Chapter 10. SR domain

Authors: *Stephen Edwards*

Other Contributors: *Christopher Hylands*
Mary Stewart

10.1 Introduction

The Synchronous Reactive domain is a statically-scheduled simulation domain in Ptolemy designed for concurrent, control-dominated systems. To allow precise control over timing, it adopts the synchronous model of time, which is logically equivalent to assuming that computation is instantaneous.

10.2 SR concepts

Time in the SR domain is a sequence of instants. In each instant, the system observes its inputs and computes its reaction to them. Each instant is assumed to take no time at all. All computation is treated as being instantaneous.

Communication in the SR domain takes place through unbuffered single driver, multiple receiver channels. In each instant, each channel may have a single event with a value, have no event, or be undefined, corresponding to the case where the system could not decide whether the channel had an event or not. Communication is instantaneous, meaning that if an event is emitted on a channel in a certain instant, every star connected to the channel will see the event in the same instant.

10.3 SR compared to other domains

SR is similar to existing Ptolemy domains, but differs from them in important ways. Like Synchronous Dataflow (SDF), it is statically scheduled and deterministic, but it does not have buffered communication or multi-rate behavior. SR is better for control-dominated systems that need control over when things happen relative to each other; SDF is better for data-dominated systems, especially those with multi-rate behavior.

SR also resembles the Discrete Event (DE) domain. Like DE, its communication channels transmit events, but unlike DE, it is deterministic, statically scheduled, and allows zero-delay feedback loops. DE is better for *modeling* the behavior of systems (i.e., to better understand their behavior), whereas SR is better for *specifying* a system's behavior (i.e., as a way to actually build it).

10.4 The semantics of SR

An SR star must be well-behaved in the following mathematical sense to make SR systems deterministic. It must compute a monotonic function of its inputs, meaning that when it is presented with more-defined inputs, it must produce more-defined outputs. In particular, an output may only switch from undefined to either present or absent when one or more inputs do, but it may not change its value or become undefined.

The semantics of SR are defined as the least fixed point of the system, meaning the least-defined set of values on the communication channels that is consistent with all the stars' functions. That is, if any star were evaluated, it will not want to change its output---the value is already correct. The monotonicity constraint on the stars ensures that there is always exactly one least-defined set, and this is what the SR schedulers calculate.

There are two schedulers for the SR domain, **`default-SR`** and **`dynamic-SR`**. The dynamic scheduler is the easiest to understand. In each instant, it first initializes all the communication channels to "undefined" and then executes all the stars in the system until none of them try to change their outputs. The default scheduler is more shrewd. It uses the communication structure of the system to determine an execution order for the stars that will make them converge. This is based on a topological sort of the stars, but is made more complicated when there are feedback loops.

10.5 Overview of SR stars

The top-level palette is shown in figure 10-1.

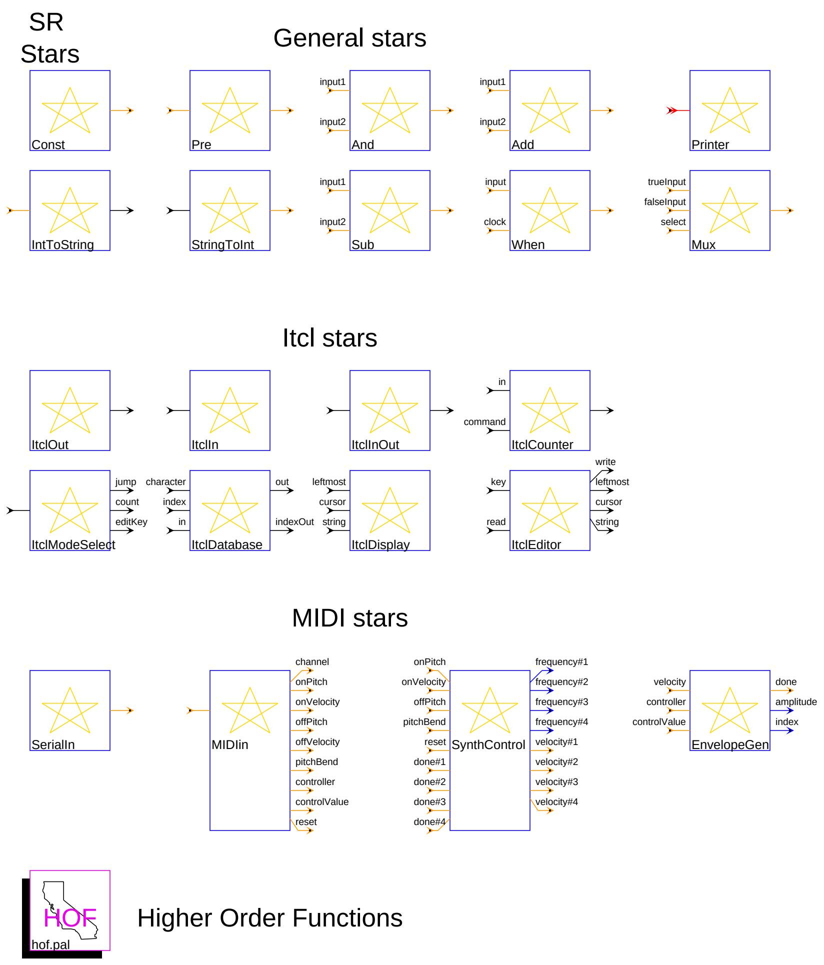

FIGURE 10-1: The top-level palette for SR demos.

10.5.1 General stars

`Const`	Output a constantly-present integer output given by the *level* parameter.
`Pre`	Emit the value of the integer input from the most recent instant in which it was present.
`And`	Emits the logical AND of the two integer inputs, or absent if both inputs are absent.
`Add`	Emit the sum of the two integer inputs, or the value of the present input if the other is absent.

`Printer`	Print the value of each input to the file specified by the *fileName* state. All inputs are printed on a single line with the prefix in the *prefix* state.
`IntToString`	Convert the integer input to a string.
`StringToInt`	Convert the string input to an integer.
`Sub`	Emits the different of the two integer inputs, or absent if both inputs are absent.
`When`	When the clock is true, emit the input on the output, otherwise, leave the clock absent.
`Mux`	Depending on the value of the select input, copy either the true input or the false input to the output.

10.5.2 Itcl stars

By default, all of these stars have no behavior. They provide an interface for user-written Itcl scripts that specify their behavior. Each has the following states: *itclClassName*, the class name of the itcl object associated with the star, *itclSourceFile*, the path name of the itcl script containing the definition of this class, and *itclObjectName*, the name of the itcl object to create. If this field is left blank, the object is given the name of the star instance. See the SR chapter in the Programmer's Manual for more information.

`ItclOut`	Single output Itcl star.
`ItclIn`	Single input Itcl star.
`ItclInOut`	One input, one output Itcl star.
`ItclCounter`	Itcl incrementer/decrementer.
`ItclModeSelect`	Itcl star used in the Rolodex demo.
`ItclDatabase`	Simple sorted database.
`ItclDisplay`	Display for the rolodex.
`ItclEditor`	Editor for the rolodex

10.5.3 MIDI stars

The MIDI stars below are used in the MIDISynthesizer demo described later in this chapter. We include these stars only as an example of what can be done with the Synchronous Reactive domain.

`SerialIn`	Emit the character waiting on the serial port, or leave the output absent if there isn't one. The *deviceName* state specifies the port, and *baudRate* specifies the speed.
`MIDIin`	An interpreter for the MIDI protocol. It takes an incoming MIDI stream and fans it out to Note On and Note Off commands.
`SynthControl`	A polyphonic synthesizer control.

`EnvelopeGen`	An envelope generator for FM sound synthesis.

10.6 An overview of SR demos

There are currently three SR demos. The palette is shown in figure.

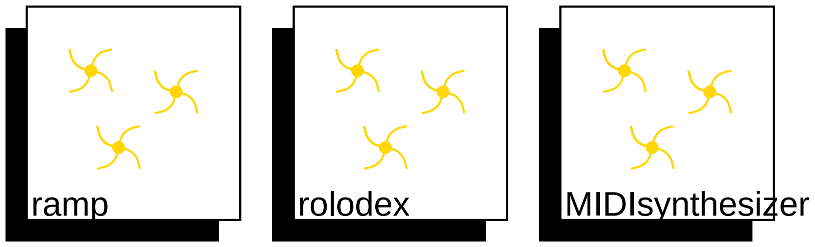

FIGURE 10-2: Synchronous Reactive demos.

`ramp`	Prints a sequence of increasing integers. Essentially a “hello world” for the SR domain, this demonstrates how Pre can interact with an adder.
`rolodex`	A digital address book implemented in SR. This demonstrates how itcl stars can be used to prototype user-interface-dominated systems at a high level. The system is divided into keyboard, database, editor, and display blocks.
`MIDIsynthesizer`	A music synthesizer written in SR. This is a polyphonic sound synthesizer written using custom SR stars for the control portion. Waveform synthesis is done using an FM algorithm implemented in CGC. This requires a MIDI keyboard to be attached to `/dev/ttya`, and functioning CGC audio drivers for your platform. More information on demonstrating the Midi stars with a Yamaha keyboard controller follows.

10.6.1 Use of the Yamaha CBX-K1XG as a midi keyboard controller

Setting Up Hardware Connections:

- The keyboard has an external power supply. Connect one end of the power supply to the DC IN jack on the rear panel of the keyboard and connect the other to a suitable electrical outlet.
- Connect the `TO HOST` Terminal of the keyboard to the serial port of the Sparcstation using an 8-pin Mini-DIN to D-Sub 25-pin cable. You will probably need to use a null modem with this connection.
- Select `PC2` (38,400 baud) with the `Host Select Switch`, located on the rear panel of the midi keyboard.
- You may need to use a Y cable if the Sparcstation to which you are connecting has more than one port (A-B).

*Note that it is necessary to have functioning CGC audio drivers in order to demo the SR stars written for a keyboard controller.

Operation of the Yamaha keyboard controller

A sequence of keys specific to Yamaha’s controller architecture must be triggered to send midi messages to the SR stars to generate synthesized sound within Ptolemy. See the Yamaha user’s manual for detailed instructions