Hierarchical FSMs with Multiple CMs

Reference:
Outline of my presentation

- What does the paper talk about?
- Why are these ideas useful?
- Some FSM definitions
- Combining FSMs and MOCs: preliminaries, semantics and subtle behaviour
- Verification and Synthesis
- Ptolemy II
- Summary
What does the paper talk about?

- * charts: Advocates decoupling the concurrency model from the hierarchical Finite State Machine (FSM) semantics
- FSM semantics:
  - Pure FSM and Hierarchical FSMs
- Semantics of combining FSMs with Models of Computation (MOCs)
Why are these ideas useful?

- Scalable approach to design
- Models will be *compositional* and support *heterogeneity*
- Side effect of heterogeneity: specialized MOCs become more useful
Some FSM definitions

- FSM = \{Q, \Sigma, \Delta, \sigma, q_0\}
- FSMs considered in this paper are deterministic and reactive
- Multiple Input Multiple Output (MIMOs):
  - \( \Sigma = \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_M \)
  - Input to the FSM consists of M signals
  - ith signal is a sequence of events from \( \Sigma_i \)
  - Multiple outputs defined in the same way
Pure FSMs

- Pure FSM

\[ a \land \neg b \rightarrow \alpha, \beta \]

\[ \neg a \lor b \rightarrow u, v \]

<table>
<thead>
<tr>
<th>Current State</th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>present</td>
<td>absent</td>
</tr>
<tr>
<td>(b)</td>
<td>absent</td>
<td>present</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Next State</th>
<th>(u)</th>
<th>(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u)</td>
<td>absent</td>
<td>present</td>
</tr>
<tr>
<td>(v)</td>
<td>present</td>
<td>present</td>
</tr>
</tbody>
</table>

\(\neg a \land b\)
Hierarchical FSM (HFSM)

- **Semantics:**
  - If the current state is not refined, HFSM behaves like a basic FSM.
  - If current state is refined, then *first* the corresponding slave reacts and then master reacts.
  - An output event is present if the action of the master or the slave FSM below it emits an event on that output.
  - No Collisions: An action does not explicitly emit the symbol for the absence of an event.

<table>
<thead>
<tr>
<th>Current State</th>
<th>(\alpha)</th>
<th>(\alpha)</th>
<th>(\beta,\gamma)</th>
<th>(\beta,\delta)</th>
<th>(\alpha)</th>
<th>(\ldots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>present</td>
<td>absent</td>
<td>present</td>
<td>absent</td>
<td>absent</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(b)</td>
<td>absent</td>
<td>absent</td>
<td>absent</td>
<td>present</td>
<td>absent</td>
<td>(\ldots)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Next State</th>
<th>(\alpha)</th>
<th>(\beta,\gamma)</th>
<th>(\beta,\delta)</th>
<th>(\alpha)</th>
<th>(\beta,\gamma)</th>
<th>(\ldots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u)</td>
<td>absent</td>
<td>present</td>
<td>present</td>
<td>absent</td>
<td>present</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(v)</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>(\ldots)</td>
</tr>
</tbody>
</table>
Combining HFSMs with MOCs

• Goals
  – A HFSM can be embedded as a module within an MOC
  – A state in an FSM can be refined to an MOC

• Termination
  – MOCs may not terminate, however the reaction of an FSM will need a finite time
  – Get around this difficulty by defining iteration of an MOC:
    • Divide execution of a non terminating system into a set of iterations. Each iteration can be associated with the reaction of an FSM
Synchronous Data Flow (SDF)

• Preliminaries
  
  – Standard definitions of DFs: actor, firings and tokens
  
  – Why SDFs and not general DFs?
  
  – Iteration: seek a finite solution to the balance equation for each arc: \( r_i p_i = r_j c_j \)
  
  – Arc is assumed to go from actor i to actor j, \( p \) is the number of tokens produced, \( c \) is the number consumed and \( r \) is the number of firings
  
  – We will deal with \textit{homogeneous} first, then extend the idea to \textit{non} \textit{homogeneous} SDFs
HFSMs inside homogenous SDFs

- Semantics:
  - FSM is a slave and externally obeys SDF semantics: each input to the SDF actor provides a single data token to the FSM, which takes on values from FSM's input alphabet
  - Subtle behaviour: “absent” event is not a well defined testable condition in SDF. Incorporate the “absent” even by encoding presence and absence using boolean-valued tokens
HFSMs inside non homogenous SDFs

- Semantics:
  - Syntactic sugar: each token of a given input is differentiated by concatenating its occurrence to its name:
    - $a$ denotes the most recent token consumed from the input
    - $a1$ is syntactic sugar for the next most recent consumed
SDFs inside HFSMs

- Semantics:
  - If an SDF graph refines the state of an FSM, when that state is the current state, the next reaction will consist of one iteration of the SDF graph followed by the reaction of the FSM.

- If the slave SDF graph is homogenous, the model fits naturally with the pure FSM model we adopted. Because, on each iteration one token is consumed and one is produced.

- However, if slave SDF graph is not homogenous, semantics are more subtle:
Heterochronous Dataflow (HDFs)

- In order to overcome this difficulty, invent a new model of computation called heterochronous dataflow (HDF)

  - Semantics:

    - In an HDF, an actor has a finite number of type signatures, where each type signature specifies the number of tokens consumed and produced
    
    - When such an actor fires, a well-defined type signature is in effect, but type signatures are allowed to change between firings
    
    - So, when an HDF system starts execution, there is an initial type signature in effect for each actor. Use these type signatures to solve the balance equations and find an iteration. The FSM components do not change state during the iteration
    
    - After the iteration, a new type signature is in effect, the balance equations are solved again and a new iteration starts
HDF example
An Application: Value FSMs

- In a valued FSM, the input and output alphabets are factored into signal alphabets, but at least one of these signal alphabets has size greater than two (it might even be infinite).

- However, value FSMs can be simplified to pure FSMs with our heterogenous *chart models.

- Example
Discrete Events (DEs)

• Preliminaries
  – Concept of a global time value
  – An event carries both a value and a time stamp that indicates the time at which the event occurs
  – Mathematical properties of DEs based on constructing a metric space with the Cantor metric
HFSMs inside DEs

- **Semantics:**
  - An FSM that refines a DE actor fires when there is an event at one of its inputs and the event has the smallest time stamp of all the events in the queue.
  - If the event has a value, the value is made available to the FSM. Absence of an event is represented in a DE with the absence of a token.
  - In order to accommodate the output of the FSM with a DE, we assume the FSM to be a zero-delay actor.
DEs inside HFSMs

• ASSUMPTION: Environment of an FSM is DE. Semantics are simple:
  - The FSM will react when any of the inputs is present. The input that triggers the firing will have as its time stamp of the current time of the environment
  - If the current state refines to a DE subsystem, then that subsystem will be simulated until the current time matches that of the environment. In the meantime, the events that it may emit will become outputs to the environment with time stamps equal to the current time
Subtle behaviour in DEs

Incomplete specification

Zero delay

- Removing incomplete specification is easy
- Number of solutions to removing zero-delay: intrinsic property of the DE
Synchronous/Reactive (SR) Systems

• Preliminaries
  – Models only the discrete steps, not only time continuum
  – Hence, notion of a clock tick
    • Execution of SR system occurs at a sequence of ticks
    • At each tick, signal either has no event (is absent) or has an event (is present, possibly with a value)
    • At each tick, signals are related by functions that have signals as arguments and define signals
    • At each tick, signals are defined by a set of simultaneous equations using these functions
    • A solution is called a fixed point and the task of a compiler is to generate code that will find such a fixed point
    • Functions are allowed to change between ticks. Hence, a module in SR has two distinct behaviours: produce and transition. In the produce phase, the current function is evaluated and in the transition phase, it is changed
Simple FSM inside SR System

- Semantics:
  - First, invoke the produce phase for each FSM however many times is needed to either define the outputs or reach a fixed point
  - If any signals remain undefined, signal a causality loop
  - Invoke the transition function of every FSM in the SR system
SR inside FSM

- Semantics:
  
  - If the current state of an FSM refines to an SR subsystem, then the semantics of the SR are simply exported to the boundary of the FSM
Synthesis and Verification

- Synthesis of hardware or software from FSMs, SDFs and SRs is standard. DE is used more for modeling, synthesis is not much of an issue

- Verification of FSMs, SDFs have been well studied

- However, synthesis and verification for the composition semantics in this paper is much more difficult

- Yet, the advantage of this approach is the decoupling of the MOCs from the FSMs. Hence, determinancy can be checked and this helps in validating the composite model through simulation
Ptolemy II

- The Ptolemy project studies modeling, simulation and design of concurrent, embedded real-time systems.
- They have developed a Java based GUI called Ptolemy II that can be used to study heterogeneous compositions of MOCs.
- Tool can be obtained from http://ptolemy.eecs.berkeley.edu
Summary

- Paper combined FSMs with MOCs
- Talked about semantics and subtle behaviour
- Looked at Ptolemy II environment