Domain Overview

- *Synchronous Dataflow (SDF)*: FourierSeries: In SDF, the firing of actors is statically scheduled, and at the start of execution, consistency and deadlock conditions are checked. SDF is suitable for applications with simple, data-independent control flow. Actors produce and consume a fixed number of tokens on each port on each firing. If these numbers result in either deadlock or unbounded buffers, then you will get an error message, as illustrated by the inconsistent example.

- *Dynamic Dataflow (DDF)*: In DDF, the firing of actors is dynamically scheduled. Each actor asserts its "firing rules," specifying the number of input tokens it requires on each input port in order to be fired next time. This can be used to make data-dependent control flow, mimicking if-then-else, do-while, and even recursion. More interestingly, dynamic dataflow models balance data-driven and demand-driven computation to get sophisticated data-dependent control flow, as illustrated for example in OrderedMerge. It can be combined with SDF to get the best of both worlds, as illustrated in RandomWalk. Unlike synchronous dataflow, the DDF model of computation is Turing complete, and questions of deadlock and boundedness are undecidable. Nonetheless, given a model that can execute without deadlock and with bounded buffers, our scheduler will execute it without deadlock and with bounded buffers.

- *Heterochronous Dataflow (HDF)*: HDF is an extension of synchronous dataflow (SDF) that permits dynamically changing production and consumption patterns without sacrificing static scheduling. In SDF, the production and consumption patterns of an actor are constant. In HDF they are allowed to change between iterations of the HDF schedule. Modal models can be used to change these patterns. Although HDF can express many data-dependent computations that cannot be represented by SDF, it is not Turing complete. Consequently, deadlock and boundedness remain decidable. The Fibonacci example uses this mechanism in a clever way to extract a Fibonacci sequence from a counting sequence.

- *Process Networks (PN)*: In PN, each actor is wrapped in a thread, and all actors execute concurrently. PN can be used whereever DDF would be used, but unlike DDF, it is harder to control the duration of a run. The OrderedMerge example illustrates a starvation strategy to get the model to stop executing. More interesting, PN can be used to achieve a NondeterministicMerge. In PN, actors can send data at any time, and it is queued until the recipient reads it. A sophisticated algorithm is used to ensure that queues remain bounded if this possible given the model.

- *Rendezvous*: The Rendezvous director also wraps each actor in a thread, but unlike PN, when a sending actor sends data, it blocks until the receiving actor is ready to receive it. This model of computation is useful for coordinating asynchronous actions, as in the Barrier example, or for modeling resource management, as in the ResourcePool example.

- *Discrete-Event (DE)*: In DE, communication between actors is via time-stamped events, and events are processed in chronological order. DE is useful for modeling timed systems, such as communication networks, computer hardware, business processes, etc. The QueueAndServer example gives a model of a simple queueing system. DE has been extended to explicitly model wireless communication, as illustrated in the WirelessSoundDetection example.

- *Continuous-Time (CT)*: In CT, communication between actors is via continuous-time signals. CT is useful for modeling ordinary differential equations, as illustrated in the Lorenz example, and hybrid systems, which combine CT with FSMs.

- *Synchronous/Reactive (SR)*: In SR, which is inspired by the synchronous languages Esterel, Lustre, and Signal, actors react instantaneously and simultaneously at ticks of a logical clock. The GuardedCount example illustrates the ability to use subclocks, in the manner of Lustre.

- *Finite State Machines (FSM)*: The FSM domain is used principally to define composite actors as modal models. A modal model is one whose behavior depends on its "mode" of operation. A modal model in Ptolemy II heterogeneously combines the *finite state machine* (FSM) domain combined hierarchically with other models. A state in the FSM represents a mode of operation, and can have a refinement that gives the behavior in that mode. The refinement can be another FSM or some other model using some other Ptolemy domain. The ModalModel example combines DE, FSM, and SDF to model a system where regularly sampled signals are perturbed by irregular events in time.

- *Graphics (GR)*: The GR domain provides 3-D animation capabilities based on the Java3D API. The director optimizes the scheduling of actors so that graphics are re-generated only when changes have been made. See the Helen example.