Simulation and Verification of Timed and Hybrid Systems

Bert van Beek and Koos Rooda

Systems Engineering Group
Eindhoven University of Technology

ISC 2007 Delft
11 June 2007

Outline

1. Introduction: Systems Engineering Group and applications
2. Model based engineering
3. Languages for dynamical systems analysis
4. The Chi language
5. Conclusions

Systems Engineering Group: Objectives

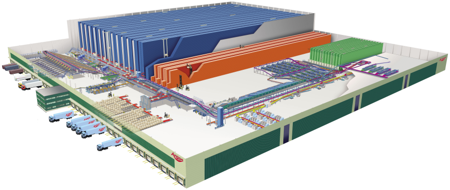
The Systems Engineering group aims to use quantitative methods for the analysis, design and implementation of (embedded) systems exhibiting concurrent behavior.

The objectives are to develop theory and techniques, and to build computational tools, inspired by mechanical engineering science, computer science and mathematics, and to apply these in selected cases from industry.

Application domains

Networks:

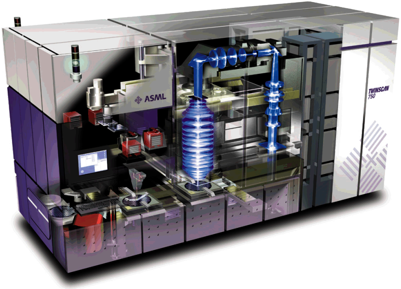
- semiconductor plants
- automotive plants
- transport systems
- container terminals



Application domains

Machines:

- semiconductor industry
    - front end: lithographic systems
    - back end: chip mounting
- medical systems
- printing / paper handling machines



Trend

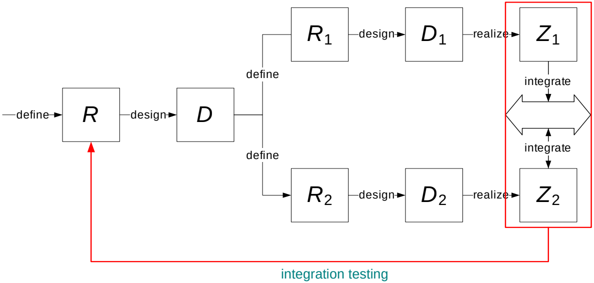
The complexity of a high tech embedded system increases considerably with each new generation.

This leads to

- increasing time to market (T $\uparrow$),
- decreasing quality (Q $\downarrow$), and
- increasing cost and manpower (C $\uparrow$).

How can time to market be decreased (T $\downarrow$), quality be increased (Q $\uparrow$), without increasing cost and man power (C $=$)?

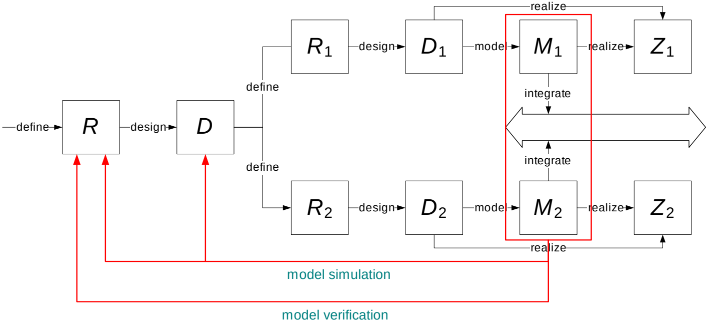
Current (embedded) system development



When integrating different interacting subsystems, usually:

- Requirements ($R_{..}$) are incomplete and ambiguous
- Designs ($D_{..}$) are incomplete and ambiguous
- Testing is possible only when realizations ($Z_{..}$) are ready
- Correcting errors in realizations is costly and time-consuming

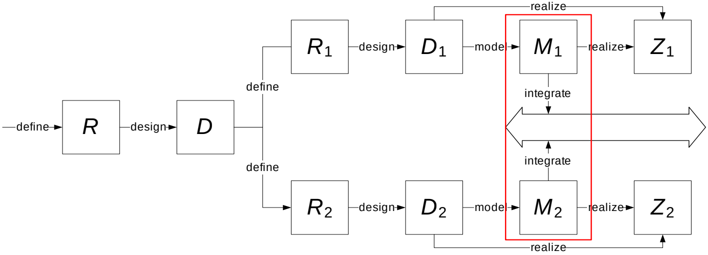
Model-based engineering



- Simulation allows early detection of *presence* of model/interface errors
- Verification allows proof of *absence* of model errors with respect to properties
- Controller synthesis allows *generation* of models that satisfy requirements by definition

Model-based engineering

Hybrid models



The combination of M_1 and M_2 often leads to hybrid models, e.g:

- M_1 is a discrete-event model of a supervisory control system
- M_2 is continuous-time (or hybrid) model of the controlled physical system

Model-based engineering

Hybrid models

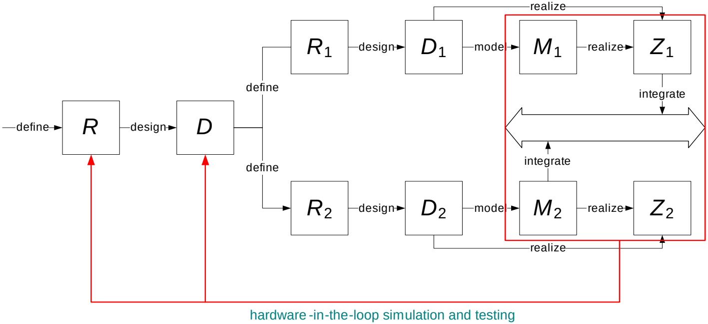
Hybrid models are composed of:

- Continuous-time models (DAEs: differential algebraic equations), including switched or switching sets of DAEs
- Discrete-time models (e.g. sampled systems)
- Discrete-event models (timed automata, process algebra), e.g. execution of a sequence of processing steps in a machine

Embedded system models can be composed of any combination of the models described above.

Model-based engineering

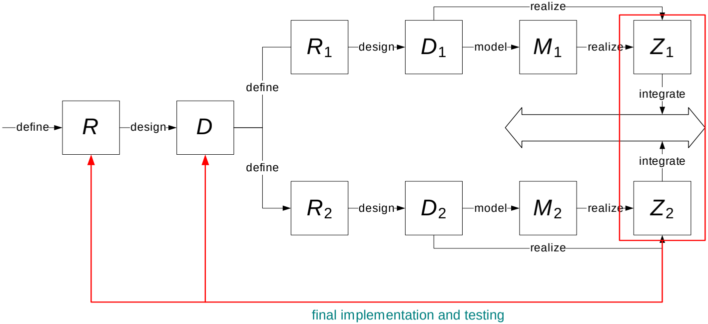
Hardware-in-the-loop simulation



- Hardware-in-the-loop simulation allows early detection of the *presence* of *realization* errors

Model-based engineering

Results



- Fewer errors in realizations (Q ↑)
- Enabler for automatic code generation (Q ↑, T ↓)

Language requirements for model based engineering

- Formal compositional semantics
- Concurrent
- Executable
- Modular and hierarchical
- Scalable
- Easy to use
- Stochastic
- Discrete-event, continuous-time and hybrid

Dynamics and control world view

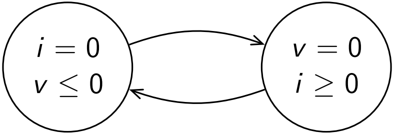
- Predominantly continuous-time system
- Modeled by means of DAEs (differential algebraic equations), or by means of a set of trajectories
- Hybrid phenomena modeled by means of discontinuous functions and/or switched equations, possibly using extended solution concepts (Filippov, Utkin) leading to sliding modes

$$(i = 0 \wedge v \leq 0) \vee (v = 0 \wedge i \geq 0)$$

DAE model of a diode

Computer science world view

- Predominantly discrete-event system
- Modeled by means of (timed/hybrid) automaton, process algebra, Petri net, data flow languages, etc.
- Evolution of a hybrid system: sequence of time transitions and action transitions
- Discontinuities are represented by actions



Automaton model of a diode

Simulation languages

- Ease of modeling $\Longrightarrow$ complex languages
- Verification not an issue, no formal semantics: (no verification)
- Languages specialize either in the discrete-event (DE) domain or in the continuous-time (CT) domain
- Hybrid languages usually DE^+ (E.g. Siman, Simple++) or CT^+ (E.g. Simulink, Modelica, EcosimPro)

Verification formalisms

- Ease of formal analysis $\Longrightarrow$ small languages with formal semantics
- Ease of modeling not an issue: cumbersome for modeling and simulation

Overview of the Chi language (1)

- Suited to:
  - simulation
  - verification
  - code generation
- Integrates:
  - discrete-event modeling (CS world view: automata, process algebra)
  - continuous-time modeling, (DC world view: switched differential algebraic equations)
  - discrete-time modeling (DC world view: sampled systems)
- Formal compositional semantics
- Consistent equation semantics of Chi ensures that equations are *always consistent*, comparable to invariants of hybrid automata

Overview of the Chi language (2)

- Is a process algebra defined by means of:
    - atomic statements, e.g. assignment ($x := 2$), DAE ($\dot{x} = -x + 1$)
    - an orthogonal set of operators, e.g. sequential comp. (;) and parallel comp. ($\|$)

  that can be freely combined.
- Core language small. Ease of use due to many syntactical extensions (all formally defined).
- Modular and hierarchical and scalable by means of process definition and process instantiation (reuse).
- Stochastic: definition of distributions and sampling.

The Chi language definition (1)

A Chi model is of the following form:

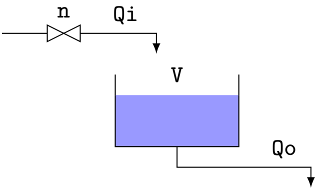
`model` *M*(*parameter declarations*) `=`
`|[` *channel and variable declarations*
`::` *p*
`]|`

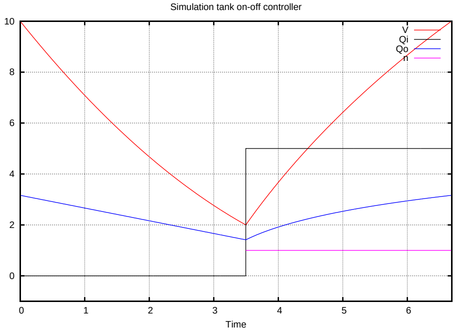
where *p* represents a process term (statement)

The Chi language definition (2)

	Process term	**Meaning**
p ::=	`skip`	internal action
|	x `:=` e	assignment
|	a `!` e	sending
|	a `?` x	receiving
|	`delay` e	delay statement
|	`inv` u	invariant (equations)
|	X	recursion variable
|	b `->` p	guard operator
|	p `;` p	sequential composition
|	p `||` p	parallel composition
|	p `|` p	alternative composition
|	`*`p	infinite repetition

Controlled tank system (1)



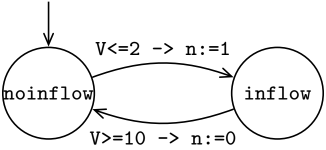


```
model ControlledTank()=
|[ var n: nat = 0, cont V: real = 10, alg Qi,Qo: real
:: inv dot V = Qi - Qo
 , Qi = n * 5
 , Qo = sqrt(V)
|| *( V <= 2 -> n:= 1; V >= 10 -> n:= 0 )
]|
```

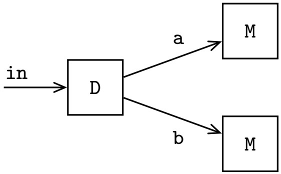
Controlled tank system (2)

Equivalent specification using modes, as in automata

```
model ControlledTank()=
|[ var n: nat = 0, cont V: real = 10
     , alg Qi,Qo: real
:: inv dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = sqrt(V)
|| |[ mode noinflow =
         V <= 2  -> n:= 1; inflow
    , mode inflow   =
         V >= 10 -> n:= 0; noinflow
   :: noinflow
   ]|
]|
```

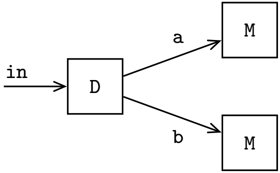


Distributor and Machines

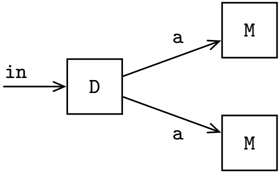


```
proc D(chan in?, out1!, out2!: nat) =
|[ var x: nat
:: *( in?x; ( out1!x | out2!x ) )
]|

proc M(chan in?: nat, val t: real) =
|[ var x: nat
:: *( in?x; delay t )
]|

proc DMM2(chan in?: nat) =
|[ chan a,b: nat
:: D(in,a,b) || M(a,4) || M(b,5)
]|
```

Distributor and Machines

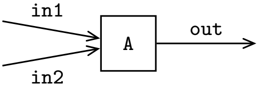


```
proc D(chan in?, out1!, out2!: nat) =
|[ var x: nat
:: *( in?x; ( out1!x | out2!x ) )
]|

proc M(chan in?: nat, val t: real) =
|[ var x: nat
:: *( in?x; delay t )
]|

proc DMM2(chan in?: nat) =
|[ chan a,b: nat
:: D(in,a,b) || M(a,4) || M(b,5)
]|
```



```
proc D(chan in?, out!: nat) =
|[ var x: nat
:: *( in?x; out!x )
]|

proc DMM2(chan in?: nat) =
|[ chan a: nat
:: D(in,a) || M(a,4) || M(a,5)
]|
```

Assembler



```
proc A( chan in1?, in2?: nat
      , out!: (nat,nat)
      , val t: real
      ) =
|[ var x,y: nat
:: *( ( in1?x || in2?y )
    ; delay t
    ; out!(x,y)
    )
]|
```

Tools for Chi

simulation

- Stand-alone symbolic simulator for hybrid and timed Chi (Python)
- S-function block hybrid Chi simulator for co-simulation in Matlab/Simulink
- Stand-alone simulator for timed Chi (C)

verification

- Translation of timed Chi to UPPAAL
- Translation of timed Chi to mCRL
- Translation of timed Chi to Promela/Spin
- Translation of hybrid Chi to PHAVer
- Prototype state space generator

real-time control

- Stand-alone Linux implementation

Conclusions

Chi language suited to model based engineering of dynamical (embedded) systems

- Concurrent process algebra allowing free combinations of statements and operators
- Integrates concepts from the dynamics and control theory with concepts from computer science
- Formal compositional semantics
- Integrates ease of modeling, simulation and verification
- Scalable: allows modular and hierarchical composition