Generating Protocol Converters from Scenario-based Specifications

Abhik Roychoudhury  P.S. Thiagarajan  Tuan-Anh Tran
National Univ. of Singapore and Vera A. Zvereva
State Univ. of St. Petersburg, Russia

The problem
- Re-use of existing components for large scale system level design.
- Enabling communication among components a must
- Components supplied by different vendors
  - Interface behavior is documented in some form.
  - Interfaces of different components may not be suitable for plug-and-play!
- Need to synthesize interconnect fabric.

A typical desired situation

The actual situation

Or, even better if ...

Known approach
- Describe behaviors of interface of each component as an FSM.
- Construct product of interface FSMs to get protocol converter as an FSM where
  - Action names in product FSM replaced by dual
  - ?a never precedes !a  [Passerone et. al DAC98]
- Powerful enough to take care of
  - Disparate component alphabets
  - Incompatible action sequences
  - Storage capability of the converter plays a role.
What are the tasks of the converter here?

Allowing this behavior may require storing D.
Is this enough?
- The example protocols refer to only one “episode” between two components
  - Transfer of data D
- Extensions:
  - Protocols among multiple components
    - Take the product of all the interface FSMs
    - Protocols spanning several episodes
  - Runs of infinite length in each native protocol
  - All episodes combined in FSM [Passerone et. al. '02]
- Can we think about the problem per-episode?

Protocol conversion, but
- ... an episode at a time.
  - Getting a per-episode inter-component description from the interface of each component
  - Flow between episodes captured as a graph
  - Using the inter-component description for
    - Protocol conversion of a single episode
    - Protocol conversion for a graph of episodes
- In this paper
  - The protocol conversion problem.

Describing an episode
- Collection of Message Sequence Charts (MSCs)

Converter for an episode
- Consume input signals
- Produce output signals
- Receive and Forward Shared signals

Additional capabilities exist, more on this later
But isn’t this similar to the intra-component approach?

Definition of Edge N1 → N2 in the transition sys.
- Asynchronous concatenation
- One process may enter episode N2 even when other processes are still executing N1.
- Only bounded overtaking enforced in our descriptions.
  - Bounded by loop sizes
  - A process cannot enter an episode N if a previous copy of N is still active.
- Alternative notion: Synchronous Concatenation
  - All processes synchronize after each episode.

Converter for each episode
- Can be viewed as an FSM
- Soups up communication of the converter with different processes
  - Alternative messy to link up converters for episode sequences in presence of asynchronous concatenation
- Alternative view for an episode’s converter
  - Multi-threaded program with threads T1, ..., Tk
  - One thread for each component in the protocol
  - Thread Ti communicates with component Pi
  - Do the converter threads need to communicate?
    - For the shared signals, such as ...

Is it now any easier to link up the converters for each episode?
Implementation

Component Views (SystemC)
Path thru Episode graph
Simulation run

Example : Split Transfers

Sample simulation trace

Our work
- How to describe
  - Inter-component interactions and their episodes
- How to synthesize
  - Component interfaces/converters
    - Interface for one episode
    - Interface for entire episode graph
  - Clock sensitivities and timers (handle timeouts etc)
    - Brings in synchronicity into our async. description
- Modeling and converter synthesis for various features of System-on-Chip Bus protocols.
- Can we always synthesize converters?

Future Work
- Extensions to converter capabilities
  - Data formatting (chopping/merging packets)
- Synthesizing the episodes and episode graph from the SystemC description of component interface
  - SystemC → Inter-component models → SystemC
  - Currently looking at conformity of a given episode graph with SystemC description of interfaces.

More aggressive converters
Speculatively generate shared signals even before they are received - integrated into our impl.
But, cannot speculate on data values s, t could be data signals