The architecture of Sova

Abstract. Sova is a live coding tool that has been originally developed by Tanguy Dubois, Raphaël Forment and Loïg Jezequel from an idea discussed between Raphaël, Loïg and Rémi Georges[1]. This document presents the design of Sova: not necessarily what exists in the current state of the tool, but what we are aiming for. It should be used as a guideline for developers wanting to contribute to Sova.

1 A bit of history

In winter 2024, Raphaël and Rémi have been invited by Athénor CNCM to meet researchers from the LS2N in Nantes. One of the goals of this meeting was to investigate the possibility to propose activities around the concept of live coding for schools in Loire-Atlantique. After discussions, Raphaël, Rémi, and Loïg agreed that it would be interesting to let the students of the schools to design their own live coding languages. This would require to develop a new live coding tool that would allow to quickly implement new languages from the ideas of the students. A few months later, this new tool would become Sova.

2 The general modular design

The general idea of Sova is to have a very modular design (Figure 1), so that any part of it could be easily replaced: the graphical interface, the sound engines, the scheduler, etc, and, obviously, the live coding languages. In this section we briefly describe the role of each part of Sova.

2.1 The scheduler

The scheduler is the central part of Sova. It is responsible for executing scripts (written in various languages, as long as there exists an interpreter or a compiler — Section 2.5 — for these languages) a little bit ahead of real time and to send the events generated by these scripts to the world (Section 2.2) alongside with timestamps stating exactly when these events should occur. How and when scripts are executed is described in details in Section 3.

2.2 The world

The world is the part of Sova responsible for ensuring that each event sent from the scheduler is executed at the correct time. It handles two kinds of events:

- immediate events (such as midi events) that are sent to corresponding devices when the time corresponding to their timestamp is met;
- timed events (such as events for SuperDirt) that are sent to corresponding devices a bit ahead of the date given by their timestamp so that this device can handle them at the desired time.

[1]This original development has been supported by Athénor CNCM and the Laboratoire des sciences du numérique de Nantes (LS2N).

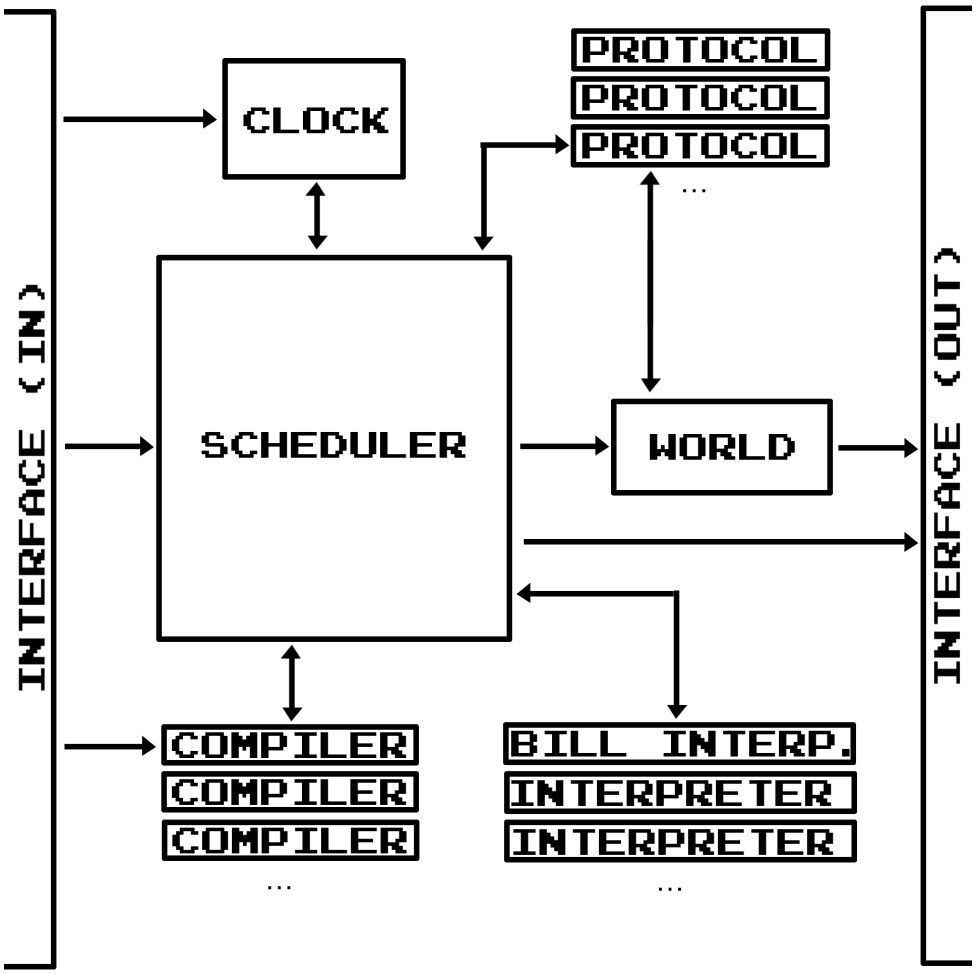

Figure 1: The modular design of Sova

2.3 The clock

The clock handles the time in Sova. The scheduler uses it to build timestamps and to ensure that it stays ahead of time. The world uses it to ensure that events are sent at the correct time. The clock is able to react to the environment (for example, it can synchronize with other clocks using Ableton Link).

2.4 The protocols

Each protocol (Midi, OSC, etc) that is usable with Sova has to be implemented. This allows the scheduler to be able to form correct messages to devices using any protocol. This also allows the world to know how to communicate with these devices.

2.5 The interpreters and the compilers

Live coding languages used in Sova are either interpreted or compiled. To be able to add a new language one can build an interpreter for this language[2]. When the scheduler needs to execute a script in this language it will call this interpreter. Alternatively, there exists an interpreter for a language called BILL that is not meant to be used as a live coding language but rather as an intermediate language. One can add a new language to Sova by building a compiler for this language to the BILL language[3].

[2]The way to do so is described in another document called *TODO*

[3]The way to do so is described in another document called *Custom Scripting Languages for Sova.*

2.6 The interface

The interface is used by the scheduler and the world to communicate with other tools and devices: graphical interfaces, sound engines, etc.

3 The scheduler of Sova in more details

Most of the parts of Sova are only technical things (clock, protocols, world, interface) — that must look like what can be found in other live coding tools — or are described in other document because they should be changed by the users (interpreters, compilers). In this document we thus focus on the scheduler of Sova.

3.1 Frames, lines, scenes (and scripts)

The scheduler maintains a collection of scenes, each of theme being constituted of lines and each line being a collection of frames. Moreover, each frame is associated to a script. The scheduler executes one scene at a time (it is possible to switch between scenes but we focus here on the execution of one scene). The execution of such a scene consists in the concurrent execution of all its lines. The execution of a line consists in the sequential execution of all its frames. Finally, executing a frame consists in starting the execution of its script.

3.2 Durations

Each scene, line and frame has a duration.

Depending of the mode in which the scheduler is used, the durations of the lines of a scene can be absolute or can depend on the duration of this scene.

When the the time from the start of the execution of the current scene reaches its duration, the scene restarts from the beginning. In this case the executions of all the lines of this scene will restart from the beginning.

When the time from the start of the execution of a line reaches its duration the line can either restart from the beginning or just stop, depending of the mode in which the scheduler has been set.

Finally, the durations of the frames of a line indicate when these frames should execute in the line: when the time from the start of the execution of a frame reaches its duration, the next frame starts. It is possible that the sum of the durations of the frames of a line is different from the duration of this line. In this case, depending of how the scheduler is configured, these duration can be stretched to correspond to the duration of the line, or they can be left as is (leading to either no frame execution at the end of the line or non-executed frames after the end of the line).

3.3 Scripts

A script is a program that can be interpreted by one of the interpreters of Sova. When a user submits a script through the interface this script is associated to exactly one frame (and thus to one line and one scene)[4].

Once the execution of a script starts, this script is executed until it reaches its end. In other words, if the script is an infinite loop it is executed forever. All the running scripts are executed concurrently by the scheduler, following a simple scheduling algorithme.

The scheduler maintains a list-like data structure indicating all the scripts that are currently running. A running script can be either active or idle. Looping on the list, the scheduler:

[4]The document *TODO* describes the TCP interface of Sova, it tells how to submit a script and how to decide to which frame this script shall be associated.

- gives the opportunity to each active script to execute one computation step. This means that the scheduler calls once the interpreter corresponding to this script[5]. From the result of this computation steps the scheduler decides if:
  - an event should be emitted (in which case the adapted protocol is used to build a message, a timestamp is computed from the clock, and the result is sent to the world),
  - the script is terminated (in which case it is removed from the list),
  - the script becomes idle for a given time (in which case the starting point of this idle time is recorded);
- checks if each idle script has finished its idle time, in which case the corresponding script becomes active.

4 Current state of the development

As of July 2025.

4.1 Scheduler

The scheduler can use our interpreter for the BILL language. Only one mode is available: lines restart at the end of the duration of a scene only.

4.2 World

The world is fully functional.

4.3 Clock

Ableton Link protocol is handled allowing our clock to synchronize with any clock implementing this protocol. Midi clocks are not yet handled.

4.4 Protocols

We implemented usable versions of Midi, OSC, SuperDirt, the protocol for our own sound engine.

4.5 Interpreters

It is not yet possible to include custom interpreters: our BILL interpreter is hardcoded in the scheduler.

4.6 Compilers

As a proof of concept we built a compiler for a language called BaLi[6]

4.7 Interface

There is a TCP interface that allowed us to develop a TUI for live coding with Sova.

[5] What is a computation step for the particular interpreter of our internal language is described in the document *Custom Scripting Languages for Sova.*

[6] See the document *The Grammar of BaLi.*