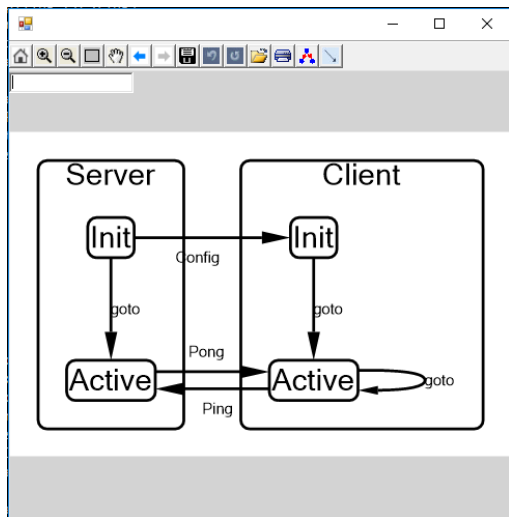


P# Program Visualization

The P# program visualization is designed to offer a high-level graphical view of the execution of a P# program. It observes a set of executions of the program and summarizes the information in a graph-based view. It is invoked by passing the flag `/visualize` to the P# tester. A typical command-line would look like the following:

```
PSharpTester.exe /test:file.dll /i:10 /visualize
```

A representation of the PingPong example (Samples\PSharpAsLibrary\PingPong) is shown below. There are two machines in the PingPong program, shown as the “Server” and “Client” boxes. States are nested nodes within the machines. Intra-machine state transitions are shown as arrows labelled with “goto” (or, in general, with the raised event that triggers a GotoEvent). When a machine M1 in state S1 sends an event e, and this event is dequeued by machine M2 while in state S2, then an arrow is drawn between S1 and S2 with label denoting the type of e.



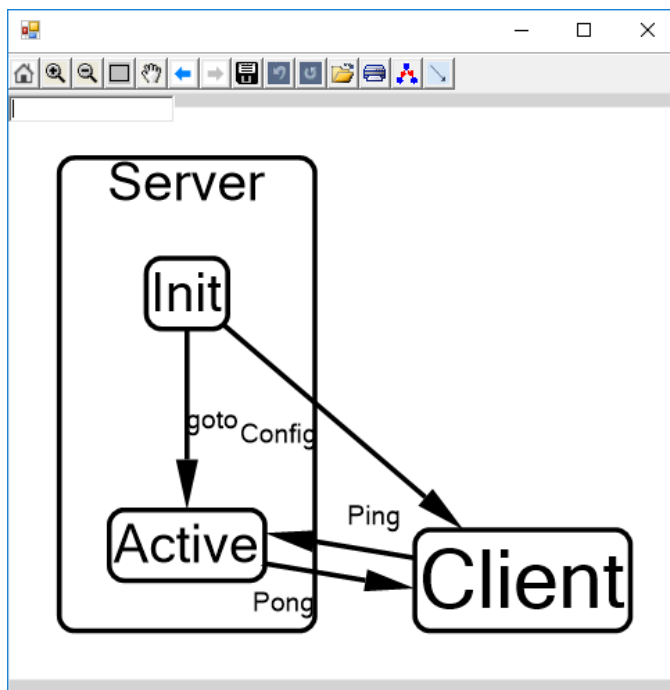
Note that the information required to draw transitions can only be obtained at runtime. Statically, it is not possible to know in what state do sent events get received.

The visualizer summarizes information across all instances of the same machine. Thus, if a program creates two machines of the same type M, the graphical view will merge both instances together into the same net of nodes. At this point, it is not possible to separate different instances, but we may provide that feature at a later point in time.

We anticipate that even with the summarization described above, the graphical view of a large program might contain too much information to digest. The visualizer allows user interaction. Users can create their own view of the program. The visualizer window includes a command box in the top-left corner. It accepts a single-line text command as input. The command can be one of the following:

```
collapse MachineName  
expand MachineName  
hide EventName  
unhide EventName
```

The command `collapse M` hides internal states of `M`. For example, “`collapse Client`” in the PingPong example will result in the following view.



The `expand` command undoes a previous `collapse`. The command `hide e` hides all transitions of event `e`, and `unhide` undoes a previous `hide`. We intend to make this interaction much richer in future releases, depending on user feedback.