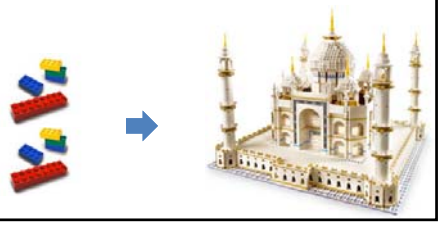
Compositional Model-Based System Design

Stavros Tripakis

Aalto University and UC Berkeley

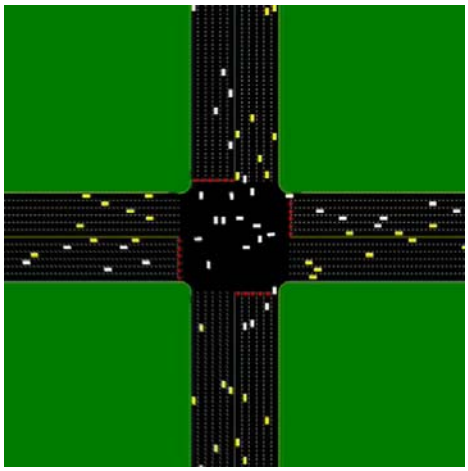


CRTS 2015, San Antonio, TX
December 1, 2015

The computer-controlled society

Example of a "smart" system (or CPS): autonomous intersection

Courtesy AIM project,
CS Dept., UT Austin



Autonomous intersection: "real-life" version

Courtesy `http://www.fastcodesign.com`
Thanks to Christos Cassandras for recommending this video

How to design safety-critical systems?

- Trial and error
  - Un-scalable
  - Un-economic
  - Un-safe
  - Yet common…

Are the drivers supposed to debug the autopilot?

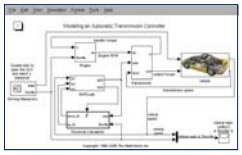
"It was described as a beta release. The system will learn over time and get better and that's exactly what it's doing. It will start to feel quite refined within a couple of months." – Elon Musk, Tesla CEO

Tesla autopilot video
(source: youtube)

Tripakis

A better approach: Model-Based Design

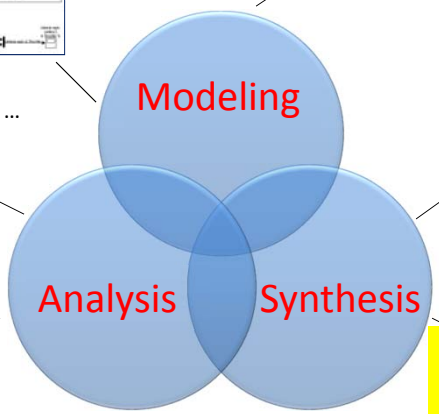
Simulink, UML, SysML, HDLs, SystemC, …



Describe the system that we want

Simulation, verification, …

Be sure that this is what we want

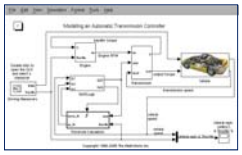


Implement the system Automatically Correct-by-construction

Tripakis

Compositional Model-Based Design

Industry: "*system integration is the real issue*"



How to model systems modularly?

Modeling

How to check properties in a compositional way?

Compositionality, modularity

Synthesis

How to implement parts of the system independently?

Tripakis

My work in the past 7 years

- Modeling:
  - Achieving compositionality in hi cal models
  - Multi-view modeling
- Analysis:
  - Increme gn using refinement theories
  - Co n of heterogeneous formalisms
- Synthesis/implementation:
  - Modular code generation

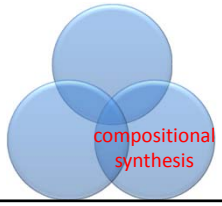
Common theme: **interfaces**

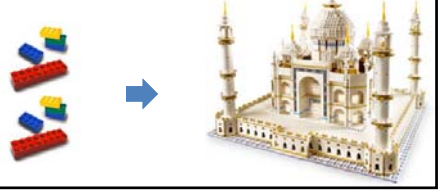
Disclaimer:
this talk is NOT a survey of compositionality/interfaces

Tripakis 8

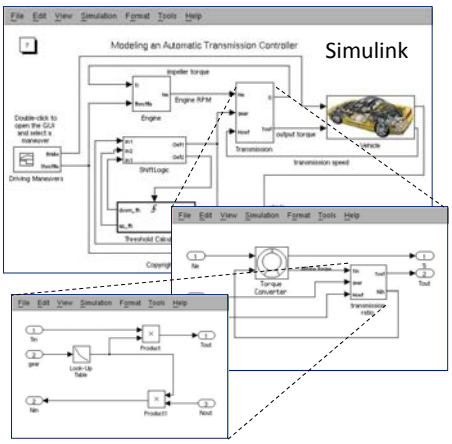
Modular code generation and compositionality in hierarchical modeling languages

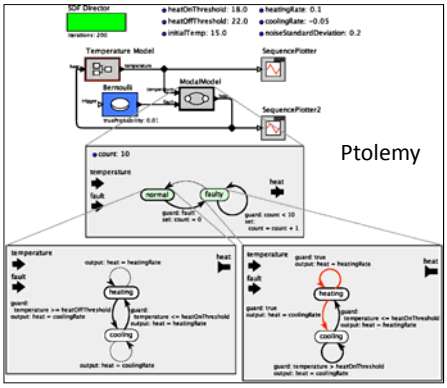
Joint work with R. Lublinerman, C. Szegedy, E. Lee, M. Geilen, B. Rodiers, D. Bui





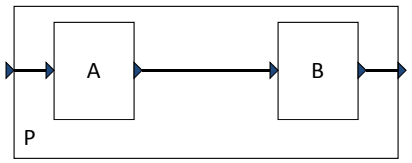
Hierarchy



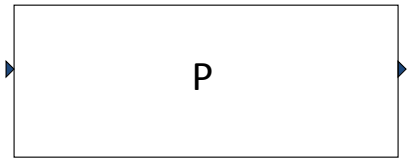


model = tree of sub-models

Hierarchy in block diagrams



Hierarchy in block diagrams

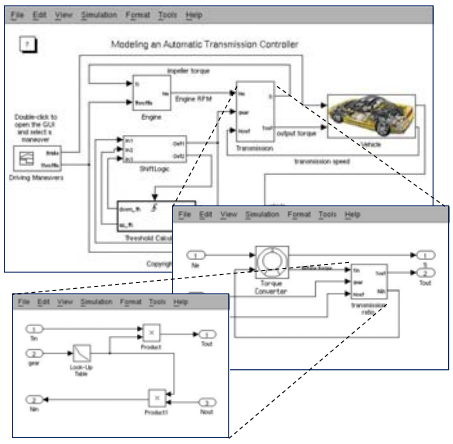


Modularization:
hide details, master complexity

Hierarchy benefits

Total number of blocks: ~100

Max. number at any level: ~6



model = tree of sub-models

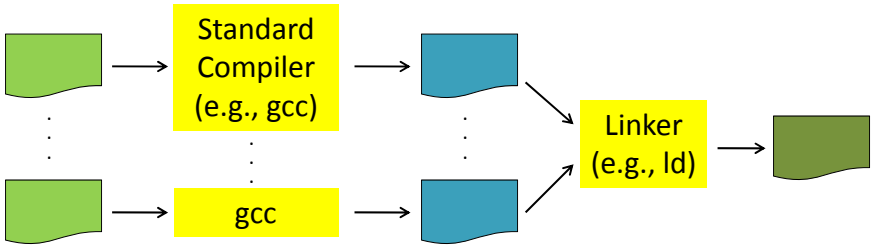
Can we exploit this hierarchy beyond syntax?

- Can we reuse a submodel? — c.f. software reuse
- Can we treat it as a "black box"? — c.f. software components
- Can we build model libraries? — c.f. software libraries

Answer: **no**, in most state-of-the-art languages and tools

These languages are fundamentally **non-compositional**
(the composition of two or more components is not always a component)

Example: suppose we want to do modular compilation



Source code (e.g., .cc files)

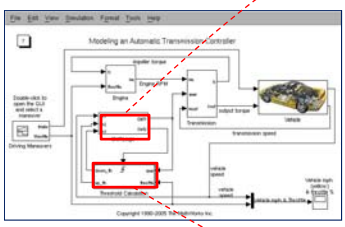
Object code (e.g., .o files)

Executable

Enables incremental compilation, IP protection, …, and libraries!

Modular code generation

Can we do the same for model-based design languages?



automatic code generation

```
initialize state;
while (true) do
  await clock tick;
  read inputs;
  compute;
  write outputs;
  update state;
end while;
```

Code for block A

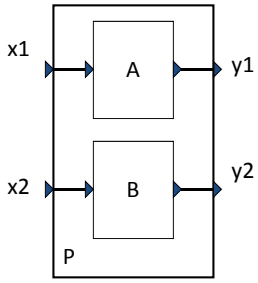
```
initialize state;
while (true) do
  await clock tick;
  read inputs;
  compute;
  write outputs;
  update state;
end while;
```

Code for block B

Code (C, C++, Java, …)

The standard approach

- "Monolithic" code generation

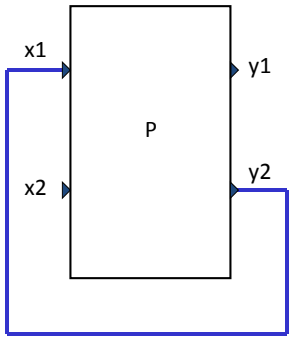



```
P.step(x1, x2) returns (y1, y2)
{
     y1 := A.step( x1 );
     y2 := B.step( x2 );

     return (y1, y2);
}
```


The standard approach

- Problem with monolithic code

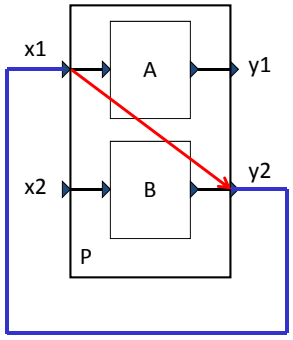



```
P.step(x1, x2) returns (y1, y2);
```

Non-compositionality of hierarchical block diagrams

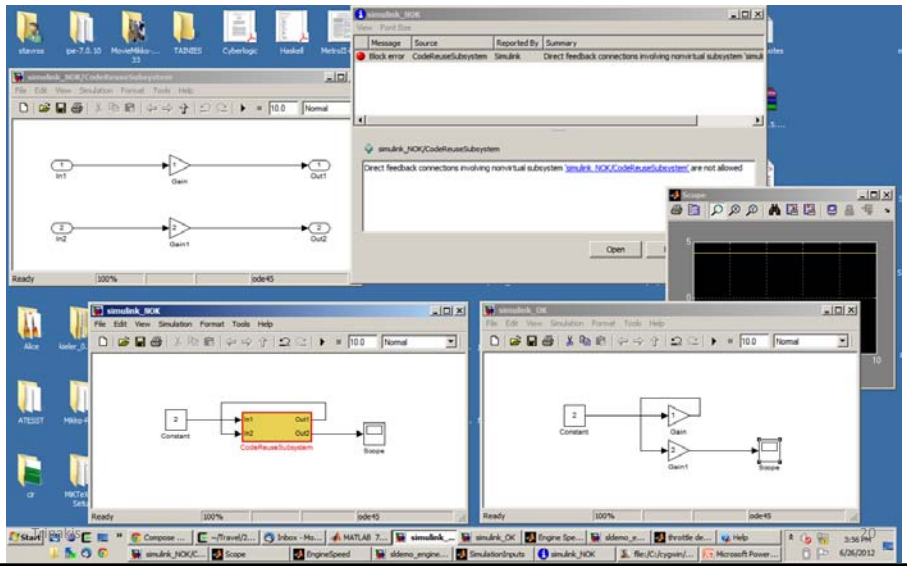
False I/O dependencies
=>
code not usable in some contexts

Parallel composition of functions is not a function!




```
P.step(x1, x2) returns (y1, y2);
```

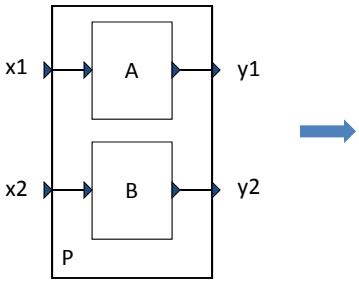

State-of-the-art tools (e.g., Simulink) suffer from this problem



[DATE'08, POPL'09]

Solution: non-monolithic code

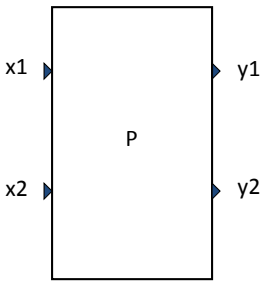
A state machine (Mealy) with multiple output functions




```
P.step1( x1 ) returns y1 {
   return A.step( x1 );
}

P.step2( x2 ) returns y2 {
   return B.step( x2 );
}
```

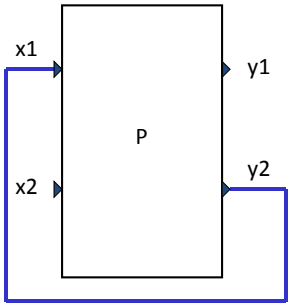

Non-monolithic interface




```
P.step1( x1 ) returns y1 ;

P.step2( x2 ) returns y2 ;
```

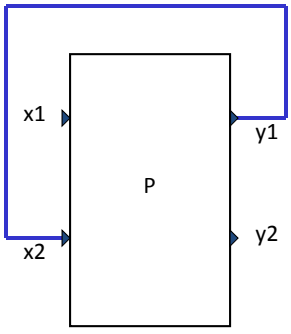
Non-monolithic interface




```
P.step1( x1 ) returns y1 ;
        ↑
P.step2( x2 ) returns y2 ;
```

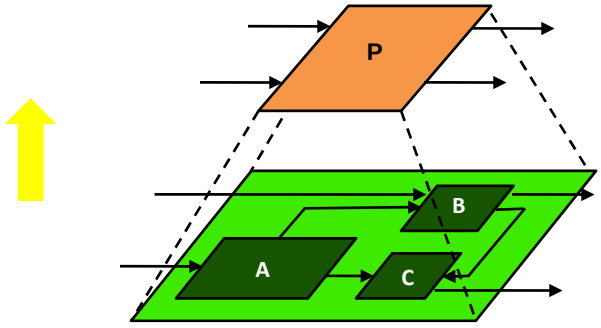

Non-monolithic interface

interface does not restrict usage



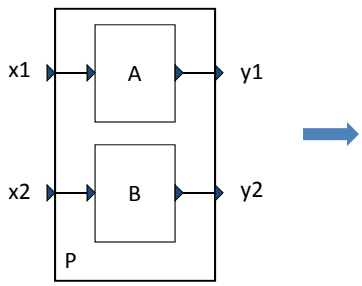

```
P.step1( x1 ) returns y1 ;
        ↓
P.step2( x2 ) returns y2 ;
```

Bottom-up interface synthesis



Given interfaces for sub-blocks A, B, C, compute interface for composite block P.

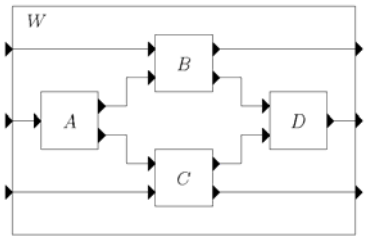
Interface synthesis: sometimes relatively simple …



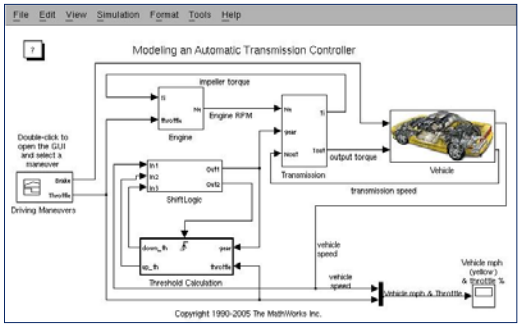

```
P.step1( x1 ) returns y1 ;

P.step2( x2 ) returns y2 ;
```

… but generally non-trivial



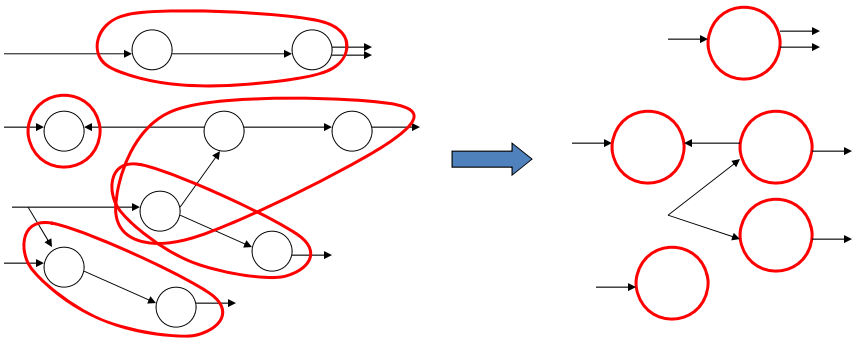
what about this?



or this?

Skip details

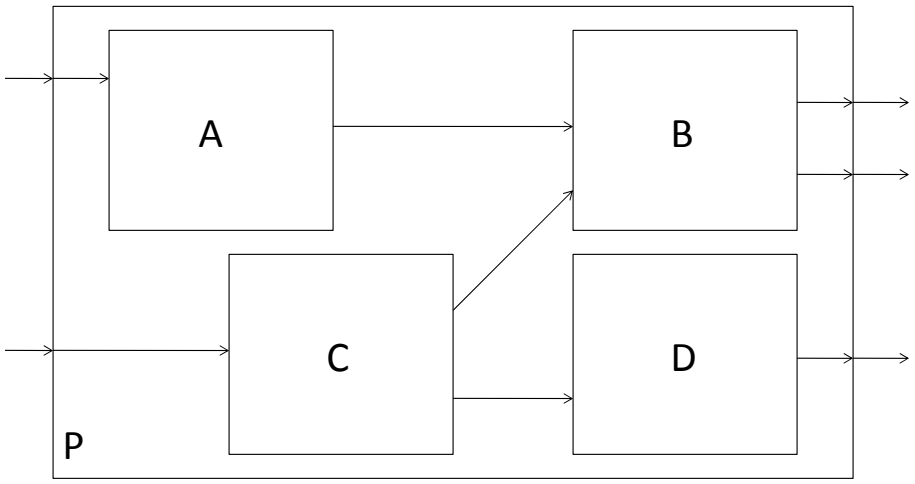
Interface synthesis for block diagrams = graph clustering



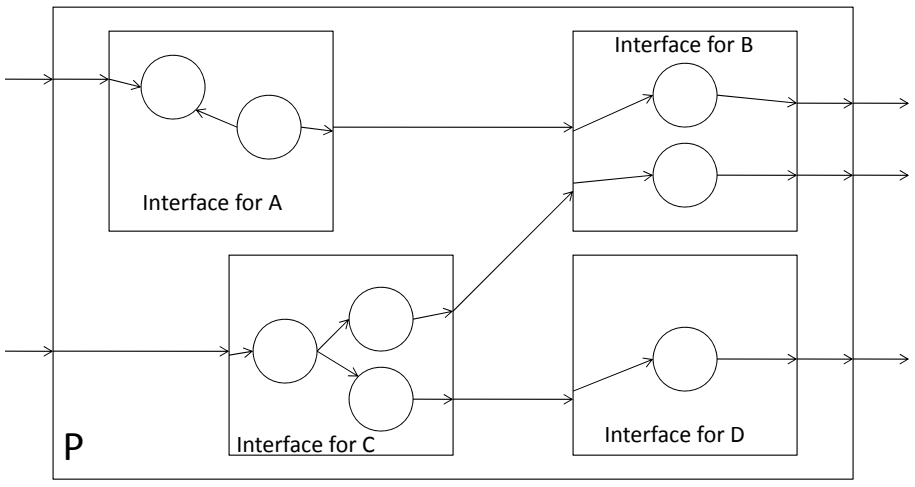
block diagram

interface

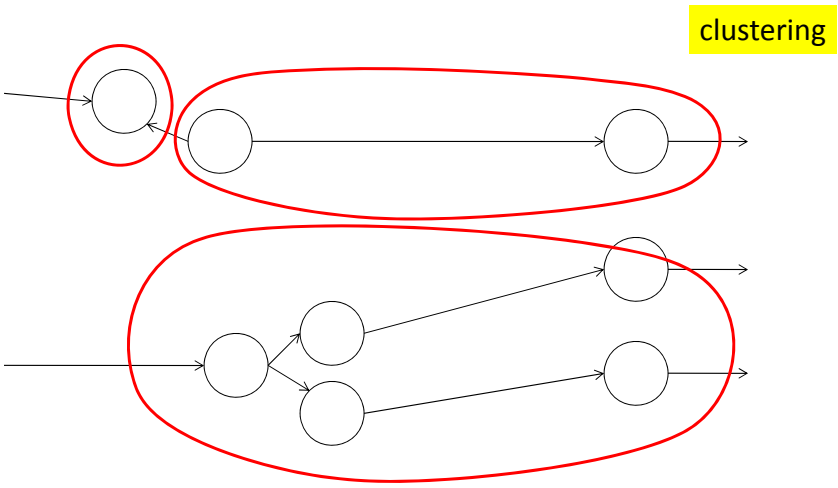
How it's done



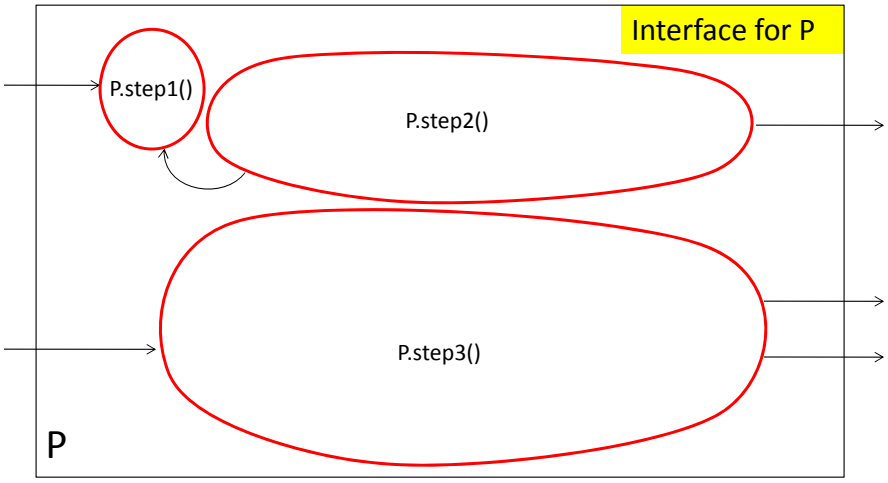
How it's done



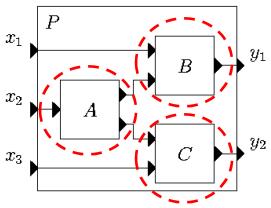
How it's done

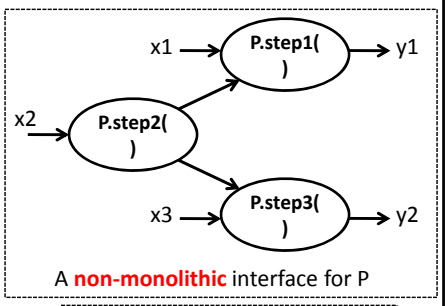


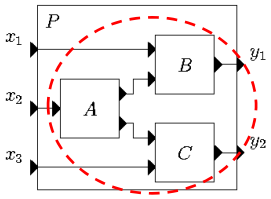
How it's done

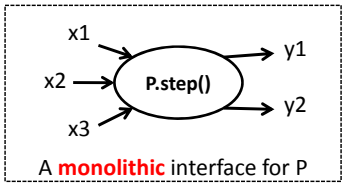


Different clusterings => different interfaces









trade-off: interface size vs. reusability

Different clustering algorithms = different tradeoffs

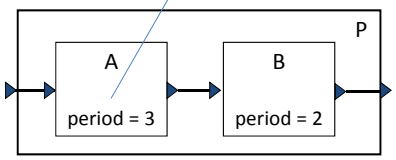
Clustering method	Complexity	Achieves maximal reusability?	Achieves minimal interf. size?	Modularity bound?	Achieves minimal code size?
"step-get"	Polynomial	No	Almost	<=2 functions	Yes
"dynamic"	Polynomial	Yes	Yes	<=N+1 functions*	No
"disjoint"	NP-complete	Yes	Yes	?	Yes
"greedy"	Polynomial	Yes	No	?	Yes

* N = number of block outputs

[RTAS'08]

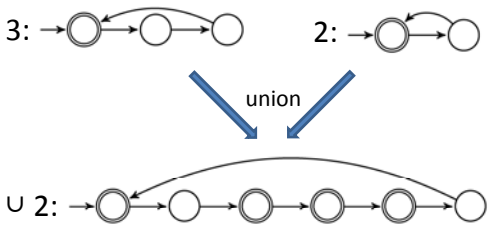
Achieving compositionality in multi-periodic diagrams

Period of block A = 3



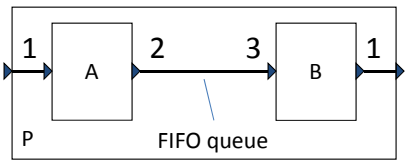
What is the period of P?
GCD(3,2) = 1 : over-approximation too conservative

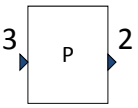
Interface enriched with deterministic unary automata



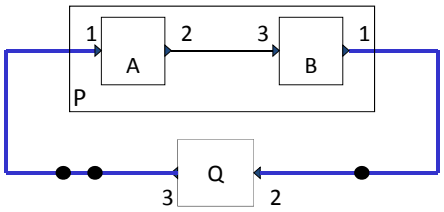
[ACM TECS 2013]

Non-compositionality of dataflow models (SDF)

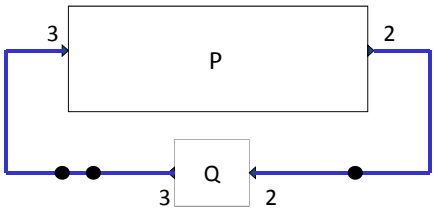




monolithic interface



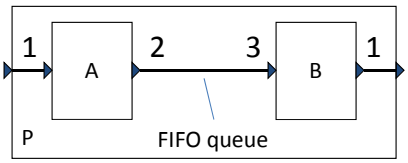
original model does not deadlock

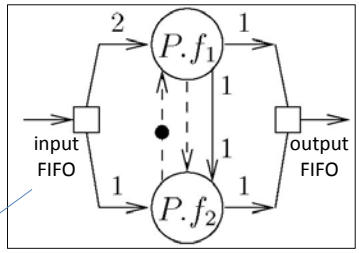


monolithic interface => deadlocks!

[ACM TECS 2013]

Interfaces for dataflow models





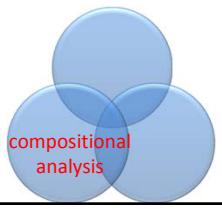
SDF++ :
- shared FIFOs
- but deterministic!

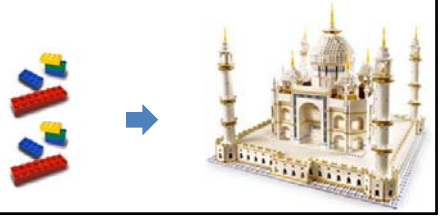
Non-monolithic interface for P
(generated automatically)

Incremental design using refinement theories

Joint work with B. Lickly, E. Lee, M. Geilen, M. Wiggers, C. Stergiou (Berkeley), T. Henzinger (IST Austria), M. Broy (TU Munich), V. Preoteasa, I. Dragomir (Aalto)

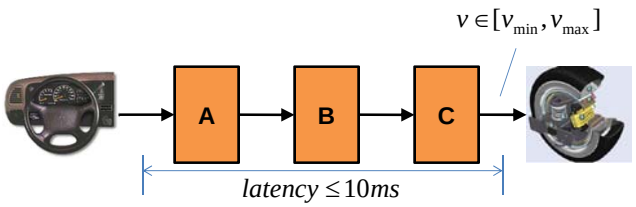
Part of NSF Project COSMOI





Incremental design

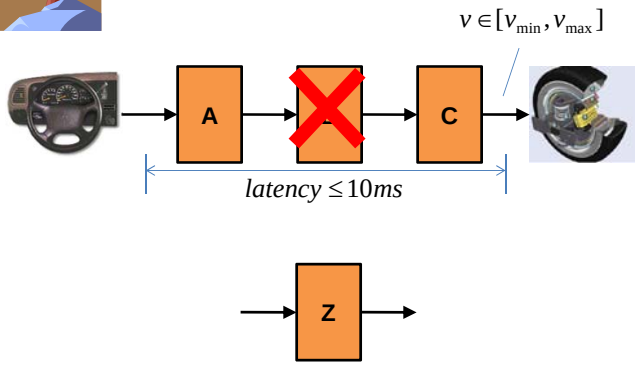
Suppose we have designed and verified this "steer-by-wire" system:



Tripakis 39

Incremental design

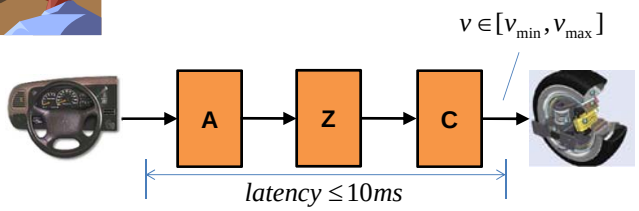
Suppose we want to replace B with Z:



Tripakis 40

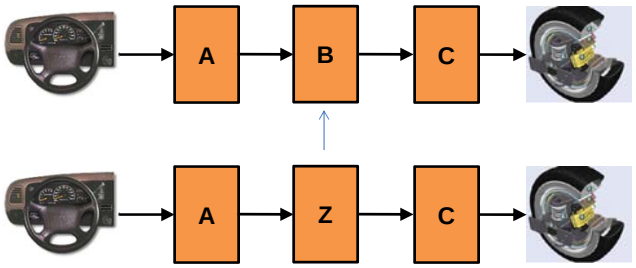
Incremental design

How to ensure properties are preserved (**substitutability**)?



Refinement theories

(behavioral type theories)

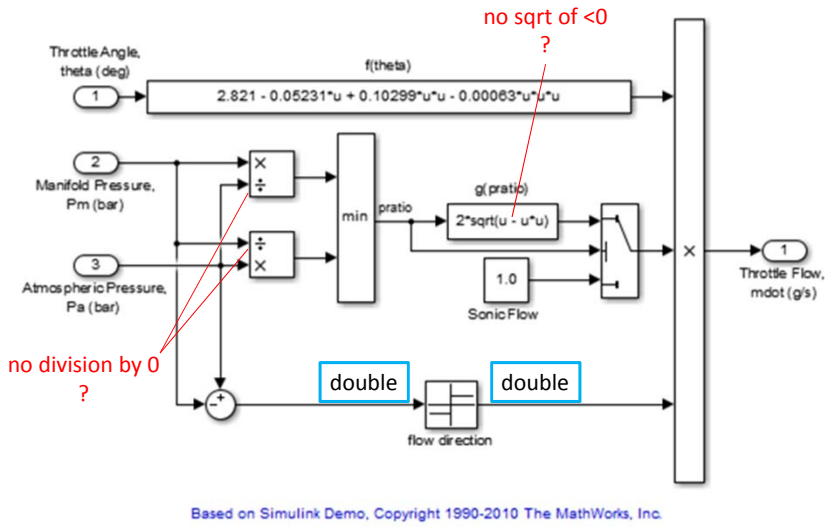


Z ≤ B:
Z *refines* B

(1) If A' ≤ A and A satisfies P then A' satisfies P.
(2) If A' ≤ A and B' ≤ B, then A' • B' ≤ A • B.

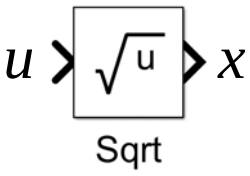
Z ≤ B and (1) and (2) => **substitutability!**

Types in Simulink



[ACM TOPLAS 2011]

Relational interfaces



`double -> double`

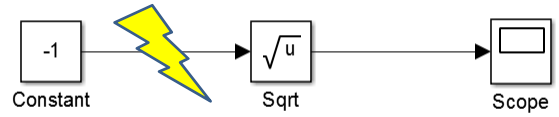
standard type

$u \geq 0 \wedge x = \sqrt{u}$

relational interface

Can describe systems which are both non-deterministic and non-input-receptive

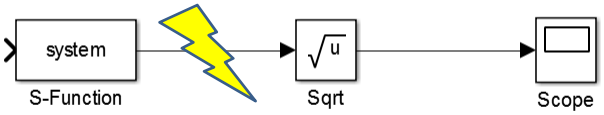
Catching incompatibility using symbolic methods



$u = -1$ $\quad u \geq 0 \wedge \cdots$

caught by taking the conjunction of the two formulas
=> use SAT/SMT solvers to check for satisfiability

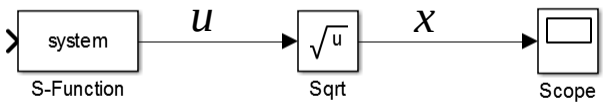
A more tricky example



$true$ $\quad u \geq 0 \wedge \cdots$

still incompatible, but ***not*** just conjunction of formulas

Serial composition involves ∀−∃ quantification

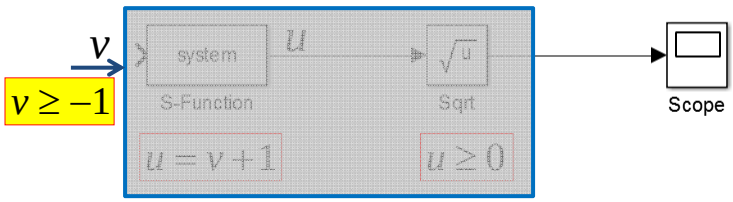


$true$

$u \geq 0 \wedge x = \sqrt{u}$

$\forall u: (true \Rightarrow \exists x: u \geq 0 \wedge x = \sqrt{u}) \equiv false$

Inferring new constraints on inputs (interface synthesis)



Refinement relation

$$\phi' \leq \phi \stackrel{\text{def}}{=} \begin{array}{l} in(\phi) \Rightarrow in(\phi') \\ (in(\phi) \wedge \phi') \Rightarrow \phi \end{array}$$

$$in(\phi) \stackrel{\text{def}}{=} \exists \text{outputs}: \phi$$

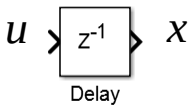
- Refinement <=> substitutability:

  A' can replace A in any context iff A' ≤ A.

  - i.e., refinement both necessary and sufficient condition for substitutability.
  - Note: sometimes the strongest contravariance condition $\phi' \Rightarrow \phi$ is used, which is sufficient but not necessary.

Tripakis 49

Handling components with state



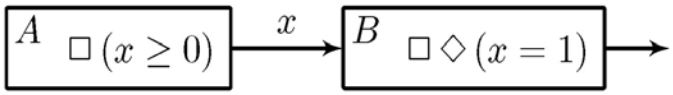
$$x = s \wedge s' = u$$

s : state variable

s' : next state variable

Tripakis 50

Expressing temporal properties in LTL (including liveness)



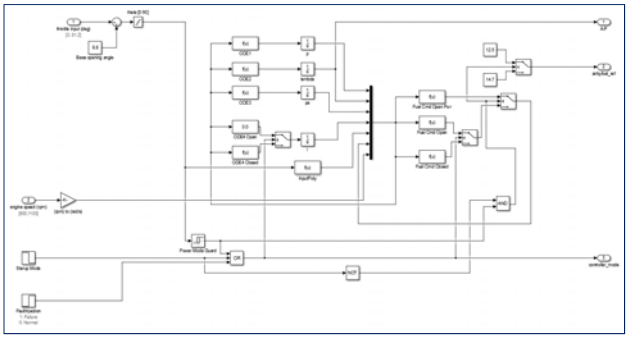
[EMSOFT 2014]

Tripakis

Latest work (in arxiv.org)

- Prototype tool:
  - Input: Simulink models
  - Theory implemented in Isabelle theorem prover
  - Translate Simulink model to relational interfaces and use Isabelle to automatically check and simplify formulas
- Case study: fuel control system by Toyota (3-level hierarchy, 70 blocks)

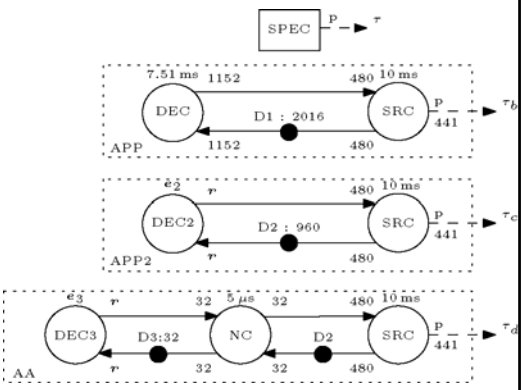
Sample subsystem of the FCS model



Tripakis

Another refinement theory for real-time properties: throughput and end-to-end latency

- Components described as dataflow models, timed automata, RTC, …
- Refinement relation: "the earlier the better"
- Worst-case throughput and latency (lower & upper bounds) preserved by refinement
- Under certain monotonicity assumptions (which don't always hold in real-life?)



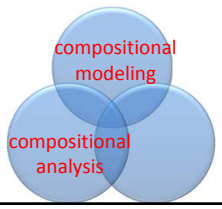
[HSCC 2011]

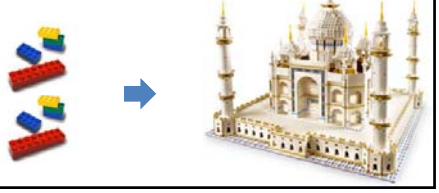
Skip

Multi-view modeling

Joint work with J. Reineke (Saarland) and
C. Stergiou (ex-Berkeley&Penn, now Google)

Part of NSF Project COSMOI

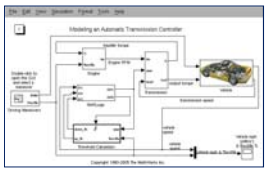




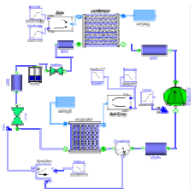
Multi-view modeling

Complex systems => different stakeholders => different concerns

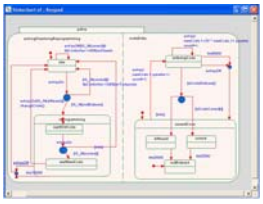
Multi-view modeling: model different **aspects** of the system (e.g., functionality, performance, energy, cost, …)



Low-level controllers
Simulink



Physical dynamics
Modelica



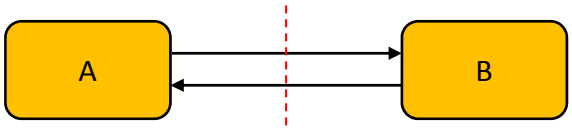
Supervisory controllers
Rhapsody/
SysML

Tripakis

55

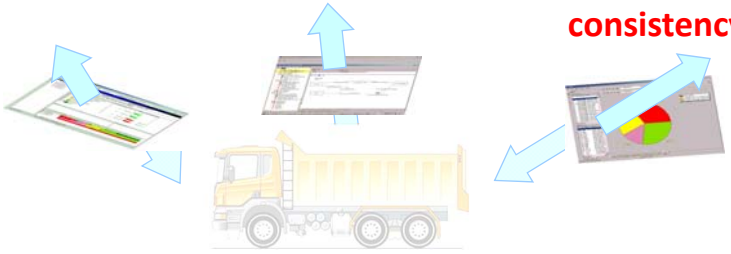
From (separate) components to (overlapping) views

- Composition theories tell us how to compose interacting but **separate** components:



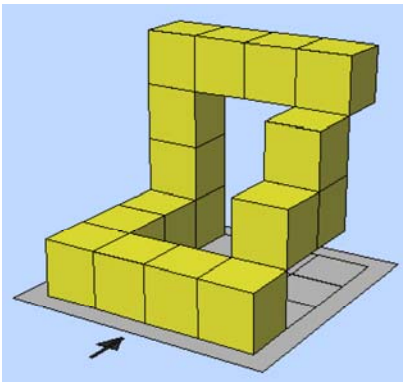
- Multi-view modeling
  - Views model **overlapping** aspects

Problem: how to guarantee view **consistency?**

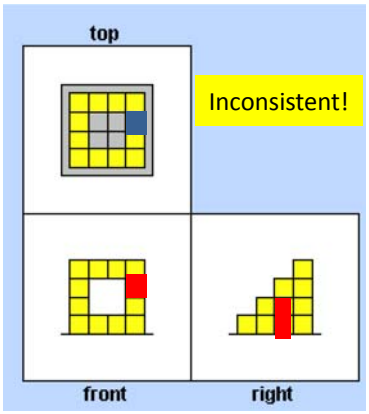


56

Example: Geometric (Static) Views



3D structure



2D views

57

Our work

[TACAS 2014, Journal paper under prep.]

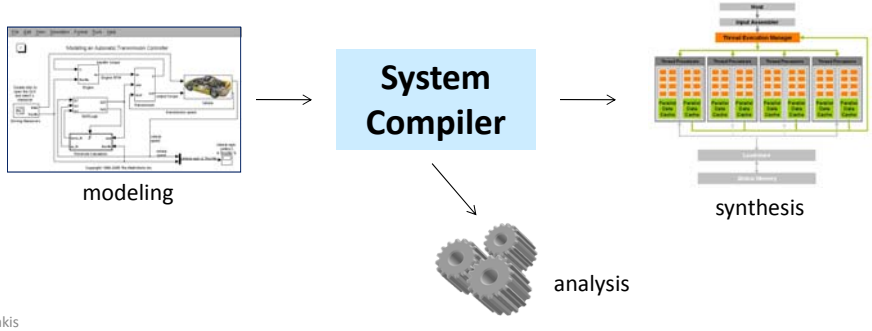
- **Dynamical** systems and views: languages, automata, symbolic transition systems, …
- Formalization of view **consistency** and other problems of multiview modeling
- Algorithms to solve these problems in concrete settings: finite automata, symbolic transition systems, …

Tripakis

58

Conclusions

- Model-based system design: key for the "smart societies" of the future, and more than a buzzword!
  - How to build "system compilers"
- **Compositionality**: key for complex system modeling, analysis, and implementation



Thank you

- Questions?