Kicad tool framework technicalities

Tomasz Włostowski/CERN, August 2013

Goals:

- Have a separate class for each tool and no need to modify anything in the core application to add new tools,
- Enforce MVC model,
- Eliminate static/global tool states,
- Independence from the GUI toolkit and its quirks on different systems,
- Tools as plug-ins.
- Pcbnew as the guinea pig

Proposal:

1. There are two tool types:
   - batch tools, that are invoked once and do their job without user's intervention (for example, the DRC checker)
   - interactive tools, that serve user's input in a sequential manner (wait for an event, do something, wait for another event, do something else, and so on). We'll focus here only on interactive tools.

2. Interactive tools are state machines. Their states are either explicitly defined as separate callbacks, as or implicitly, using coroutines and Wait() statements.

3. All tools are kept by an application-wide TOOL_MANAGER object. It doesn't care about the application type (same for pcbnew, eeschema, etc.). TOOL_MANAGER takes an event from a TOOL_DISPATCHER object and forwards it to the tools that have requested the particular event type. The manager also deals with context menus.

4. TOOL_DISPATCHER provides a translation layer between the wx UI and the internal event system. It fixes all wx quirks (no mouse warping on OSX or weird focus handling under Linux) and translates between the screen and world (board) coordinate systems. It also generates motion events when the view is auto-scrolling (cursor close to the border).

5. Coroutine and Callback-based approaches can be mixed to create large, hierarchical tool FSMs.

6. One tool can invoke another (passing some tool-specific parameters) and fetch its return value. For example, *a Zoom* tool might want to call *a Rectangular Selection* tool to determine the region to be zoomed.

7. Each tool can define its own independent context menu through the CONTEXT_MENU object and launch it automatically (on right click) or whenever it wants to (clarification menu).

8. Common actions (Copy, Move, etc) are passed to the menus and tools via a TOOL_ACTION class . TOOL_ACTION also provides human-readable names, icons, etc. **TO BE DEFINED.**

Overall design:

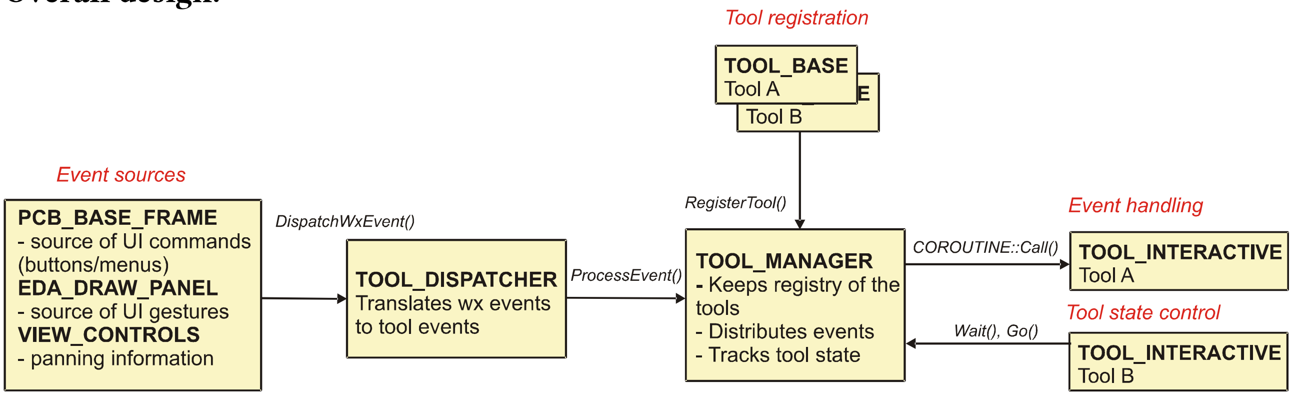

Figure 1: Architecture of the tool framework.

Classes:

TOOL_BASE:

- base registration interface for all tools.
- setting/getting setting tool name/ID/type.

TOOL_INTERACTIVE:

- base class for each interactive tool
- each tool is a state machine with either explicitly or implicitly defined states.
- states are C++ delegates (pointer to a member method of a particular object)
- tool can wait for an event without disturbing the application through a coroutine mechanism

TOOL_MANAGER:

- registry for all the tools for a given application.
- redirects events coming from the UI to tools that have requested them.
- handles context menus

TOOL_EVENT:

- generic tool event, wx-independent
- coordinates are world space
- standard ones: mouse, keyboard
- UI commands (forwarding wx command identifiers)
- context menu commands (mouse over selection & done selection)
- view feedback (zoom/pan changed)
- tool activate/cancel

TOOL_DISPATCHER:

- takes wx events,
- fixes all wx quirks (mouse warping, etc)
- translates coordinates to world space
- low-level input conditioning (drag/click threshold), updating mouse position during view auto-scroll/pan.

Ways to write a tool:

Approach 1: using explicitly defined state methods

- each state is a separate method
- state transitions are done through Go() statement.
- Go(State) with no conditions jumps immediately on next event
- Go(State, conditions) jumps when a matching event arrives

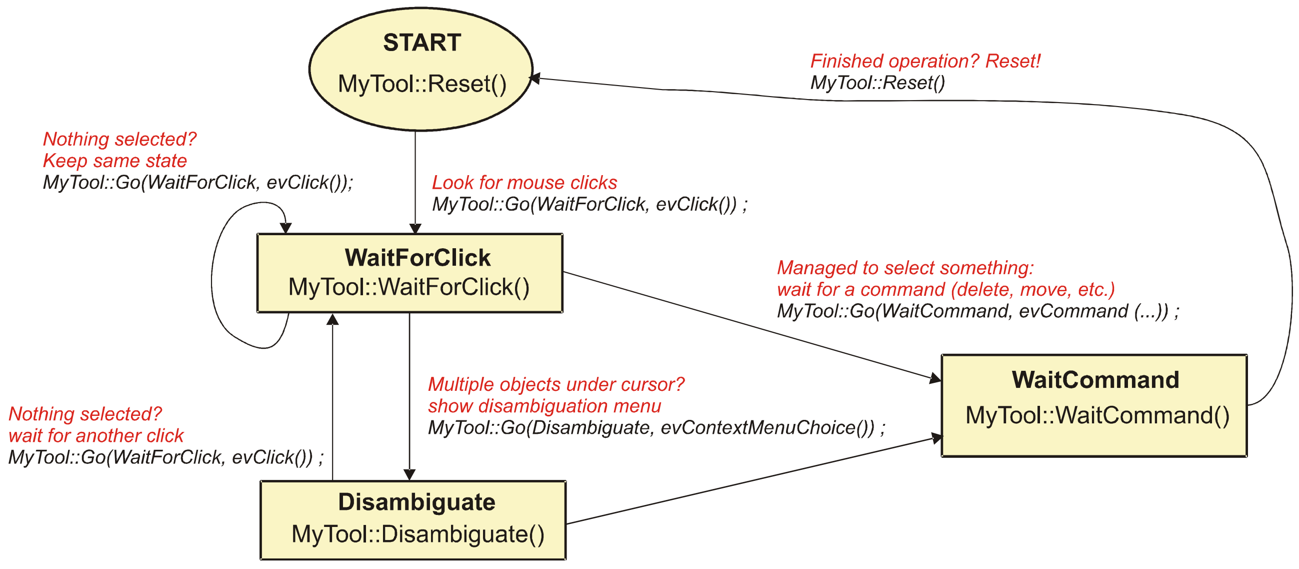
START
MyTool::Reset()
Finished operation? Reset!
MyTool::Reset()
Nothing selected?
Keep same state
MyTool::Go(WaitForClick, evClick());
Look for mouse clicks
MyTool::Go(WaitForClick, evClick()) ;
WaitForClick
MyTool::WaitForClick()
Managed to select something:
wait for a command (delete, move, etc.)
MyTool::Go(WaitCommand, evCommand (...)) ;
WaitCommand
MyTool::WaitCommand()
Multiple objects under cursor?
show disambiguation menu
MyTool::Go(Disambiguate, evContextMenuChoice()) ;
Nothing selected?
wait for another click
MyTool::Go(WaitForClick, evClick()) ;
Disambiguate
MyTool::Disambiguate()

Figure 2. A tool using explicitly defined states.

Approach 2: using coroutines

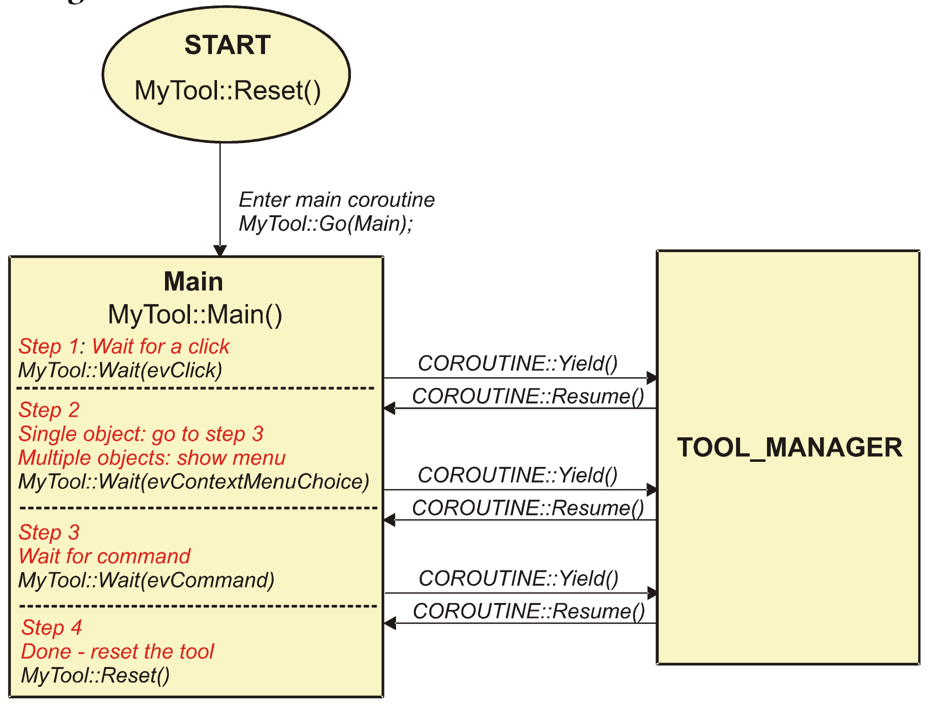

Figure 3. A tool using coroutines.

- single state method
- event request (Wait()) causes yielding the coroutine
- when a machine event arrives, the coroutine is woken up.

Approach 3: mix of the above

- good for large complex tools
- Wait() for linear sequences of sub-states
- Go() for the overall structure

Approach 4: state machine in a switch()

- if you really like it this way
- might be useful for quickly porting legacy tools.