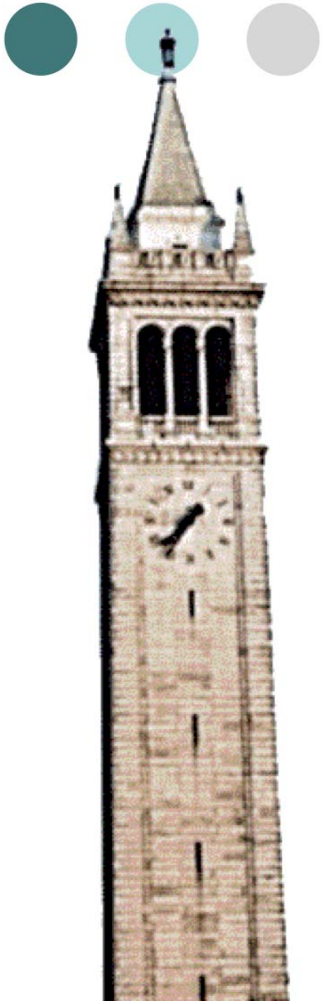


Model-Based Code Generation is not a Replacement for Programming

Edward A. Lee

Robert S. Pepper Distinguished Professor

UC Berkeley

With thanks to Man-Kit Leung, Bert Rodier, Gang Zhou

Invited Talk

Workshop on Software Synthesis (part of ES Week)

Grenoble, France, October 16, 2009

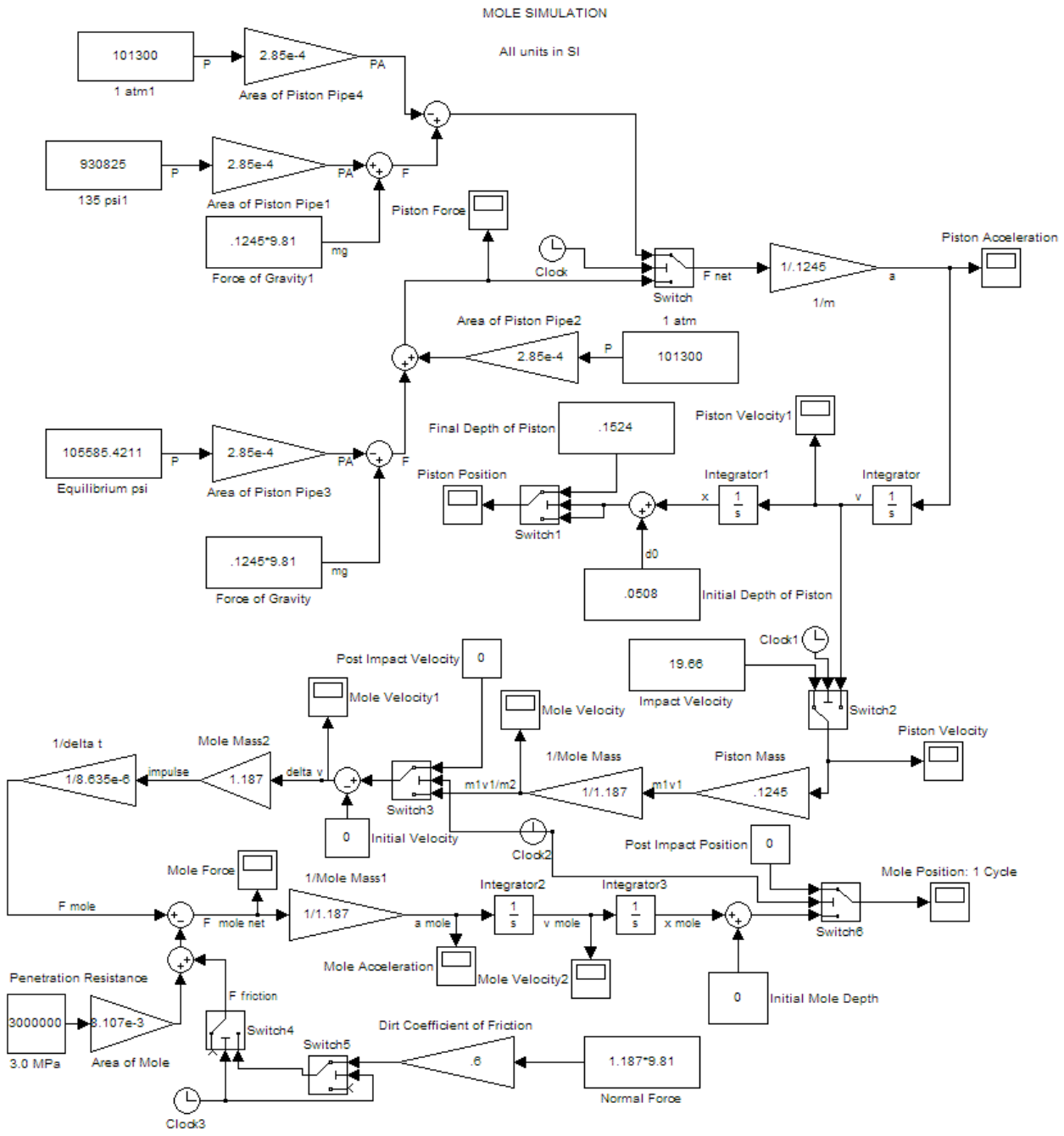
It *is* programming!

- The language is different.
- The language has features well suited to express some things.
- The language is not well suited to express some other things.
- Recognizing the difference appears to be difficult.

Raffaello Sanzio, The Athens School

The Problem

Students, professors, engineers, and even grownups will use whatever modeling tool they are familiar with for every task at hand, whether it is suitable or not.



Properties of Languages

- Modeling languages like Simulink
  - Concurrent
  - Timed
  - Express dynamics well
- Imperative languages like C
  - Sequential
  - Untimed
  - Express algorithms well
- Hybrid languages that mix imperative with threads like Java
  - Sequential unstructured nondeterminate concurrency
  - Untimed
  - Express few things well, unless you limit yourself to the sequential subset.

Maybe the features of the first two should be mixed in a different way

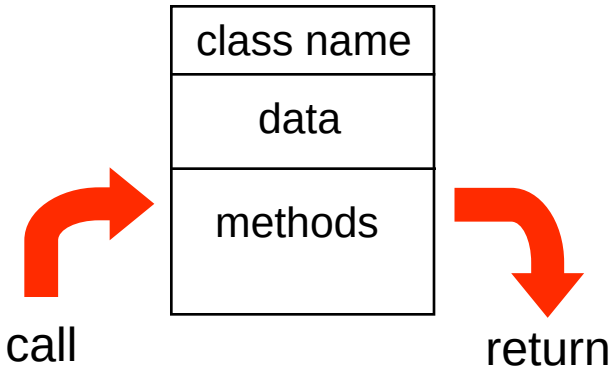
Respect for Imperative Reasoning!

- Imperative reasoning (algorithms, proofs, recipes, etc.) is unnatural to express in actor-oriented languages (as it is also in functional languages).
- Banning imperative reasoning does not seem like a good idea.
- Since people seem to insist on a homogeneous solution, the result is that only a tiny fraction of programmers use actor-oriented languages.

This can be fixed!

Our Proposal: Modeling Languages as Component Architectures rather than Languages

Established component architectures: Object-oriented:

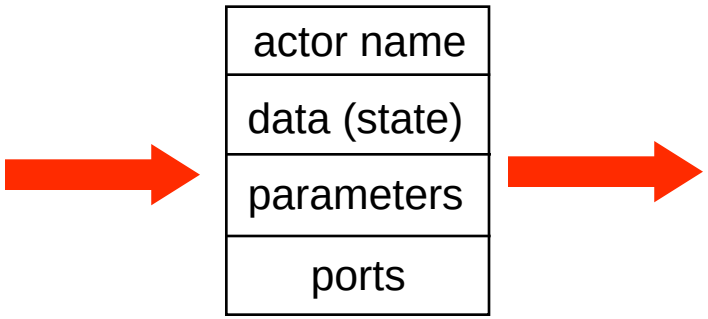


What flows through an object is sequential control

Things happen to objects

Proposed component architectures: Actor oriented:

Actors make things happen

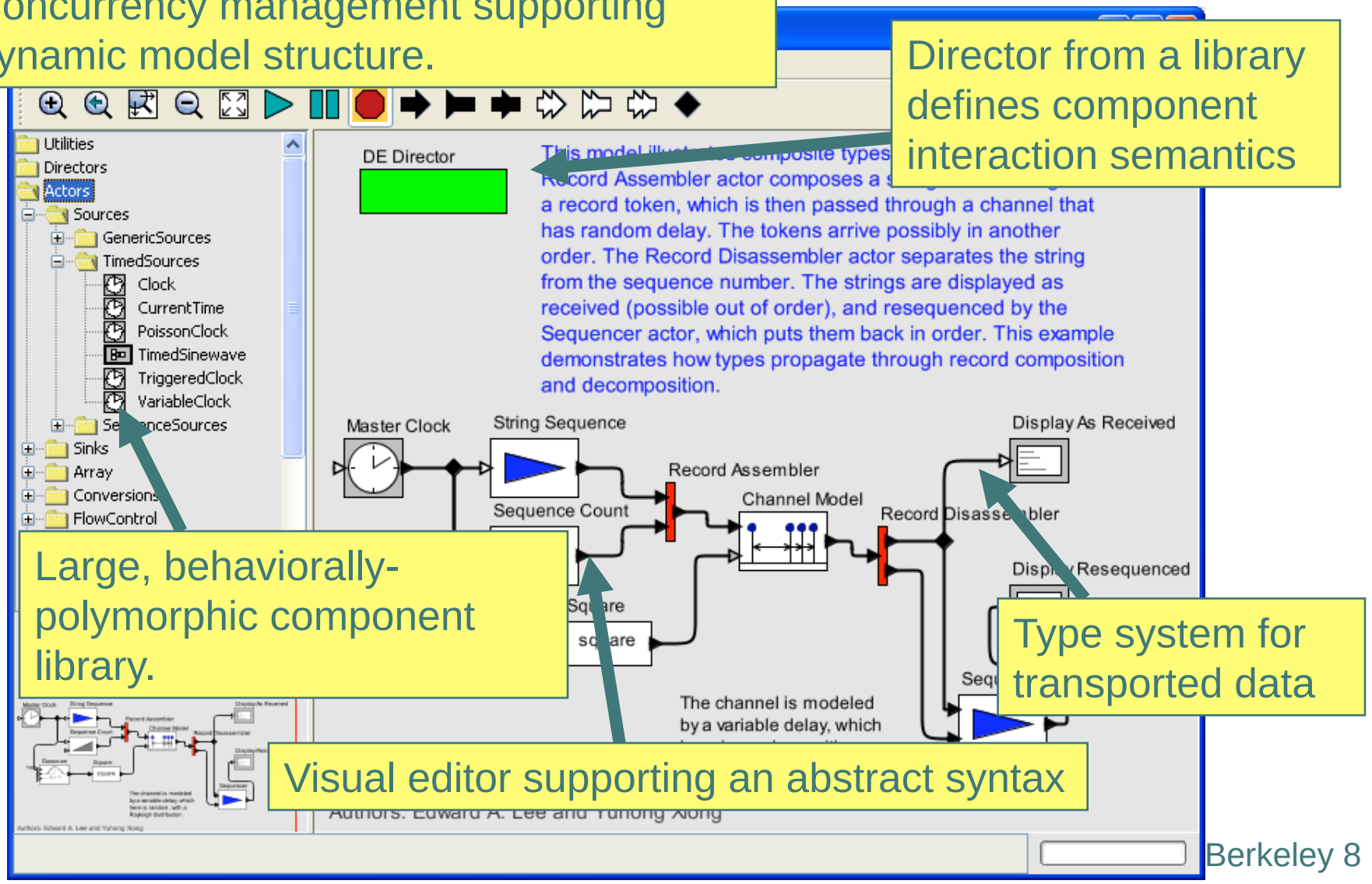


What flows through an object is evolving data

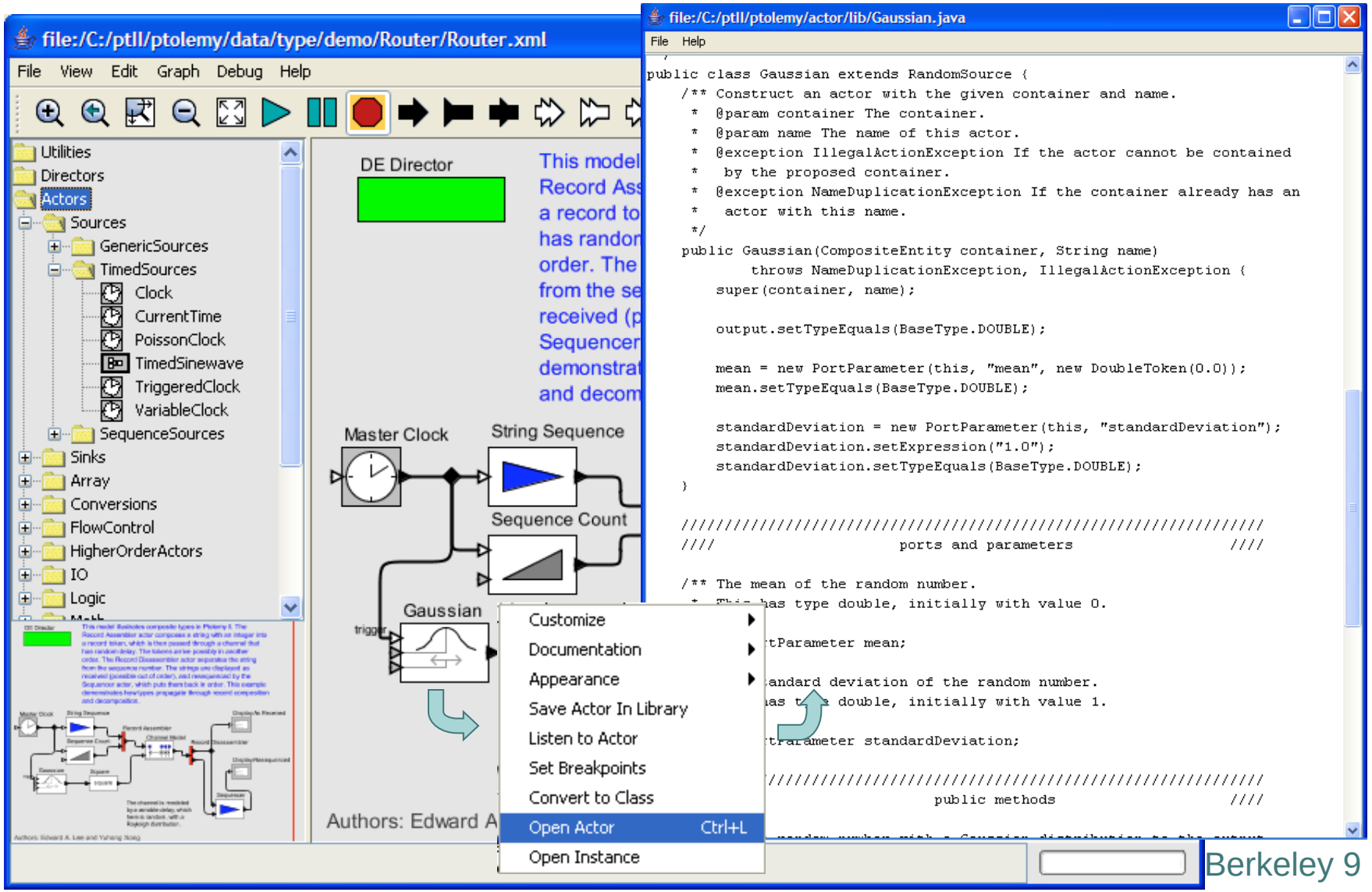
Ptolemy II: Our Open-Source Laboratory for Experiments with Actor-Oriented Design

http://ptolemy.org

Concurrency management supporting dynamic model structure.



Approach: Concurrent Composition of Software Components, which are themselves designed with Conventional Languages (Java, C, C++ MATLAB, Python)



The challenge is to synthesize good implementations from the blend!

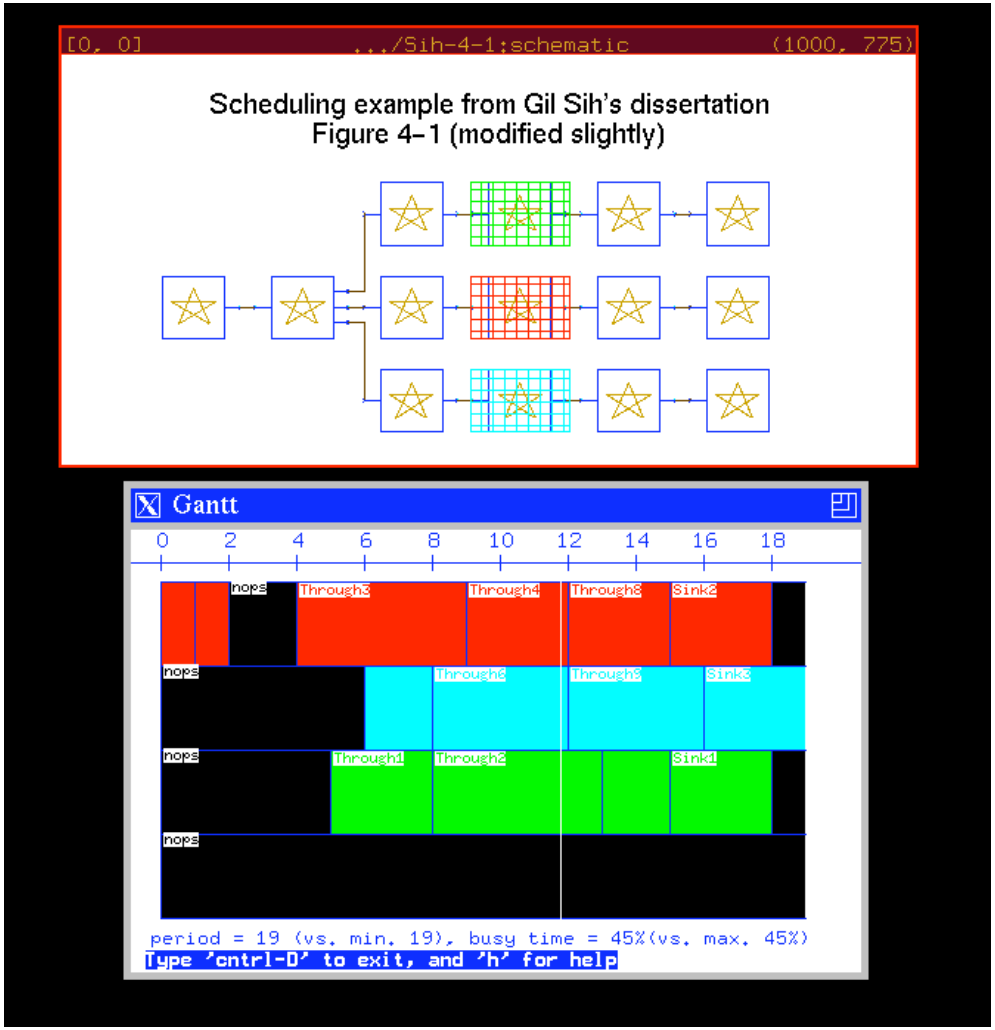
This would be a good problem for bored compiler people!

Our attempts:

- Ptolemy Classic (Buck, Pino, Ha, ... 1990-1997)
- Copernicus (Neuendorffer, 2002-2006)
- Ptolemy II codegen version 1 (2004-2008)
- Ptolemy II codegen version 2 (2009- ??)

Ptolemy Classic Leveraged SDF to Generate Parallel Code

SDF model, parallel schedule, and synthesized parallel code (1990)



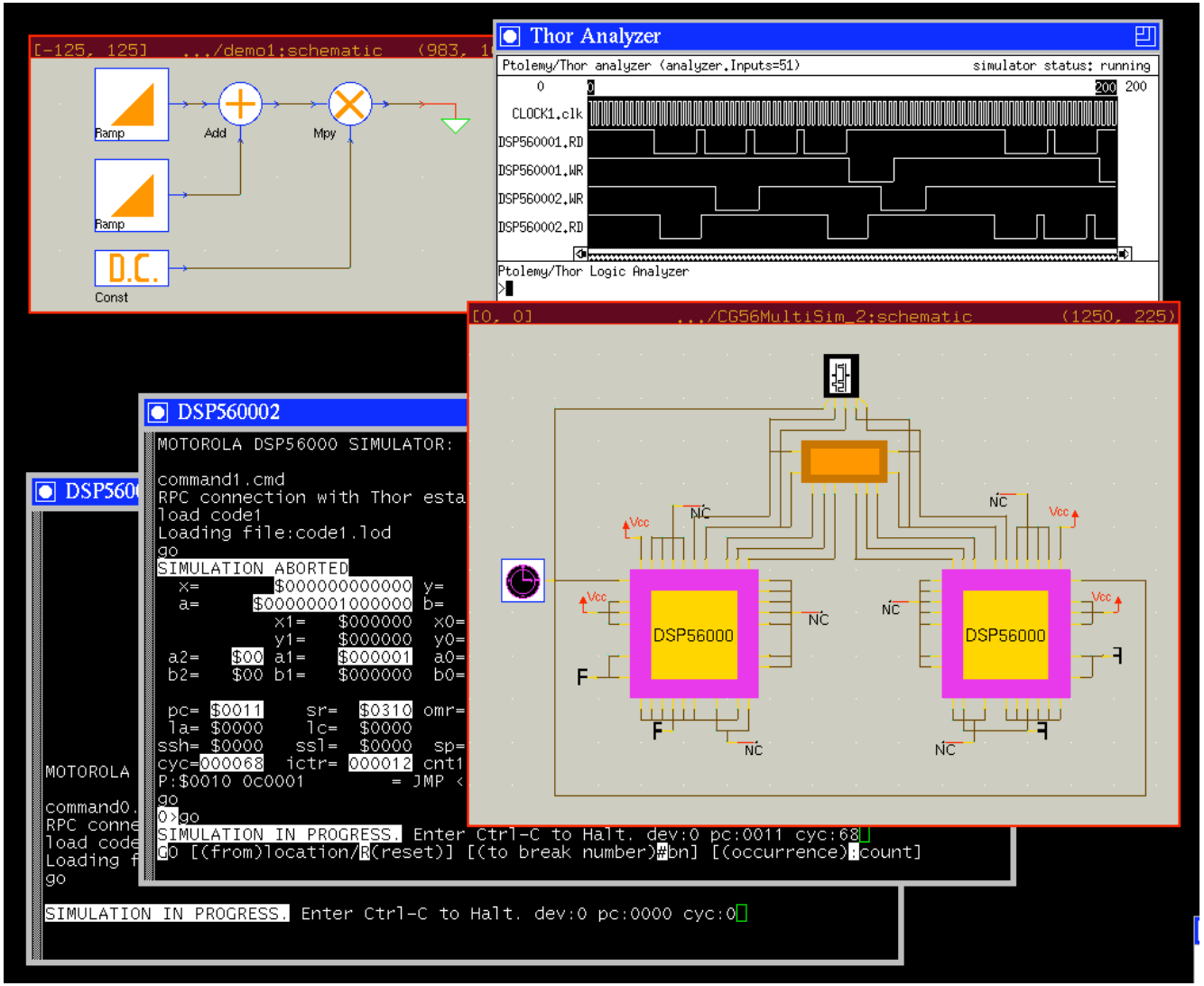

```
        codeblock(std) {
        ; initialize address registers for coef and
 delayLineove     #$addr(coef)+$val(coefLen)-1,r3
; insert here
        move     $ref(delayLineStart),r5
; delayLine
        move     #$val(stepSize),x1
        move     $ref(error),x0
        mpyr     x0,x1,a
        move     a,x0
        move     x:(r3),b          y:(r5)+,y0
        }

        codeblock(loop) {
        do       #$val(loopVal),$label(endloop)
        macr     x0,y0,b
        move     b,x:(r3)-
        move     x:(r3),b          y:(r5)+,y0
$label(endloop)
        }

        codeblock(noloop) {
        macr     x0,y0,b
        move     b,x:(r3)-
        move     x:(r3),b          y:(r5)+,y0
        }
```

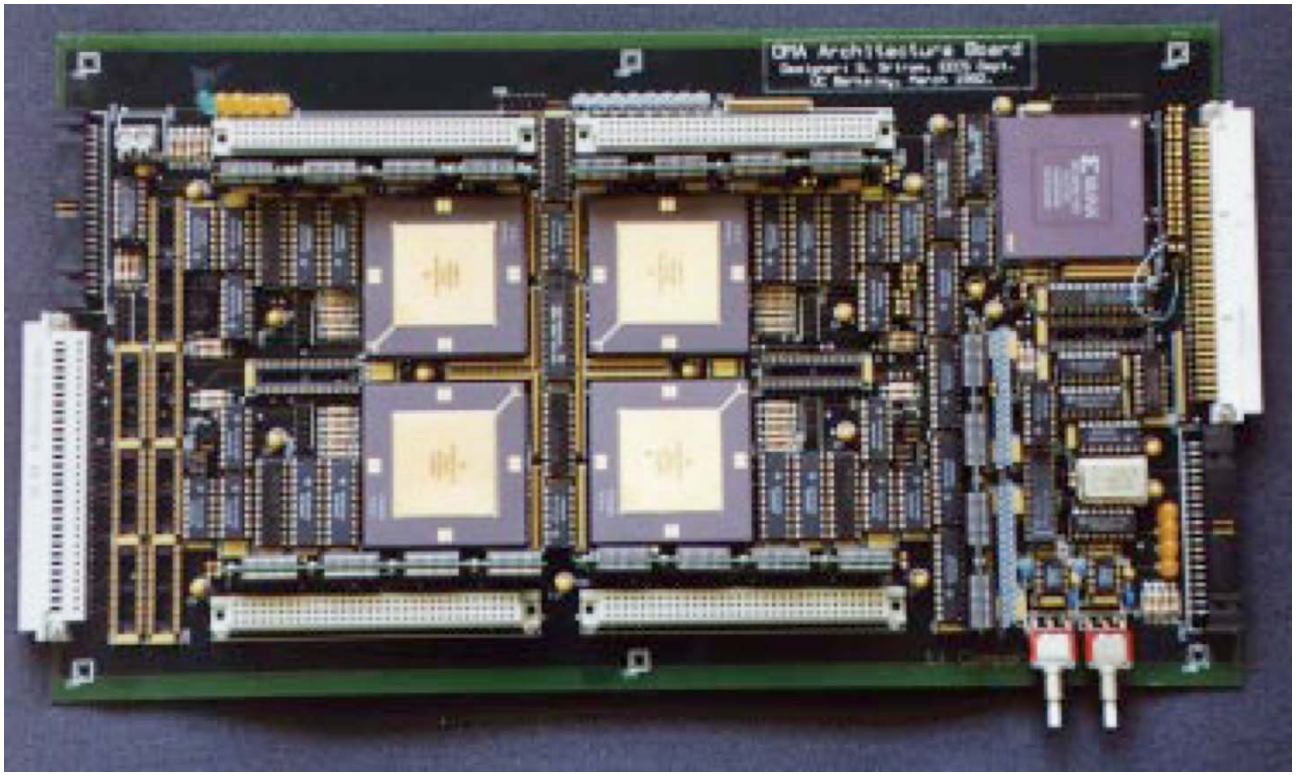
It is an interesting (and rich) research problem to minimize interlocks and communication overhead in complex multirate applications.

Ptolemy Classic Provided Cosimulation of Hardware and Generated Software



An SDF model, a "Thor" model of a 2-DSP architecture, a "logic analyzer" trace of the execution of the architecture, and two DSP code debugger windows, one for each processor (1990).

Multicore Architecture Targeted by Ptolemy Classic Code Generation (1993)

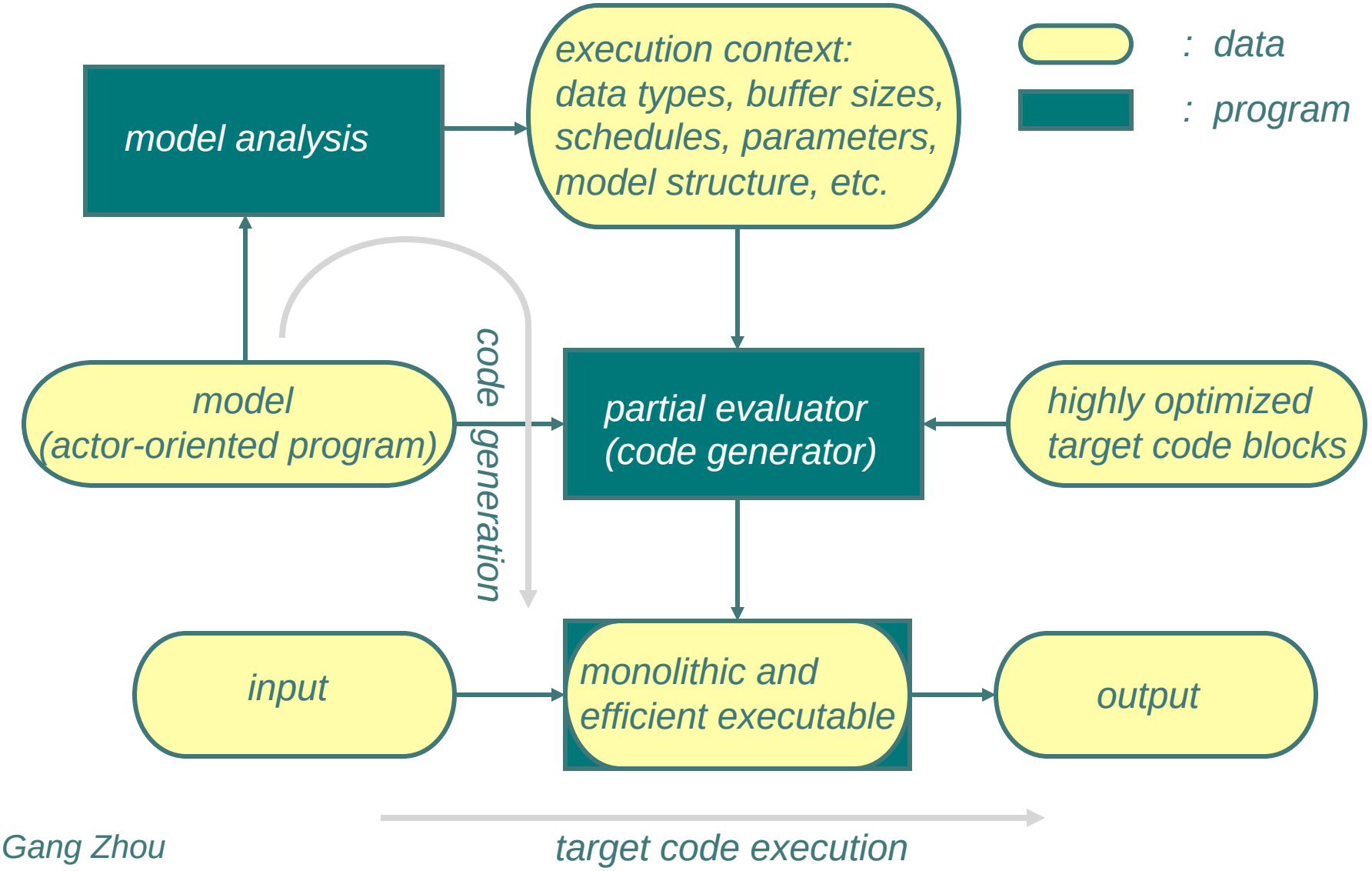


Four DSP 96000 floating point processors interconnected using the "ordered memory architecture," which greatly reduced shared memory synchronization costs [Sriram, 1993]

Second Attempt (Copernicus)

- Steve Neuendorffer created in Ptolemy II a code generator base on the idea of object specialization.
- Java objects would be translated at the byte code level to more specialized Java objects based on their usage in a particular context.
- A tour-de-force, but unmaintainable in our context...

Third attempt: resurrect Ptolemy Classic codegen, but with partial evaluation concepts



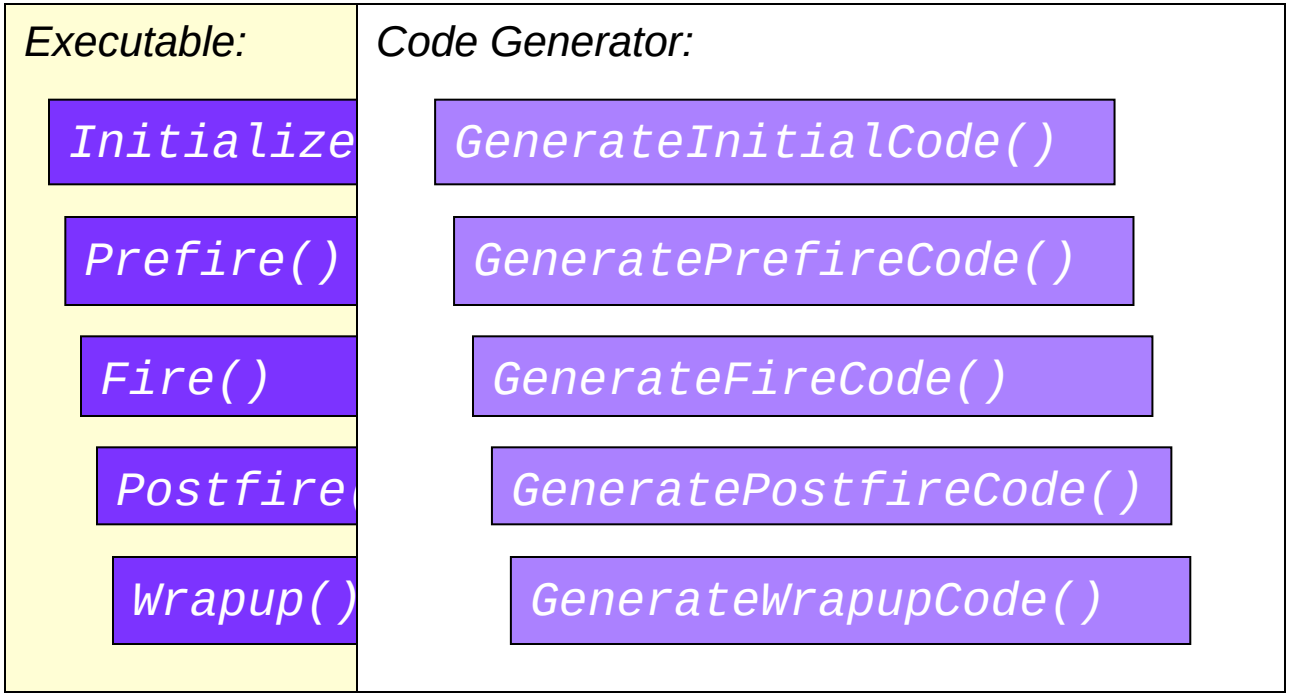
Gang Zhou
Man-Kit Leung

The Code Generation Process

- Definition

  ITERATION := *prefire* . fire* . *postfire*

  EXECUTION := *initialize* . ITERATION* . *wrapup*



The Code Generation Process

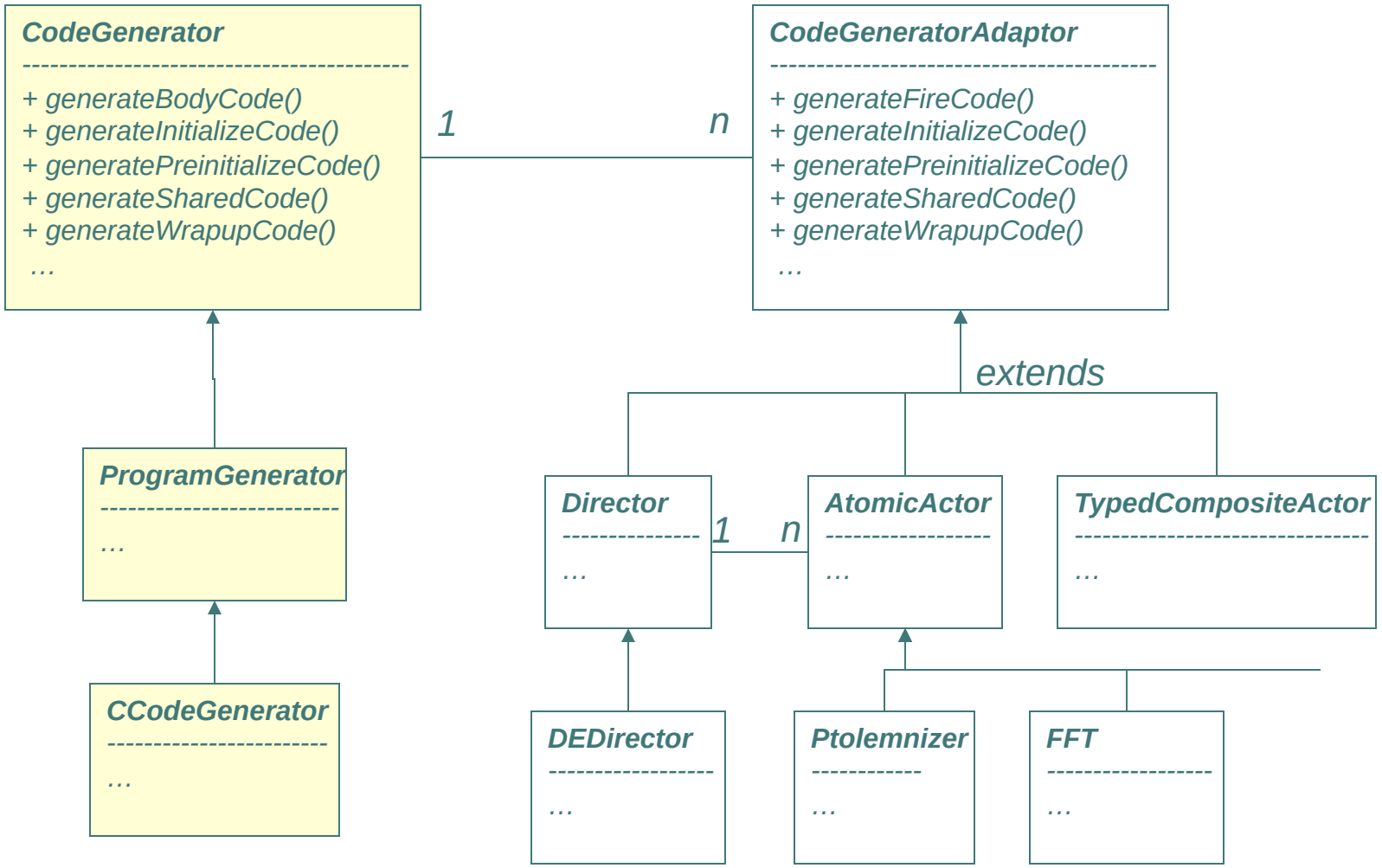
```
CompositeActor:
  fire() {
    D.fire()
  }
------------------------
Director:
  fire() {
    order = getSchedule()
    for each A ∈ order
      A.fire()
  }
```

Generated code:

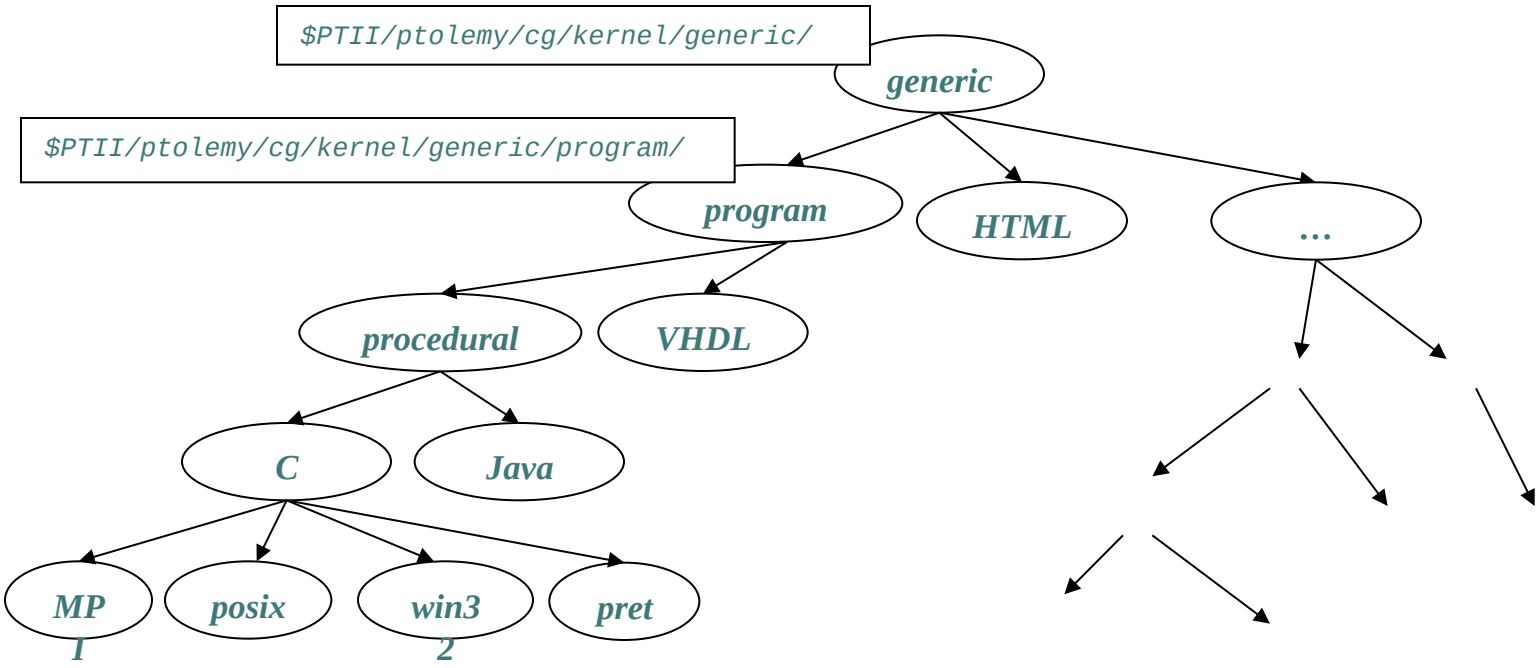
```
CompositeActor_fire() {
   A1_fire();
   A2_fire();
   A3_fire();
}
```

The order is pre-determined at the time of code generation

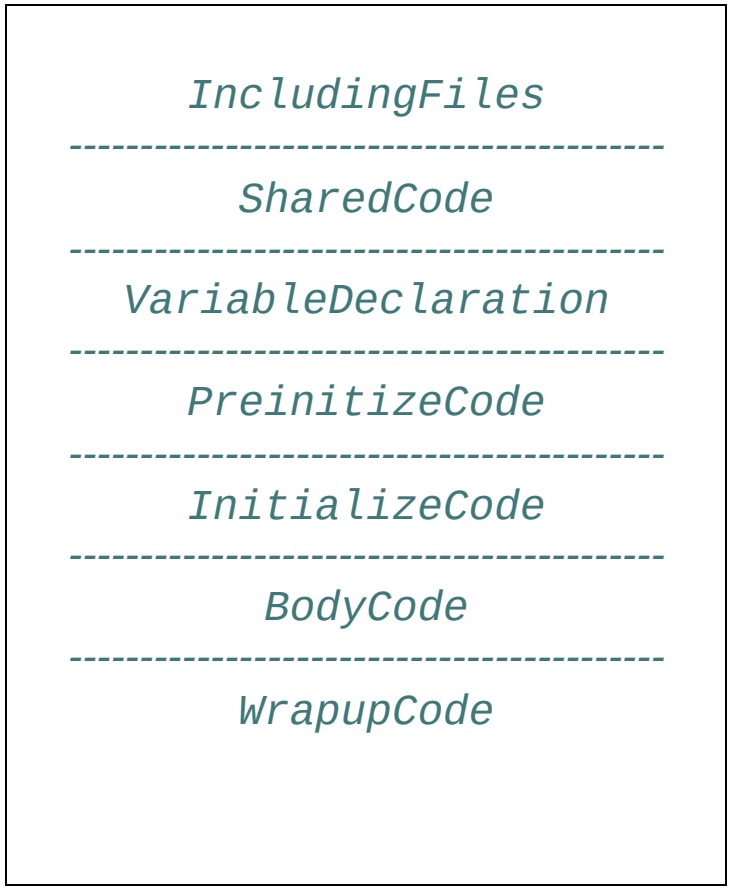
Fourth attempt: Build on this, but create a Software Architecture for Experimentation



Target Hierarchy

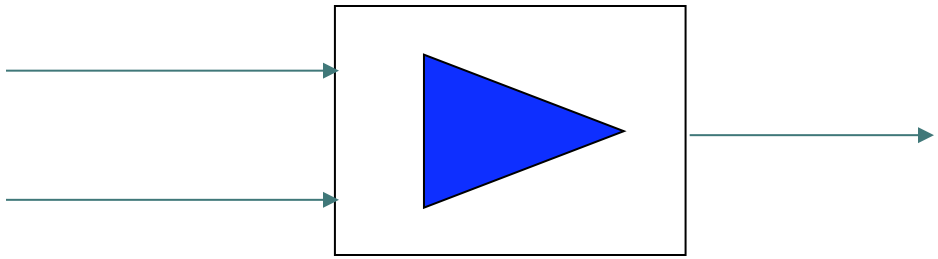


Sections of the Generated Content:



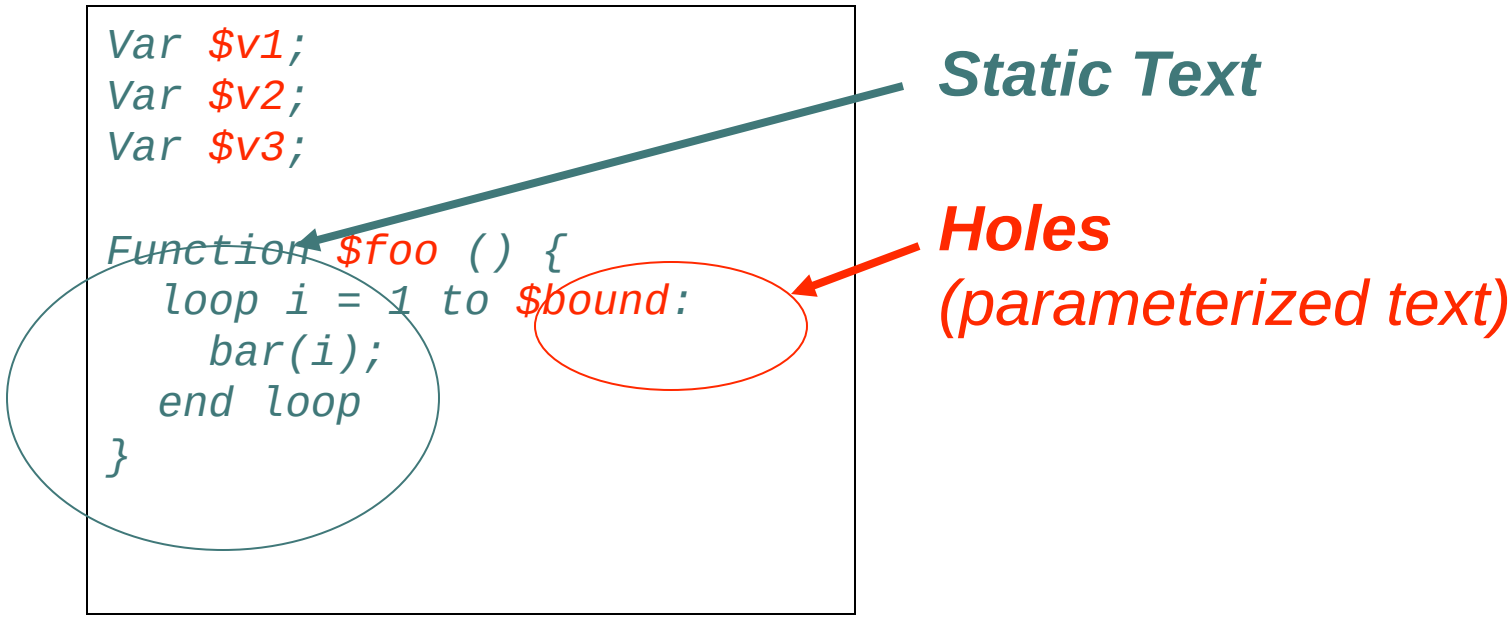
Non-trivial Components

If we need to generate complex code for an atomic component (e.g. FFT) that is highly parameterizable…

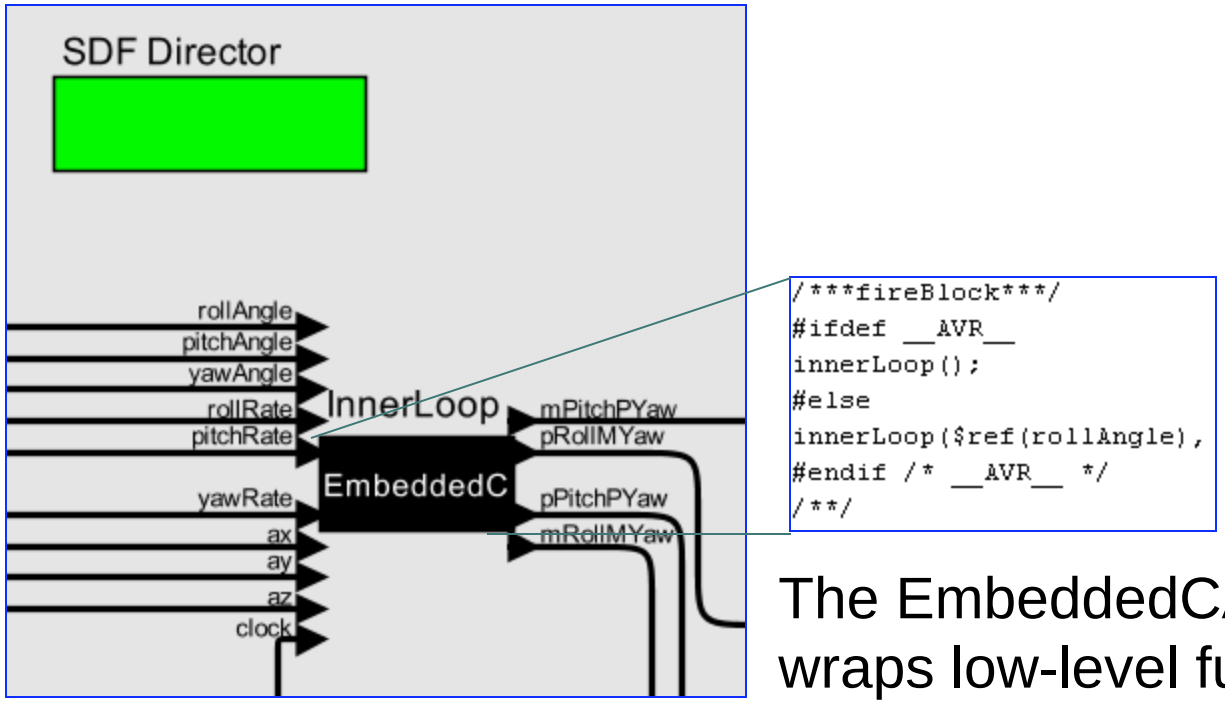


Meta Programming

Our (rather primitive) meta-programming mechanism uses templates:



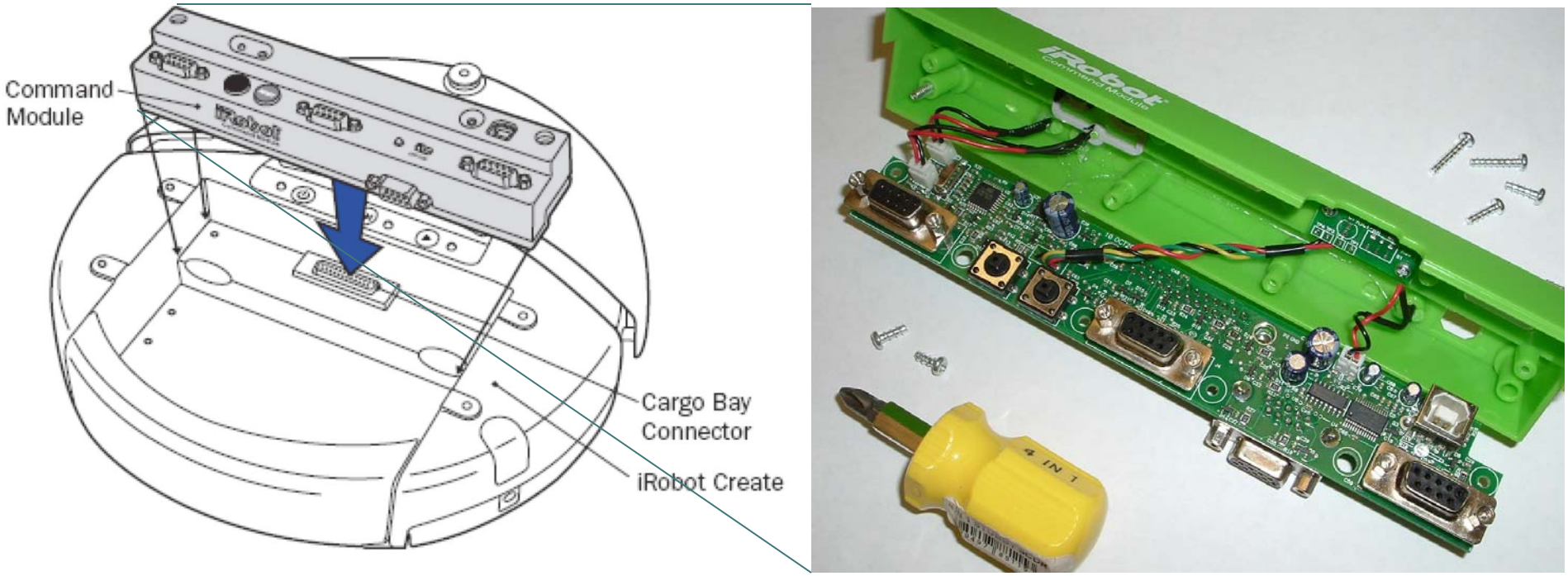
This mechanism enables integration of C code into actor-oriented models.



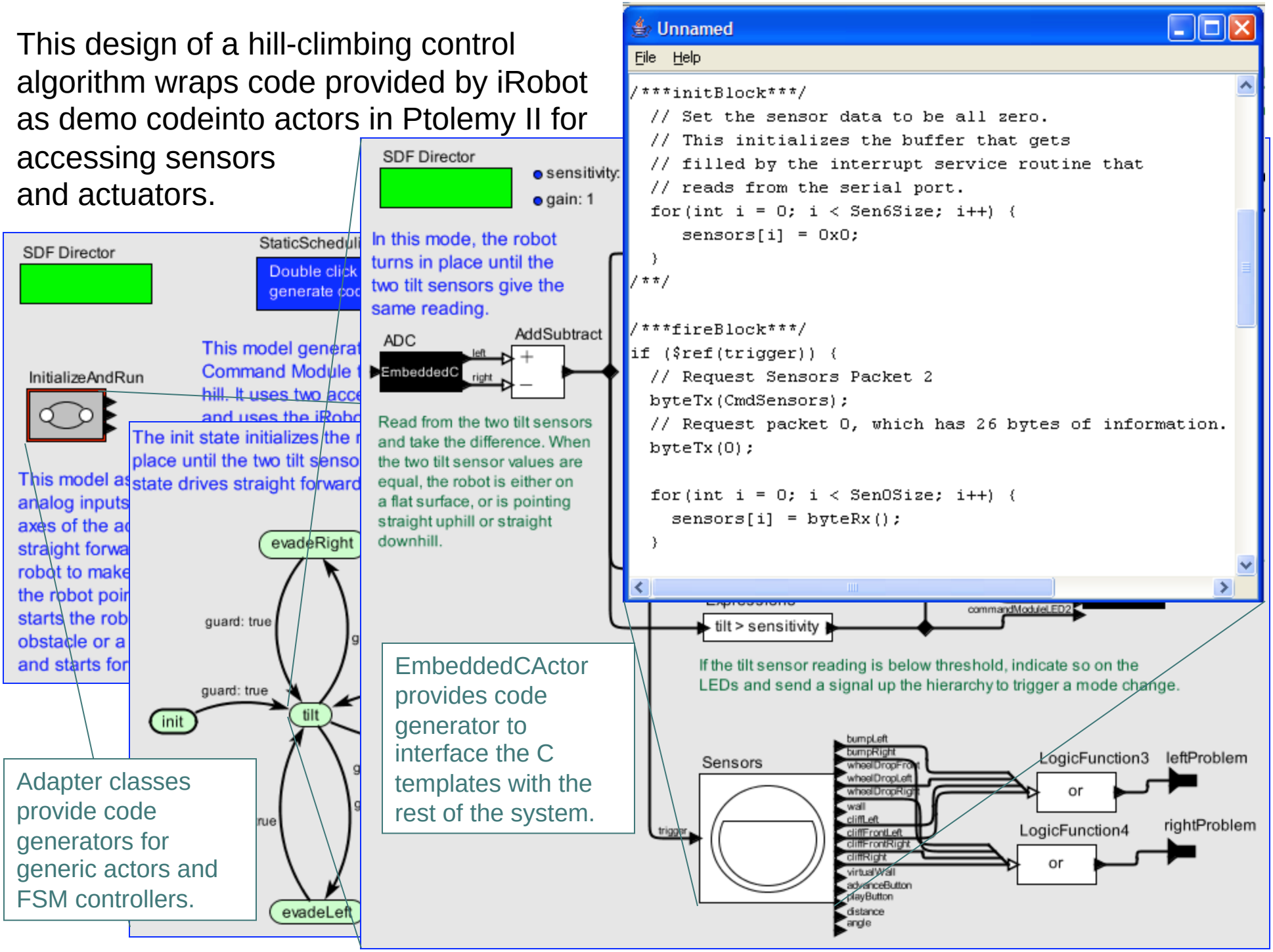
The EmbeddedCActor in Ptolemy II wraps low-level functionality (written in C) to define an actor. This approach makes it easy to build actor-oriented models and to generate efficient, platform-specific C implementations.

Example target showing that very low overhead code generation and integration of legacy C code is possible.

The iRobot Create (the platform for the Roomba vacuum cleaner) with a pluggable Command Module has an Atmel 8-bit microcontroller with a very small amount of memory.



This design of a hill-climbing control algorithm wraps code provided by iRobot as demo codeinto actors in Ptolemy II for accessing sensors and actuators.



Conclusion

- Heterogeneous models
  - Actor models
  - Imperative code
- Code generation
  - Synthesis of practical realization
- A component technology
  - Chunks of imperative logic encapsulated in a concurrent MoC