

Model Checking meets Agile Methods

Jim Hoover

Dept. of Computing Science, University of Alberta
Research support by NSERC, ASRA,
Avra Software Lab Inc, Curtiss-Wright Flow Controls

Cmput 401 Notes - 2005-02-01

Motivation

Building engineering design and audit tools in which errors in workflow can lead to safety critical problems in plants.

Users require some kind of V&V for the software.

Validation: have we built the right thing?

Is this how we want to process an order?

Verification: have we built the thing right?

Prove impossible to ship without payment.

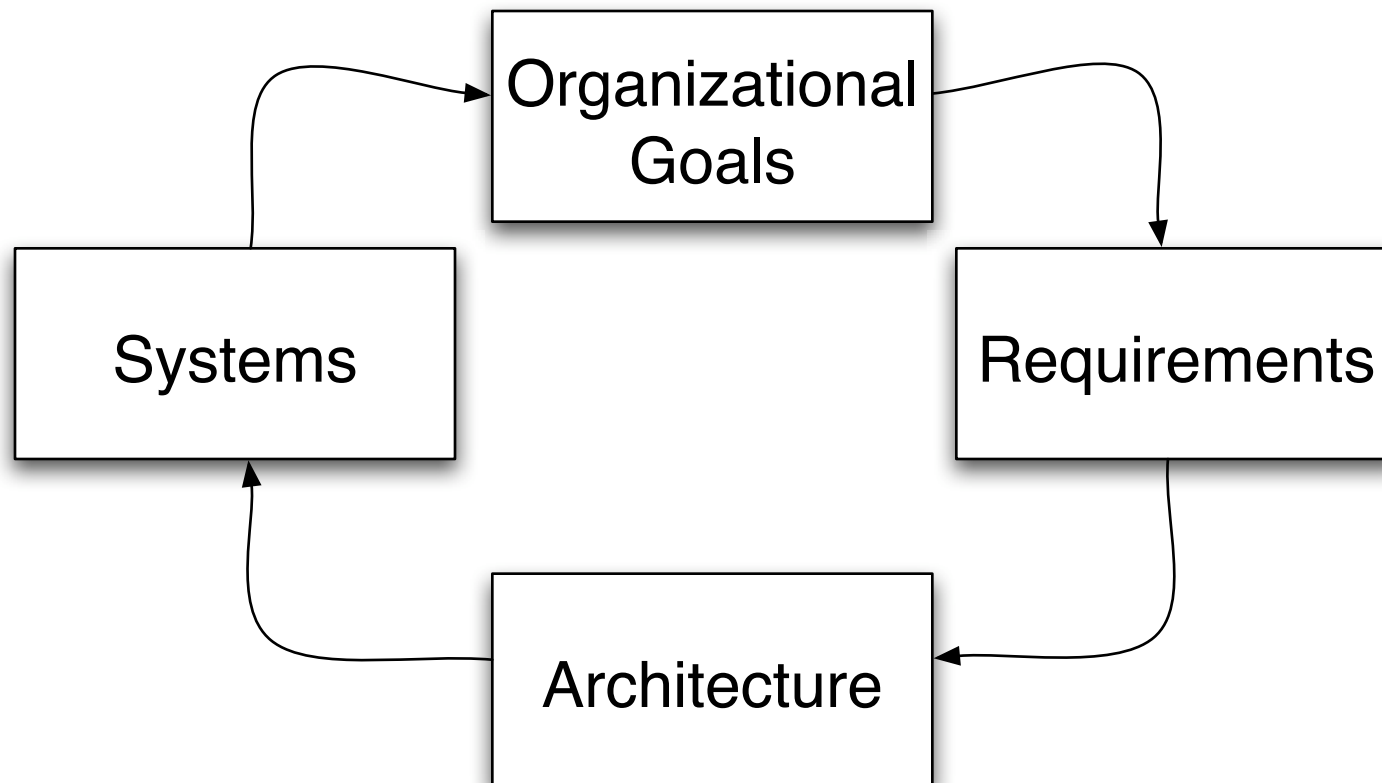
Weave together ...

- Agile Methods
- Frameworks
- Model Checking

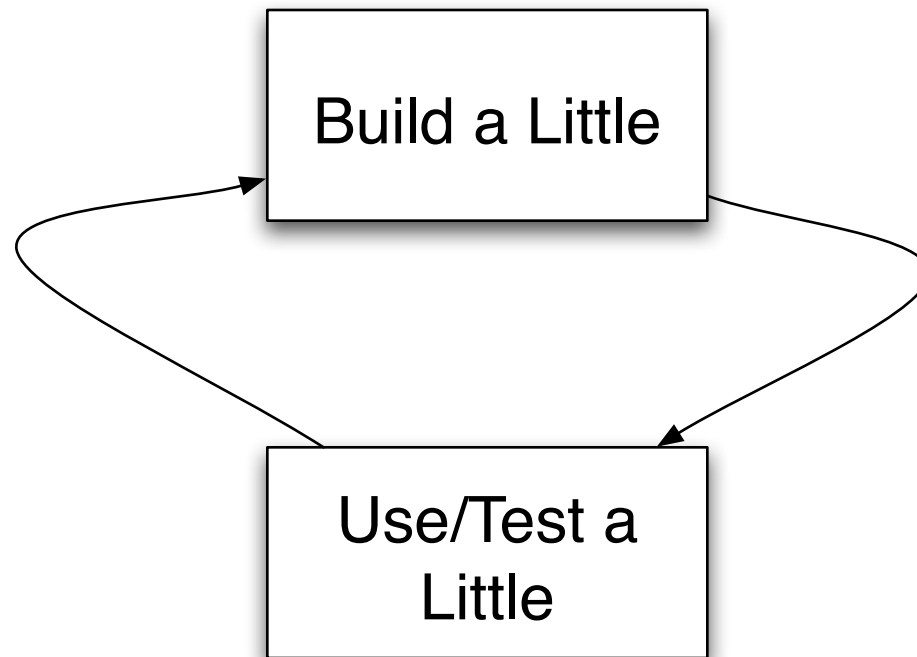
Talk Plan ...

- Agile Methods
- Test-driven Development
- Agile Framework
- Model Checking
- Integration
- Next Steps

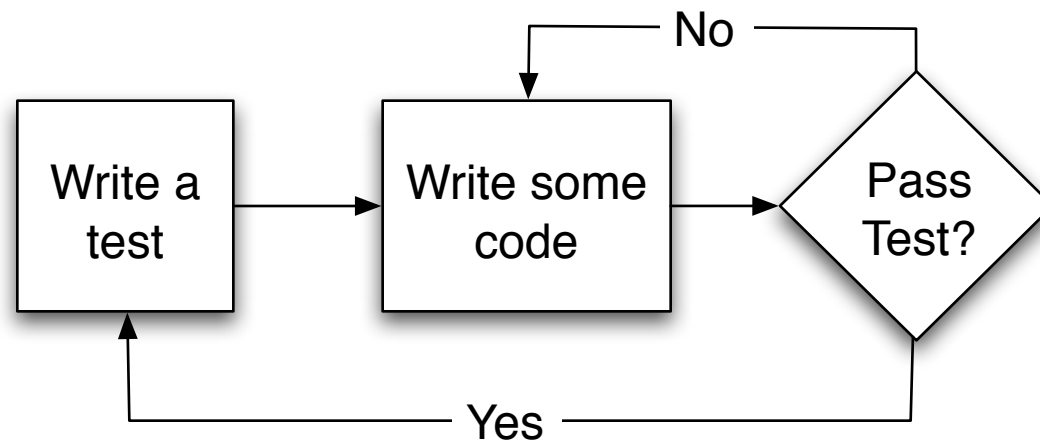
Len Bass' Architecture Business Cycle



ABC implies
System building is
iterative requirements gathering



Agile Test-Driven Development says:
Tests capture requirements.
Build to the tests (aka operational specifications).



Test-Driven Agile
Development is
good at validation.

Can we also make it
good at verification?

Both are hard unless we constrain our world

We do this via architecture.

Applications dominated by workflow
generally have these components ...

The so-called DB-centric reference
architecture

DB-centric reference architecture

Business objects stored in a DB

Forms to query for sets of business objects to manipulate.

Forms to modify the state of business objects.

Browse Customers

Existing Customers

Alpha Inc.
Beta Inc.
Gamma Inc.
Delta Inc.

New View Delete

View Customer

Name: Gamma Inc.
Status: active

Set Active	Set Inactive
------------	--------------

Browse Orders

Query: Gamma Inc.

Order 1
Order 10
Order 11
Order 20

New View Delete

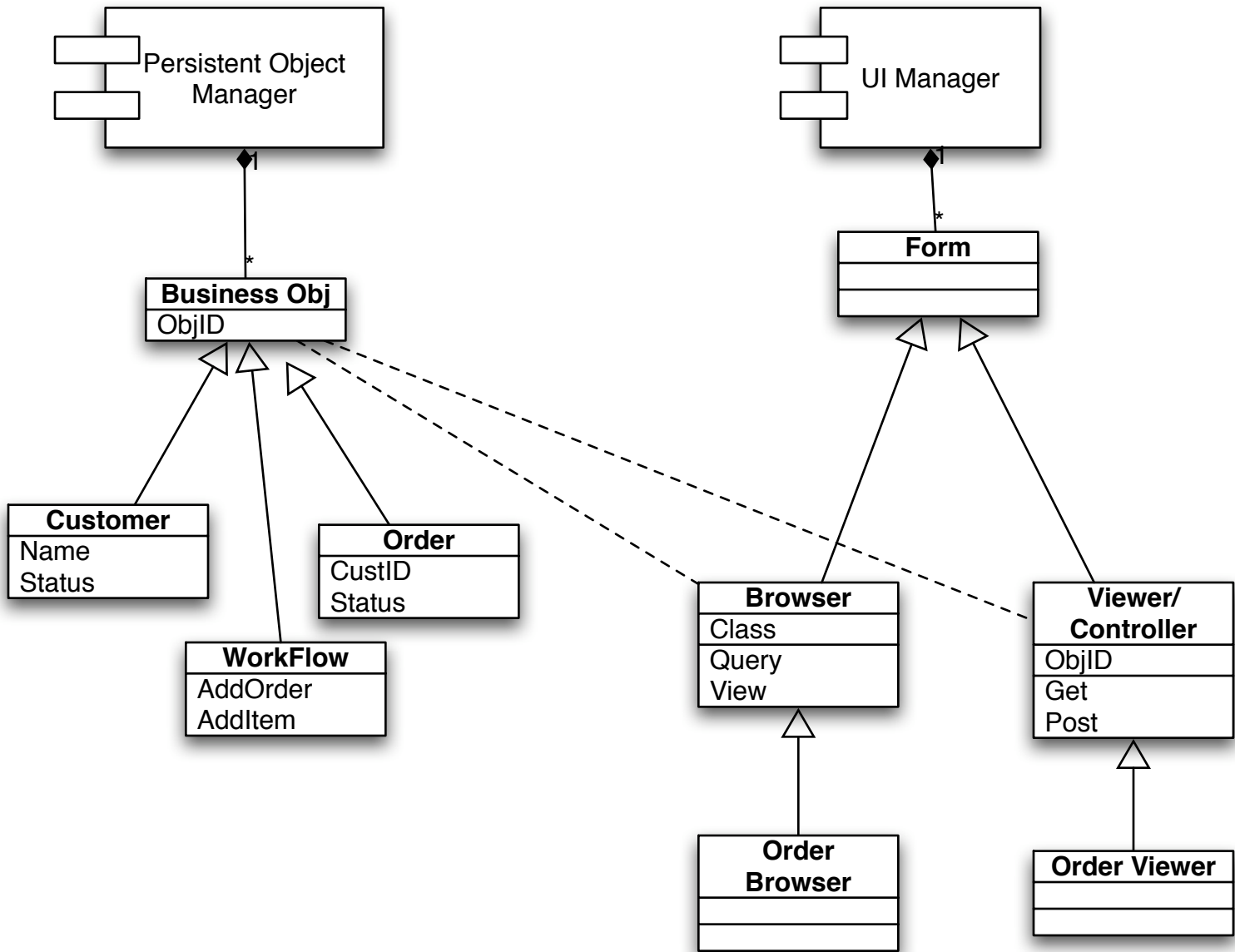
View Order 10

Customer: Gamma Inc.
Status: quoting

Order Items

Widget 1
Widget 2
Widget 3
Widget 4

Send to Quoting Leave Open Ship



Exploit this commonality with a framework.

Given the business object schema,
the Agile Framework can generate
a default application.

Use that to validate the business logic /
workflow.

Business Object Schema

Sequence of attributes

Attribute is a $\langle \text{name} \rangle \rightarrow \{ \text{properties} \}$

Attributes are typed:

scalars: strings, ints, etc.

references: obj ids of existing objects

methods: closures executed in context

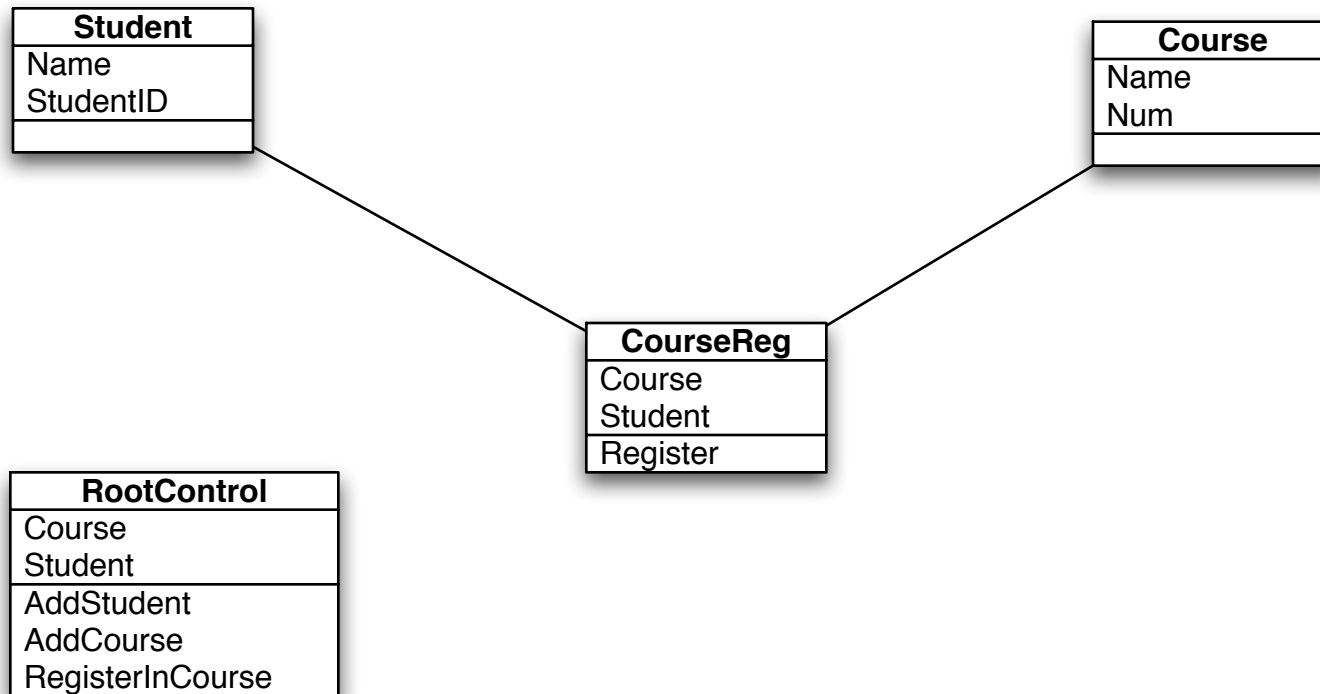
How?

attributes: input/output entry fields
when UIActive = 1

methods: action buttons on form
when UIActive = 1, Pre = 1

Quick Demo

CourseReg System



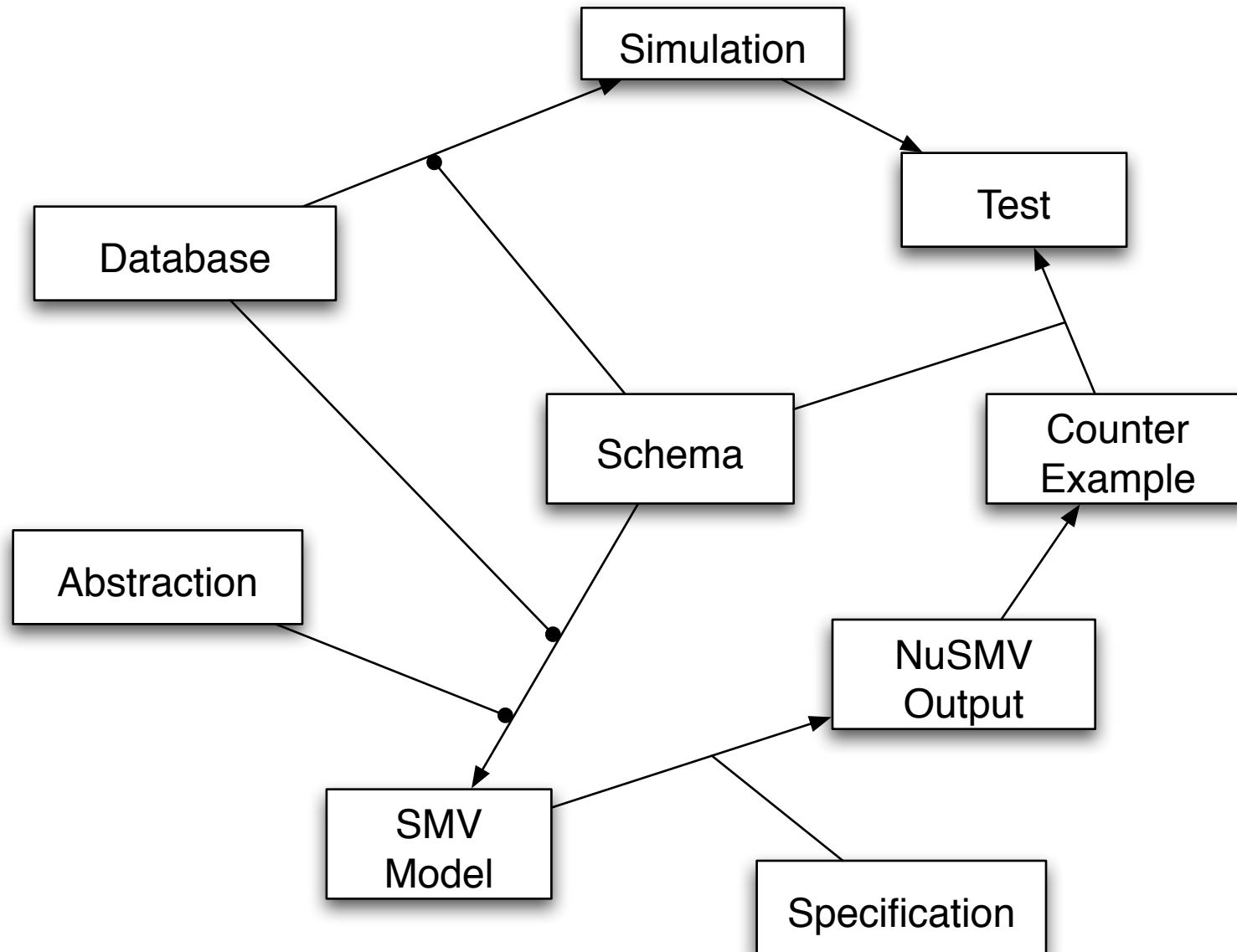
Agile Framework Goals

Goal 1: Write application procedurally

Goal 2: Generate a UI automatically so we can validate by using/testing.

Goal 3: Extract a precise formal version that we can verify.

The Big Picture



Computation Tree Logic

CTL is a:

temporal logic - lets us talk about future

branching time logic - lets us talk about
possible futures

Lets us talk about how a system can
evolve over time.

Model checking lets us check CTL expressions over finite systems.

! EF (
CourseReg3001ModLC = 1 &
CourseReg3001Student = Student1001 &
CourseReg3001Course = Course2001
&
CourseReg3002ModLC = 1 &
CourseReg3002Student = Student1001 &
CourseReg3002Course = Course2001)

Quick Demo

CourseReg Model Check

How did we get a model
from the code?

Need abstraction function:

- what attributes are important?
- what is their range?
- what are the state transitions?

Transitions come from

Input from active fields in UI.

Executions of active methods
in UI with enabled preconditions.

How do we extract these for the
model checker?

Model Extraction

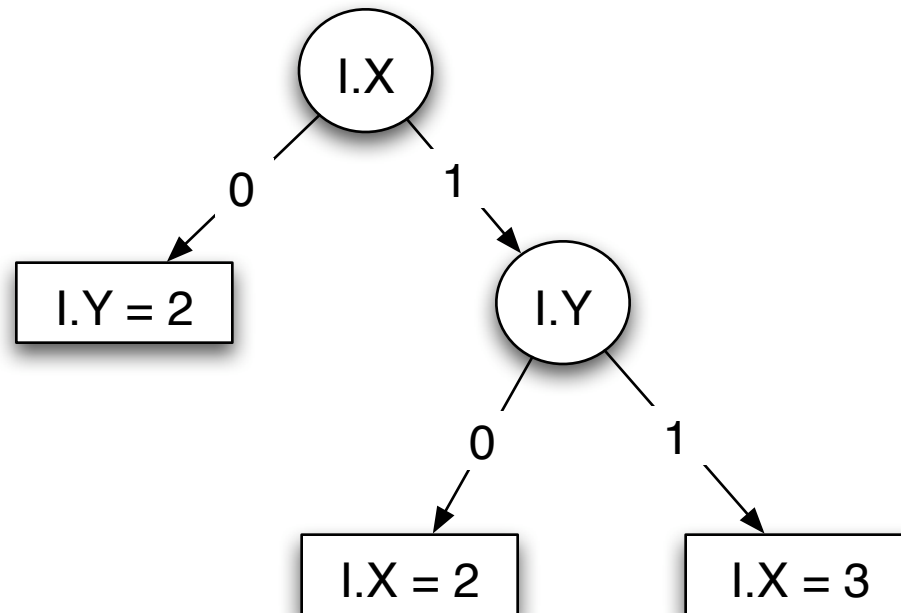
Schema contains all state.

Transitions are atomic.

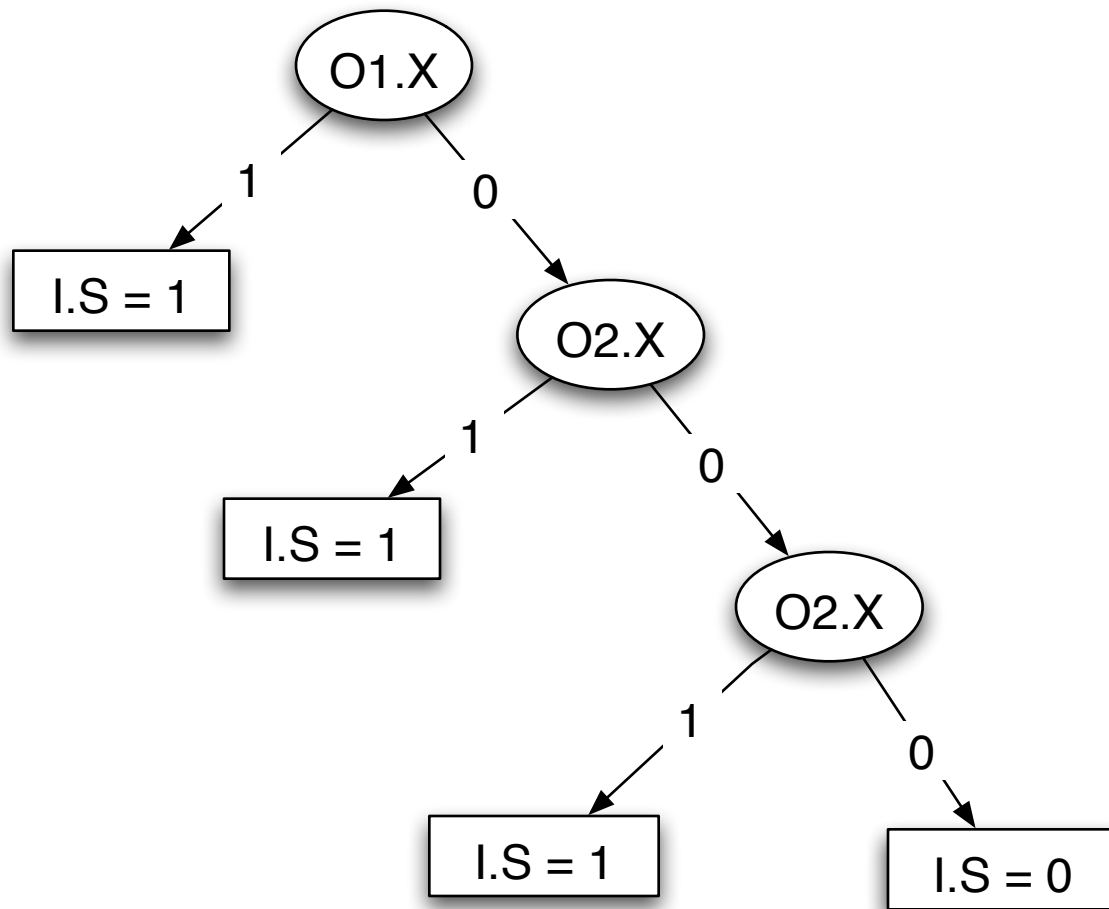
POM lets us intercept all state changes
so can build dependencies on the fly.

Backtrack until dependencies converge.

```
if I -> get('X') then
  if I -> get('Y') then
    I -> set('X', 2)
  else
    I -> set('X', 3)
  endif
else
  I -> set('Y', 2)
endif
```



```
I -> set('S', 0)
foreach O in Query(...) {
  if ( O -> get('X') ) I-> set('S', 1)
}
```



Dynamic Objects

Finite pool of dynamic objects.

Lifecycle of inactive, active, destroyed.

No objects left imply no transition possible.

Time outs and invariants to handle inconsistent states.

Final Remarks

This approach works.

It's accessible to 4th year CS students.

How will it scale to "real" problems?