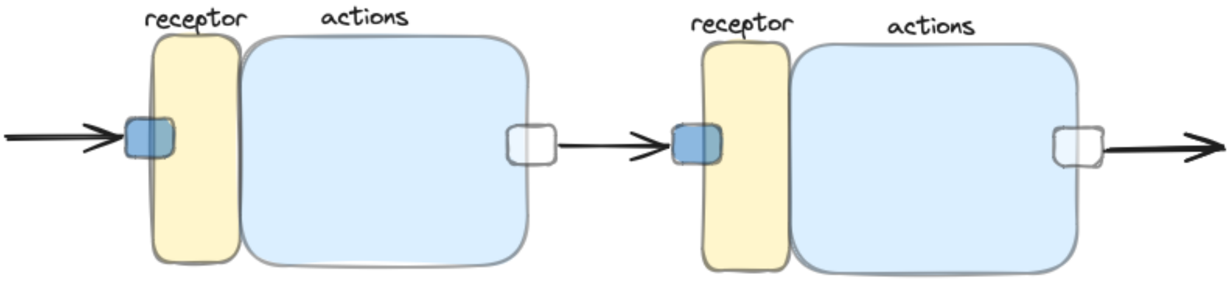
2024-05-12-Free Range Programming

SWIBs and Receptors

SWIBs - SoftWare Interlocking Blocks.

The general gist of SWIBs is that of forming pipeline, as show in the figure below



A *receptor* is essentially a stream-based *parser.* A receptor accepts incoming Unicode characters one-by-one and executes *actions* based on the path specified by the parse. If the incoming characters don't match any pattern specified in the .swib file, a *syntax error* is displayed and the receptor stops, or, tries to continue parsing the rest of the input. In either case, no further messages are sent on the output pin. This is not shown on the diagram above for sake of simplicity. Each *swib* has at least one[1] extra output pin which corresponds to some kind of error message(s).

Receptors are very similar to *parsers,* except that receptors work on streaming input instead of slurping a whole file in and making a parse tree. Receptors don't create internal parse trees by default, and, simply call *action* code during the parsing operation. This allows *receptors* to handle inputs (and files) of any size, but, does not automatically allow for undoing or reversing the flow of operations, unless care is taken in the design of the .swib file and the actions that it calls.

An analogy for receptor-based *swibs* is that of RNA in biology. RNA walks down the strands of DNA and duplicates the DNA strand as the walk proceeds. A swib

[1] SWIBs can have many outputs - and, many inputs. You are not constrained to having only one input and two outputs. In practical projects, there might be many *happy paths* through the code and each path may wish to produce outputs and consume inputs on customized ports. Ports can be named, for exactly this kind of usage. The most common ports usually have empty strings as their names (this is like *stdin* and *stdout* in UNIX®). We'll skip over this kind of nuance in this essay, to keep things simple.

receptor does a similar walk, but is not restricted to making a duplicate of its input - it simply invokes *actions* as the walk proceeds.

Notes to self

- FoPoC[2] developers use only function-based text and think inside the box. FoC[3] developers use many notations and think outside the box.
- goal: show one way to create new notations
- goal: use existing tools
- Code: a way to create scripts to program actions of electronic machines
- CPU is a script sequencer. A sequencing chip attached to storage.
- FOPOC finding a way to program cpu based sequencer only.
- Definition of computer science: digitized mathematics
- Demo 1: show sequential echo boxes.
- Demo 2 edit sequential demo to parallelize boxes
- Double 3: show XML of parallel boxes ; show JSON from XML ; discuss code to evaluate JSON
- PROLOG: the demonstration of a programming language in a declarative style ; uses engine to decide on implementation of details
- S/SL - Syntax / Semantic Language - demonstration of typeless language ; types can be declared, but not defined
- S/SL - demonstration of the power of a typeless language. Can implement a whole compiler in S/SL, e.g. PT Pascal. Logic is declared in S/SL, then we use object oriented techniques to modularize implementation.
- MiniKaren: demonstration of exhaustive search without specifying implementation.
- Production Engineering: implementation of types, implementation of code.

[2] FoPoC - Future of Past of Coding

[3] Future of Coding

- Choosing algorithms is Production Engineering.

- Question: is DPL (Diagrammatic Programming Language) declarative or Production Engineering?

- There is a difference between Engineering and Production Engineering.

- SWIB style thinking outside of the box: example: pipeline of components to implement Dungeon Crawler game, used something like 15 stages (project is unfinished)

- Focus follows mouse issue: major drawback of MacOS interface. Scrolling focus follows mouse, but you have to click (a second time) to have keyboard follow mouse. You can mouse over to a window and scroll it, but, to type in it, you have to click on the window first. Schizophrenic behaviour.

Appendix - See Also

See Also

References *https://guitarvydas.github.io/2024/01/06/References.html*
Blog *https://guitarvydas.github.io/*
Blog *https://publish.obsidian.md/programmingsimplicity*
Videos *https://www.youtube.com/@programmingsimplicity2980*
[see playlist "programming simplicity"]
Discord *https://discord.gg/Jjx62ypR (Everyone welcome to join)*
X *(Twitter) @paul_tarvydas*
More writing *(WIP): https://leanpub.com/u/paul-tarvydas*