



Co-evolution in State Machine Based Systems

December 13, 2007

Jan Vlegels

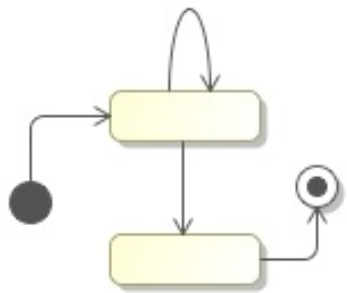


Design/Implementation Gap

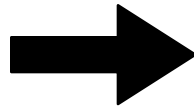
- Development process is not bijective function
 - Developer has own programming techniques
 - Design models don't detail everything
 - Modifying code without modifying design
 - ➔ Link between code and design unclear



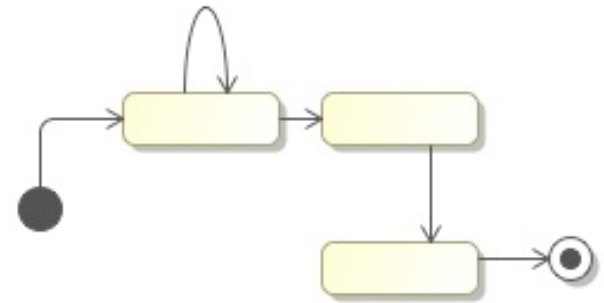
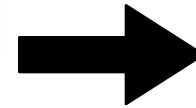
Problem: no co-evolution



**state diagram
design**



source code



**extracted state
diagram**



Industrial Case: Psi-Control

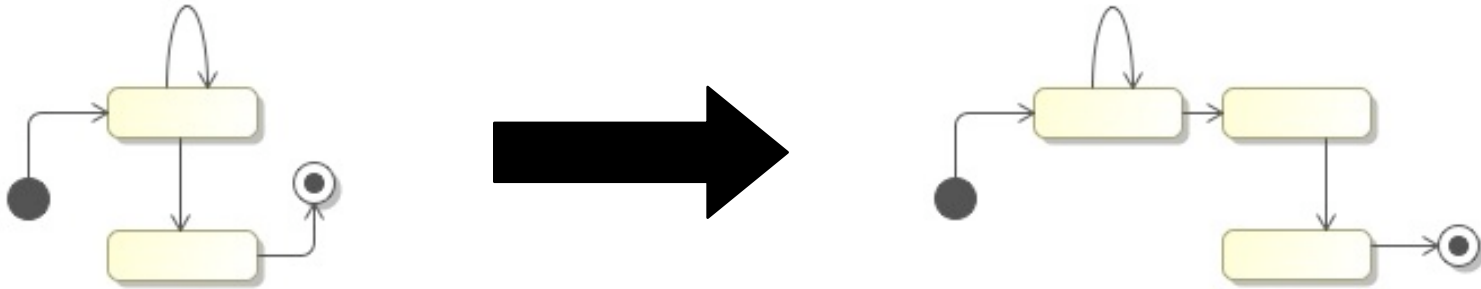
- Software for looms
- 100 state machines
- State diagrams are readable for everyone
- Automated code generation but not vice versa



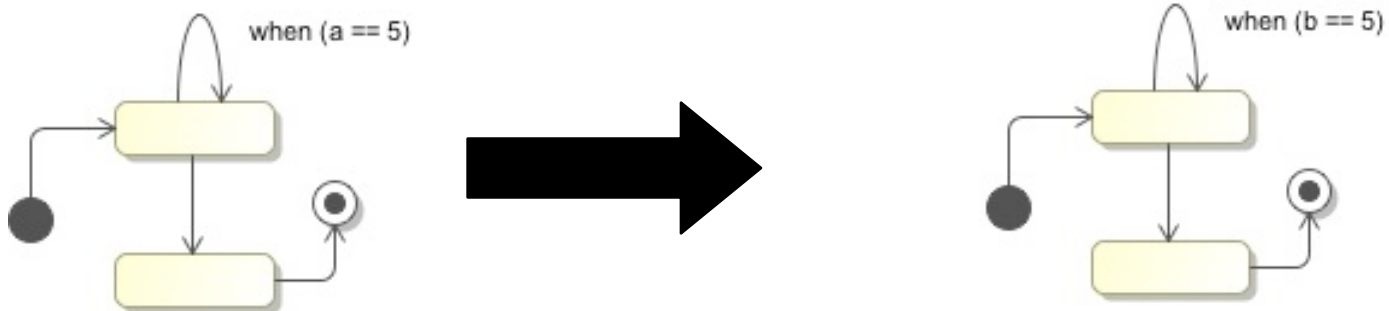


Examples of inconsistencies

- Added State



- Using different data



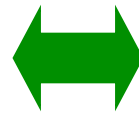
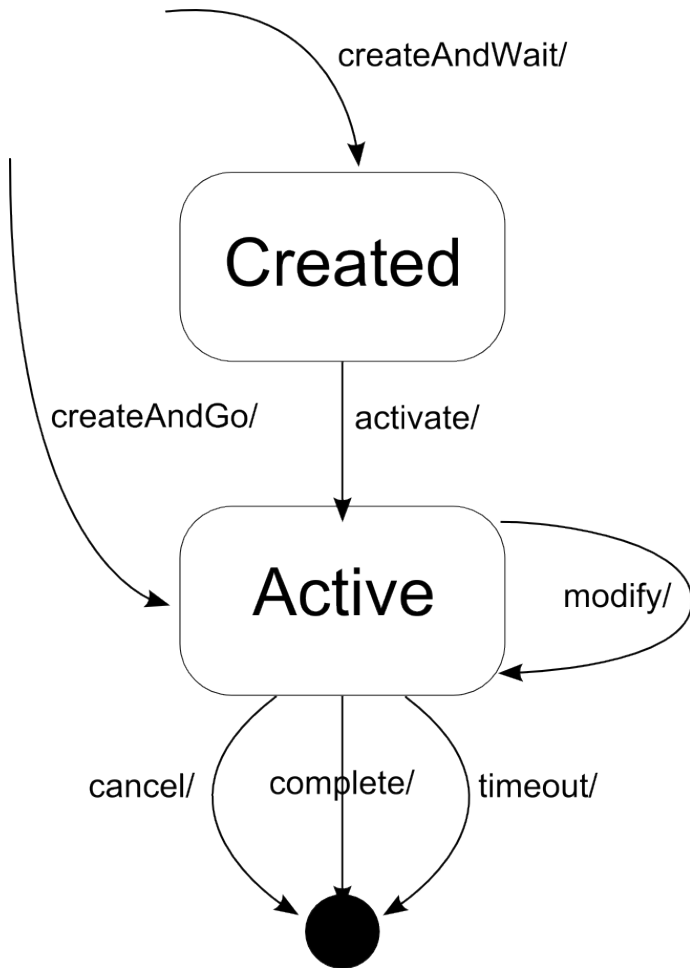


Recover states and transitions

- Keep the state machine explicit
- Refactor to an explicit state based system
- Synthesize state diagrams from code



Annotation example

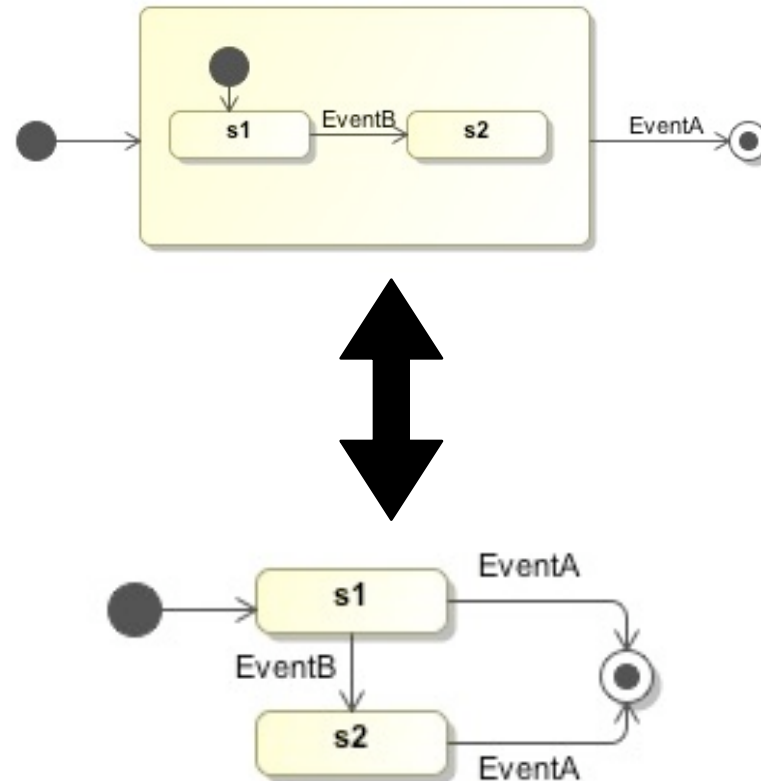


```
st = "" #@state
...
if st == "created":
    actCreated()
    st = "activate" #@trans cr
act
else if st == "active":
    actActive() #@action act
...
...
def actCreated(): #@action cr
```



Comparing state diagrams

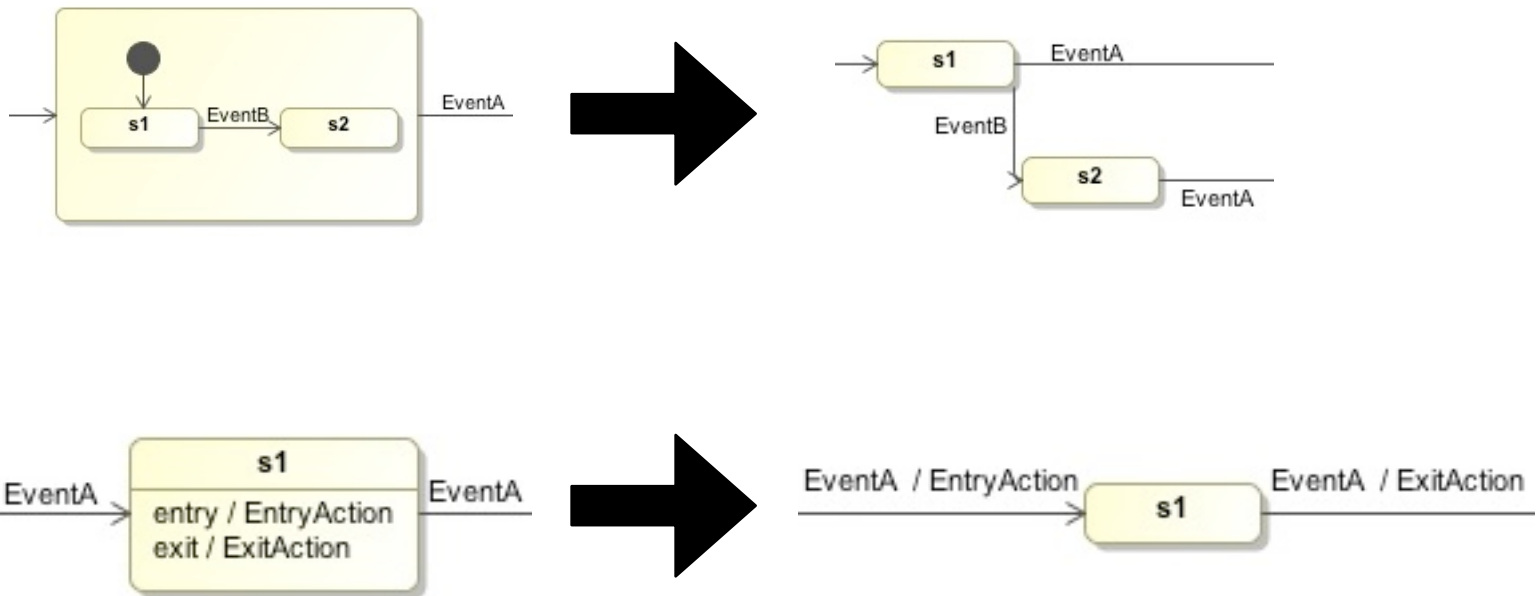
- Different state diagrams, same semantics





State diagram canonical form

- “Rewrite” rules





At the end: co-evolution

- Environment ensuring co-evolution
- Reflect changes in all artifacts
- States, transitions... as constraints on code