Model-Based Design: Semantics-preserving implementation of synchronous models

Stavros Tripakis
UC Berkeley and Aalto University
Computers as parts of embedded systems

“~98% of the world’s processors are not in PCs but are embedded”

“a premium car today has:
- ~80 computers (ECUs – Electronic Control Units)
- ~100 million lines of code
- ~2km of wiring (CAN bus, other networks ...)”
Dependability vs. affordability

How to build systems that are both dependable and affordable?
Embedded system languages & tools

Simulink: 1 million licenses in 2004

Key concepts:
- reactive behavior
- concurrency
- timing
- I/O
- ...

Key capabilities:
- simulation
- code generation
- verification

LabVIEW

Modelica / Dymola

Simulink

SCADE
Vision

- These modeling languages of today will become the **system-programming** languages of tomorrow.

Richer languages: concurrency, time, robustness, reliability, energy, security, ...

More powerful analyses: model-checking, WCET analysis, schedulability, performance analysis, reliability analysis, ...

More complex execution platforms: networked, distributed, multicore, ...
Model-Based Design

How to describe what we want?

How to be sure that this is what we want?

How to build it? Automatically Semantics-preserving
Semantics-preserving implementation of synchronous models

**Design**
- Discrete-Time Simulink
- Lustre
- Ptolemy SR

**Implementation**
- Single-processor single-task
- Distributed, synchronous (TTA)
- Single-processor multi-task
- Distributed, asynchronous (KPN, LTTA)

**Application**
- Execution platform
Synchronous block diagrams

blocks = state machines

connections = “wires”
Synchronous semantics

Time = sequence of steps / cycles / rounds / reactions

“logical” time (synchrony)
not necessarily periodic
Computation within a step

- Propagate local outputs throughout the diagram
- Feedback: cyclic dependencies “broken” by unit delays

local outputs => local inputs
Questions

• Can synchronous semantics be implemented?
  – What does this mean?

• On which types of execution platforms?
  – Distributed?
  – Asynchronous?

• Under which conditions?
Reactive computation in real life: periodic control loop

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

“bare-iron” single-processor, single-task implementation
Semantical preservation for single-task implementations

- Semantics (I/O streams) is preserved provided:

\[ WCET(\text{reaction}) < \text{period} \]
Single-processor, multi-task

[Scaife, Caspi, ECRTS 2004]
[Tripakis, Sofronis, Scaife, Caspi, EMSOFT 2005]
[Sofronis, Tripakis, Caspi, EMSOFT 2006]
[Caspi, Scaife, Sofronis, Tripakis, ACM TECS 2008]
Multi-task implementations: what & why?

• What
  – One computer + RTOS + many tasks running “concurrently”
  – Preemptive scheduling: fixed-priority, EDF, ...

• Why
  – For multi-rate applications
  – Example: block A runs at 10 ms, block B runs at 40 ms
  – Model: synchronous block diagrams with (logical) triggers
“Naïve” multi-task implementation

- Tasks = threads communicating with shared memory
- Use locks/semaphores to protect access to shared variables
Synchronous semantics generally not preserved by “naïve” implementation.
Synchronous semantics generally not preserved by “naïve” implementation.

\[ x(k+1) \xrightarrow{\mathrm{ERROR}} x(k) \]

\[ \text{Prio}_Q \succ \text{Prio}_A \succ \text{Prio}_B \]

Real implementation.
The Dynamic Buffering Protocol (DBP)

- Key idea: memorize the order of task arrivals
- Semantics-preserving
  - Assuming schedulability
- Memory-optimal
- Wait-free
- For RMA or EDF

Communication:
\[ \tau_w \rightarrow \tau_i, \text{ for } i = 1, \ldots, N_1, \quad \tau_w \rightarrow^1 \tau_i, \text{ for } i = N_1 + 1, \ldots, N_1 + N_2, \text{ and } \tau_w \rightarrow^1 \tau'_i, \text{ for } i = 1, \ldots, M. \text{ Let } N = N_1 + N_2. \]

Task \( \tau_w \) maintains a buffer array \( B[1..N+2] \), one pointer array \( R[1..N] \) and two pointers current and previous.

Each task \( \tau'_i \), for \( i = 1, \ldots, M \), maintains a local pointer \( P[i] \).
All pointers are integers in \( [1..N+2] \). A pointer can also be null.
Initially, current = previous = 1, all \( R[i] \) and \( P[i] \) are set to null, and all buffer elements are set to \( y^d_0 \).

During execution:

**Writer**
- When \( \tau_w \) is released:
  - previous := current;
  - current := some \( j \in [1..N+2] \) such that \( \text{free}(j) \), where
    \[ \text{free}(j) \equiv (\text{previous} \neq j \land \forall i \in [1..N], R[i] \neq j). \]
- While \( \tau_w \) executes it writes to \( B[\text{current}] \).

**Lower-priority reader \( \tau_i \):**
- When \( \tau_i \) is released:
  - if \( i \in [1..N_1] \) then \( R[i] := \text{current} \) (link \( \tau_w \rightarrow \tau_i \))
  - else \( R[i] := \text{previous} \) (link \( \tau_w \rightarrow^1 \tau_i \))
- While \( \tau_i \) executes it reads from \( B[R[i]] \).
- When \( \tau_i \) finishes:
  - \( R[i] := \text{null} \).

**Higher-priority reader \( \tau'_i \):**
- When \( \tau'_i \) is released:
  - \( P[i] := \text{previous} \).
- While \( \tau'_i \) executes it reads from \( B[P[i]] \).

Applicable under static-priority scheduling when
\[ \forall i = 1, \ldots, N . p_i < p_w \text{ and } \forall i = 1, \ldots, M . p'_i > p_w. \]

Applicable under EDF scheduling when
\[ \forall i = 1, \ldots, N . d_i > d_w \text{ and } \forall i = 1, \ldots, M . d'_i < d_w. \]
Distributed, synchronous (TTA)

[Caspi, Curic, Maignan, Sofronis, Tripakis, Niebert, LCTES 2003]
[Tripakis, Sofronis, Caspi, Curic, ACM TECS 2005]
The project

- European IST projects:
- Automotive applications:
  - Audi.
Time Triggered Architecture (TTA)

- Time-triggered:
  - Processors synchronize their clocks.
  - Static TDMA non-preemptive scheduling for tasks running on processors and messages transmitted on the bus.
- Fault-tolerance services
From Lustre to TTA

- Conceptually easy: both are synchronous

- Practically: non-trivial resource-allocation problems
  - Decomposition of Lustre program into tasks
  - Allocation of tasks to processors
  - Scheduling of tasks and messages (also WCET analysis)
  - Generation of “glue code”

- To “help” the compiler: Lustre extensions (“pragmas”):
  - Real-time primitives (WCET, deadlines, …)
  - Distribution primitives (which part is to be executed where)
Distributed, asynchronous

[Tripakis, Pinello, Benveniste, Sangiovanni-Vincentelli, Caspi, Di Natale, IEEE TOC 2008]
Implementation on asynchronous distributed platforms

Asynchronous distributed platform:
- Many computers, each with a local clock
  - No clock synchronization
- Computers communicate using some network/protocol
  - Don’t care which network, as long as finite FIFO queues (TCP) can be implemented on top

Synchronous model

Asynchronous platform with some communication network
Implementation on asynchronous distributed platforms

synchronous model

Intermediate layer

asynchronous platform with some communication network
Implementation on asynchronous distributed platforms

*Intermediate layer:* asynchronous processes communicating with finite FIFO queues

This is like Kahn Process Networks but here FIFOs are finite.

FIFOs must be large enough to avoid deadlocks.

Thm: \( \text{size(FIFOs)} \geq 2 \Rightarrow \text{semantical (stream) preservation} \\
(also tighter bounds)
Intermediate layer: FFP

Similar to a Kahn process network but FIFOs are finite
How to go from Sync to FFP?

- **Theorem**: semantics is preserved provided queues have sufficient sizes
  - Sufficient: 1-place FIFOs for non-unit-delay links, 2-place for UDs
  - Tighter bound: at least \( m+1 \) places for every loop, where \( m \) is number of UDs in that loop
- Semantical preservation = **stream** preservation

```plaintext
while (true) {
    await clock trigger;
    if (exists empty input queue OR full output queue) {
        /* skip */
    } else {
        read from input queues;
        compute;
        write to output queues;
    }
}
```
Why 2 place queues for unit-delays?

M1

UD

UD

M2

What would happen if both queues had size =1?
Why 2 place queues for unit-delays?

Deadlock!
Proving preservation

• Used old theories [1970s]

• Marked graphs [Commoner et al ‘71]
  – Used to show that FFP is live: every process can fire infinitely often (does not deadlock)
  – Therefore all streams are infinite

• Kahn Process Networks [Kahn’74]
  – Every KPN is determinate: streams do not depend on process interleaving
  – One possible interleaving is the static one of the original synchronous diagram: this obviously yields the same streams as in the synchronous semantics
  – Thus every other interleaving will also yield equal streams
Proving liveness

- View FFP as a marked graph

- Marked graphs:
  - Subclass of Petri Nets
    - Every place has a unique input and a unique output transition

- Theorem [Commoner et al]:
  - A marked graph is live iff every loop has positive token count
  - (token count invariant in a loop)
Example 1

Petri net with bounded-capacity places

Marked graph
Example 2: deadlock

Petri net with bounded-capacity places
Performance analysis

• **Throughput:**
  – How often processes fire (vs. skip) – i.e., how often outputs are produced

• **Latency:**
  – What is the life-span of a token from production to consumption

• **Metrics:** *real-time* (RT) vs. *logical-time* (LT)
  – Real-time: metrics depend on real-time behavior of clocks (e.g., clock rates)
  – Logical-time: metrics depend only on topology and queue sizes!

• **Results:**
  – Algorithms to compute worst-case LT throughput/latency for arbitrary topologies
  – Theorems to compute them analytically (formulas) for special topologies
  – Theorems that bound RT metrics given LT metrics and bounds on clock rates
Logical-time throughput: example

Worst-case scenario:

LT throughput:

½ times

LT thput = 1/2

no process skips after that!
Relating real-time and logical-time throughput

Theorem 9: Let $F$ be an SFFP. Let $\Delta$ be any positive real number. Let $c$ be a vector of clocks such that $\forall i, \forall n, c_i(n + 1) - c_i(n) \leq \Delta$. Then, for any process $P_i$ of $F$:

$$\lambda^{rt}(F, P_i, c) \geq \frac{\lambda^*(F, P_i)}{\Delta}$$
Computing worst-case logical-time throughput

Reachability lasso of marked graph

LT thput = 3/4

Deterministic firing policy

4 places, 4 transitions
Place bounds: [3,2,2,2]
Initial markings: [[0,0,0,0]]
Incidence matrix (row-by-row, rows are places): [1,0,0,-1,1,-1,0,0,1,-1,0,0,0,1,-1]
Performance analysis

• **Throughput**:  
  – How often processes fire (vs. skip) – i.e., how often outputs are produced

• **Latency**:  
  – What is the life-span of a token from production to consumption

• **Metrics**: *real-time* (RT) vs. *logical-time* (LT)  
  – Real-time: metrics depend on real-time behavior of clocks (e.g., clock rates)  
  – Logical-time: metrics depend only on topology, queue sizes and initial conditions!

• **Results**:  
  – Algorithms to compute worst-case LT throughput/latency for arbitrary topologies  
  – Theorems to compute them analytically (formulas) for special topologies  
  – Theorems that relate RT metrics and LT metrics
Real-time and logical-time throughput: definitions

\[
\lambda^{rt}(\mathcal{F}, P_i, c) = \lim_{t \to \infty} \frac{\text{fireno}_{\mathcal{F},c,P_i}(t)}{t}
\]

\[
\lambda^{lt}(\mathcal{F}, P_i, c, \chi) = \lim_{n \to \infty} \frac{\text{fireno}_{\mathcal{F},c,P_i}(\chi(n))}{n}
\]

\[
\lambda^{wclt}(\mathcal{F}, P_i, \chi) = \inf_{c \in C(\chi)} \lambda^{lt}(\mathcal{F}, P_i, c, \chi)
\]
The “slow” triggering policy

- At each synchronous cycle:
  - First trigger all disabled processes (they will skip)
  - Then trigger all enabled processes (they will fire)
  - Note:
    - Each queue has a unique reader and writer
    - Therefore firing one process cannot disable another
    - Thus the order in which the enabled processes are fired does not matter

- **Theorem**: worst-case logical-time throughput is achieved by real-time clocks that follow the slow triggering policy
Logical-time throughput: example

Worst-case scenario:

LT throughput:

½ times

LT thput = 1/2

P2 skips
P1 skips

no process skips after that!
Computing the worst-case logical-time throughput

deterministic (slow) firing policy

Reachable lasso of marked graph

LT thput = 3/4
Demo

Incidense matrix (row-by-row, rows are places): \[
\begin{bmatrix}
1 & 0 & 0 & -1 \\
1 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 \\
0 & 0 & 1 & -1
\end{bmatrix}
\]

4 places, 4 transitions

Initial markings: \([0,0,0,0],[0,0,0,1]\)
Worst-case logical-time throughput: connected networks

**Theorem 8** Given a connected SFFP $F$,

$$\forall P_i, P_j \in F, \lambda^*(F, P_i) = \lambda^*(F, P_j).$$

i.e.: for connected networks, WCLT throughput is the same for all processes

Intuition: total # tokens produced = total # tokens consumed
Relating real-time and worst-case logical-time throughput

**Theorem 9:** Let $\mathcal{F}$ be an SFFP. Let $\Delta$ be any positive real number. Let $c$ be a vector of clocks such that $\forall i, \forall n, c_i(n + 1) - c_i(n) \leq \Delta$. Then, for any process $P_i$ of $\mathcal{F}$:

$$\lambda^{rt}(\mathcal{F}, P_i, c) \geq \frac{\lambda^*(\mathcal{F}, P_i)}{\Delta}$$
Latency: defined w.r.t. a path

\[ \mu^{rt}(\mathcal{F}, \pi, c) = \sup_{z} \text{travel}_{\mathcal{F}, c, \pi}(z) \]
Latency: defined w.r.t. a path

\[ \mu^{lt}(\mathcal{F}, \pi, c, \chi) = \sup_z \text{travel}^{\chi}_{\mathcal{F}, c, \pi}(z) \]

\[ \mu^{wclt}(\mathcal{F}, \pi, \chi) = \sup_{c \in \mathcal{C}(\chi)} \mu^{lt}(\mathcal{F}, \pi, c, \chi) \]
Worst-case logical-time latency: computation

How long until this token gets consumed?

3 firings of P2
Worst-case logical-time latency: computation

• Compute reachability graph

• For every reachable marking m s.t. the first queue in the path is non-empty:
  – Compute $T(m) = \text{sum of all tokens in the path}$
  – Compute $L(m) = \#\text{steps it takes to fire the destination process } T(m) \text{ times (w.r.t. the slow policy)}$

• WCLT latency = $\max L(m)$ over all such m

unfortunately not just a lasso ...
Demo

1 places, 2 transitions
Incidence matrix (row-by-row, rows are places): [1,-1]
Place bounds: [1]
Initial markings: [[0]]

1 places, 2 transitions
Incidence matrix (row-by-row, rows are places): [1,-1]
Place bounds: [2]
Initial markings: [[0]]
Relating real-time and worst-case logical-time latency

**Theorem 17** Let \( F \) be an SFFP. Let \( \Delta \) be any positive real number. Let \( c \) be a vector of clocks such that \( \forall i, \forall n, c_i(n + 1) - c_i(n) \leq \Delta \). Then, for any path \( \pi \) of \( F \):

\[
\mu^{rt}(F, \pi, c) < \Delta \cdot (\mu^*(F, \pi) + 1)
\]
Conclusions

• Synchronous models
  – Widely used in the industry
  – A simple, deterministic model of concurrency (unlike threads)

• Semantics-preserving implementation
  – On synchronous platforms (TTA)
  – On asynchronous platforms (centralized multi-task, distributed with non-synchronized clocks)

• Future directions
  – Robustness, fault-tolerance
Thank you

• Questions?