Compositional System Modeling with Interfaces COSMOI

PI: Stavros Tripakis **Co-PI: Edward A. Lee**

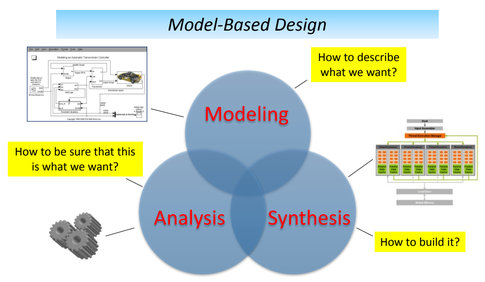
University of California, Berkeley

Model-based design

- Modeling: essential part of modern system design

Design ≈ Modeling

- Executable models: simulation, verification
- Find and fix design defects before building prototype => reduce costs



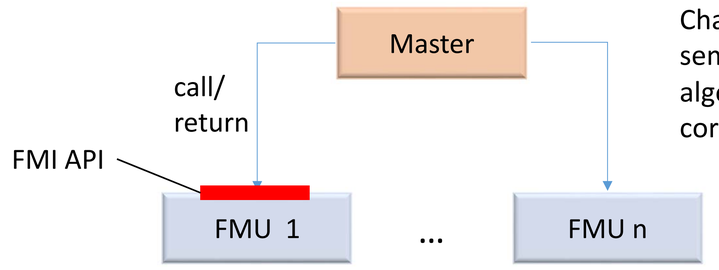
Problem: lack of compositionality

- Little support for model exchange, e.g., export model from tool A, import into tool B
- Little tool interoperability, little support for co-simulation
- Little support for model libraries: most models fundamentally non-compositional:
  - Composite submodel cannot be abstracted as an atomic "black box"
- Unsatisfactory alternative: copy-paste model code

This project: interface-based

- Interfaces for heterogeneous co-simulation
  - Leveraging work on recent standard: FMI (Functional Mock-up Interface)
- Interfaces for incremental design
  - Leveraging work on interface theories
- Interfaces for incremental implementation
  - Leveraging work on modular code generation
- Interfaces for multi-view modeling

Thrust: Co-simulation with FMI



Challenge: how to design a semantically sound master algorithm (determinism, correctness, max progress)

FMI formal API

- Notation:

C	set of FMU instances in a model
$c \in C$	FMU instance
S_c	set of states of FMU c
U_c	set of input ports of c
Y_c	set of output ports of c
$\mathbb{V}$	set of values that a port can take

- API's main functions:

$$\texttt{init}_c : \mathbb{R}_{\geq 0} \to S_c$$
$$\texttt{set}_c : S_c \times U_c \times \mathbb{V} \to S_c$$
$$\texttt{get}_c : S_c \times Y_c \to \mathbb{V}$$
$$\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0}$$

Additional API method:
$\texttt{getMaxStepSize}_c : S_c \to \mathbb{R}_{\geq 0} \cup \{\infty\}$

Deterministic master algorithm

At every simulation step:

1. set values for all input/output ports (using I/O dependencies)
2. save the states of all FMUs (to enable rollback)
3. set communication step size to an initial default value: $h := h_{max}$
4. find h acceptable by all FMUs:
   for each $c \in C$ do
   4.1 $h' := \texttt{doStep}_c(h_{max})$
   4.2 $h := min(h, h')$
5. assert $0 \leq h \leq h_{max}$
6. if $h < h_{max}$ then // roll back
   restore saved states of all FMUs;
   for each $c \in C$ do
   6.1 $h' := \texttt{doStep}_c(h)$
   6.2 assert $h' = h$
7. return h.

FMU contract:

(A0) If $\texttt{doStep}_c(s, h) = (s', h')$ then $0 \leq h' \leq h$.
If $h' = h$ then FMU accepts h, otherwise it rejects it.

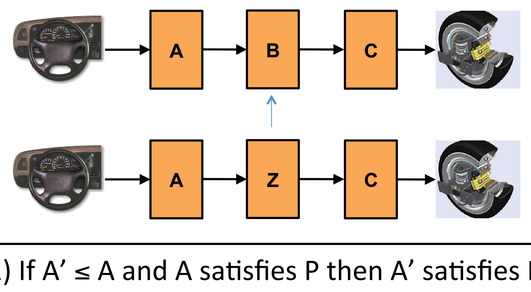
(A1) If $\texttt{doStep}_c(s, h) = (s', h')$, then for any h'' where $0 \leq h'' \leq h'$, $\texttt{doStep}_c(s, h'') = (s'', h'')$ for some s''.
If FMU accepts h', then it accepts any $h'' < h'$.

(A2) Let $s' = \texttt{set}_c(s, u, v)$. Then $s' = s[u := v]$.
set only changes the value of the variable being set.

(A3) Let $v = \texttt{get}_c(s, y)$ and $v' = \texttt{get}_c(s', y)$. If $s' = s[u_1 := v_1, ..., u_k := v_k]$, and output y does not directly depend on any input $u_1, ..., u_k$, then $v' = v$.
the value returned by get is the same when only irrelevant inputs change.

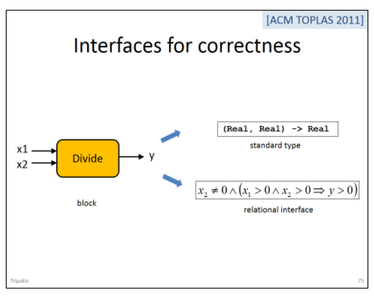
Thrust: Interface theories for incremental design [EMSOFT 2014]

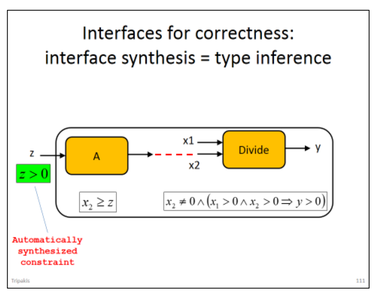
Substitutability with interface theories



(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



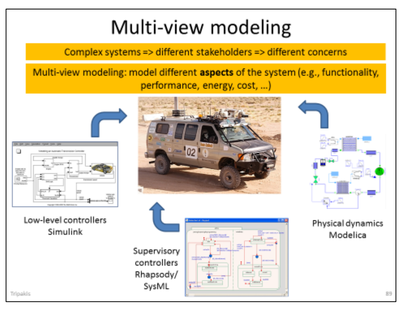


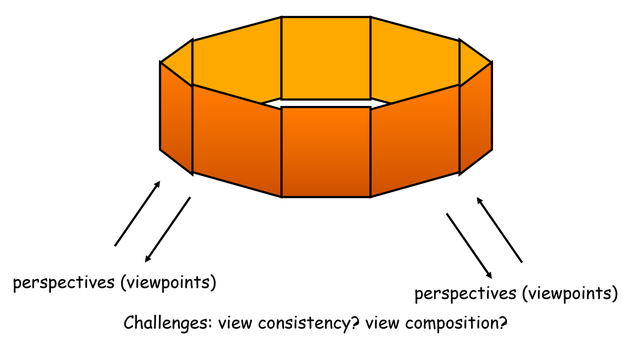
[EMSOFT 2014]

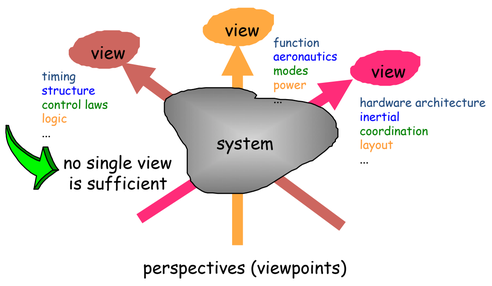
Recent work: Refinement Calculus for Reactive Systems

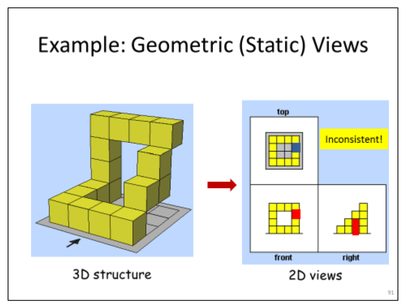
- Extends classic **refinement and weakest precondition calculus** [Dijkstra, Back, et al] to reactive systems.
- Components specified as symbolic transition systems or using LTL formulas.
- Can handle not only safety but also **liveness** properties

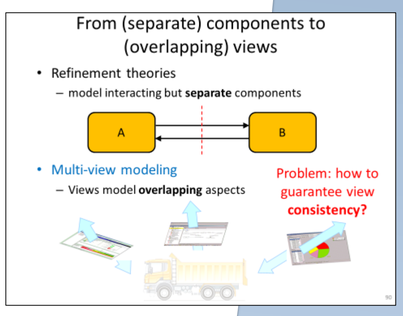
Thrust: Multi-view modeling [TACAS 2014]











[TACAS 2014]

Recent work

- From static to **dynamic** (i.e., dynamical system) views.
- Formalized views and view **consistency** (does there exist a system which could generate a given set of views?).
- Algorithms for:
  - Verification: view consistency checking
  - Synthesis: given set of consistent views, synthesize witness system.