *Programming Heterogeneous Mpsocs: Tool Flows to Close the Software Productivity Gap*. Springer Publishing Company, Incorporated.
- José Meseguer and Grigore Roşu. 2013. The rewriting logic semantics project: A progress report.*Inf. Comput.* 231 (October 2013), 38-69.
- Nenad Medvidovic and Richard N. Taylor. 2000. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Softw. Eng.* 26, 1 (January 2000), 70-93.
- Pengcheng Zhang, Henry Muccini, and Bixin Li. 2010. A classification and comparison of model checking software architecture techniques. *J. Syst. Softw.* 83, 5 (May 2010), 723-744.
- Peter Csaba Ölveczky. Real-Time Maude 2.3 Manual. Department of Informatics, University of Oslo, (August 8, 2007)