Classes and Inheritance in Actor-Oriented Models

Stephen Neuendorffer
Edward Lee
UC Berkeley

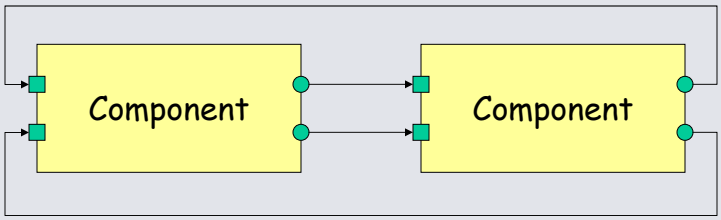
Chess Review
May 8, 2003
Berkeley, CA

Chess

Introduction

- Component-based design
  - Object-oriented components
  - Actor-oriented components
- Most Actor-oriented tools lack the class mechanisms of Object-oriented languages.
  - inheritance
  - subclassing
- A preliminary approach to providing class-like mechanisms in Ptolemy II

Component-based Design



Complex systems built primarily through composition.

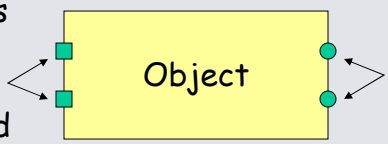
Encapsulation of intellectual property.

Visual languages and design tools.

Component reuse.

Object-Oriented Components

Provides ports expose methods that can be invoked on this object



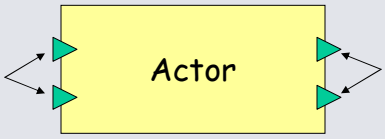
Requires ports expose methods that this object might invoke.

This interface specification allows for consistency checking of compositions.

However, this interface lacks important pieces of information:

- Method Requirements. (Sequencing? Preconditions?)
- Concurrency constraints. (Deadlock? Re-entrancy?)

Actor-Oriented Components



Input ports expose flows of data that this actor consumes.

Output ports expose flows of data that this actor produces.

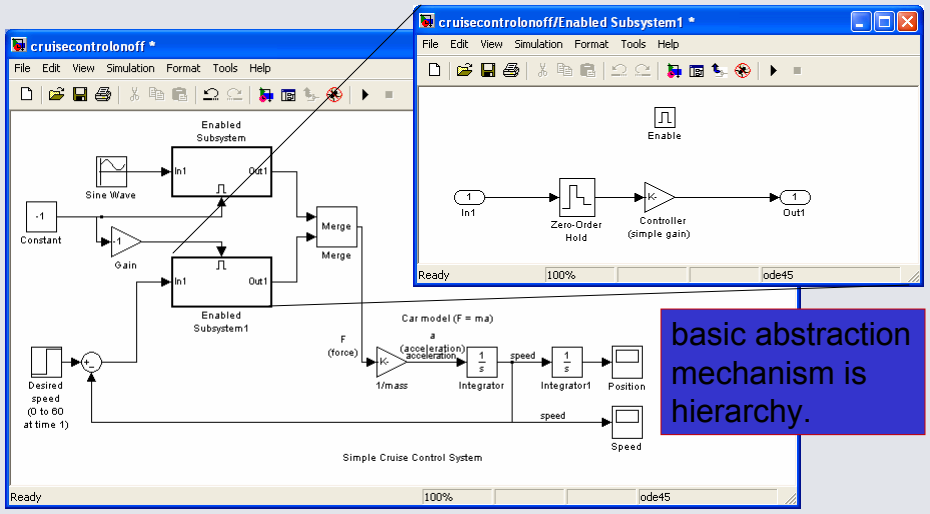
This interface specification allows for consistency checking of compositions.

This interface also lacks important pieces of information:

- How data is transported between ports
- Concurrency constraints between actors

These are largely orthogonal issues for actors

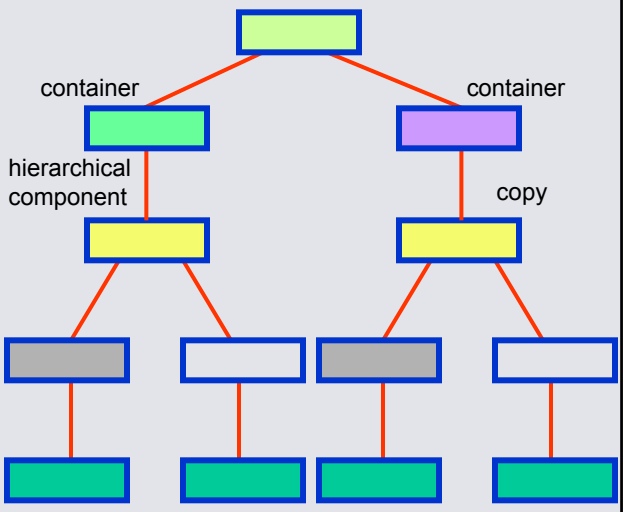
Example of an Actor-Oriented Framework: Simulink



basic abstraction mechanism is hierarchy.

Hierarchical Abstraction

- Complex components can encapsulate smaller components.
- Object-oriented *delegation pattern*

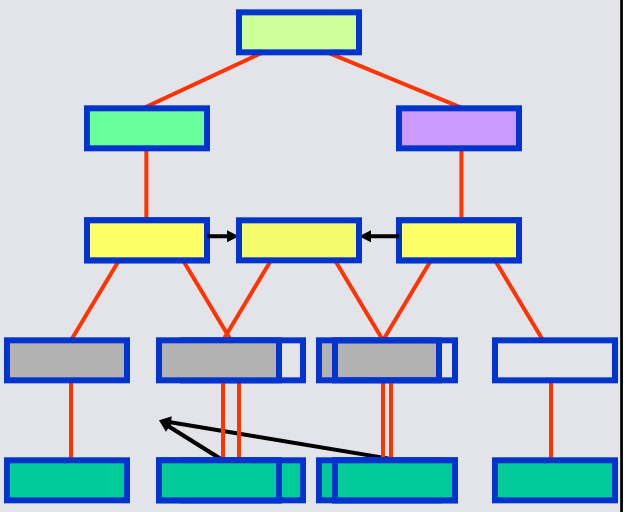


Class mechanisms

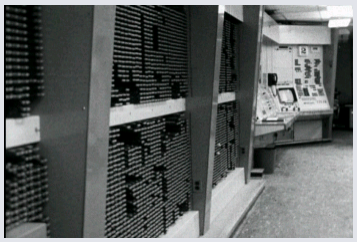
- Realization:
  - Most components in a large system operate in the same basic fashion.
- Object-oriented *classes* provide several important capabilities.
  - Central point of design.
  - Basis for type checking.
  - Static compilation.
  - Extension and variation.
- But also present some complications.
  - Run-time modifications become difficult.
  - Source of inconsistencies.

Classes and Hierarchy

- Classes simplify the structure of complex models.
- Classes extend the containment hierarchy with an inheritance hierarchy.

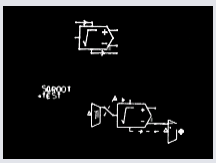


The First (?) Actor-Oriented Programming Language (1966)



MIT Lincoln Labs TX-2

Bert Sutherland with a light pen



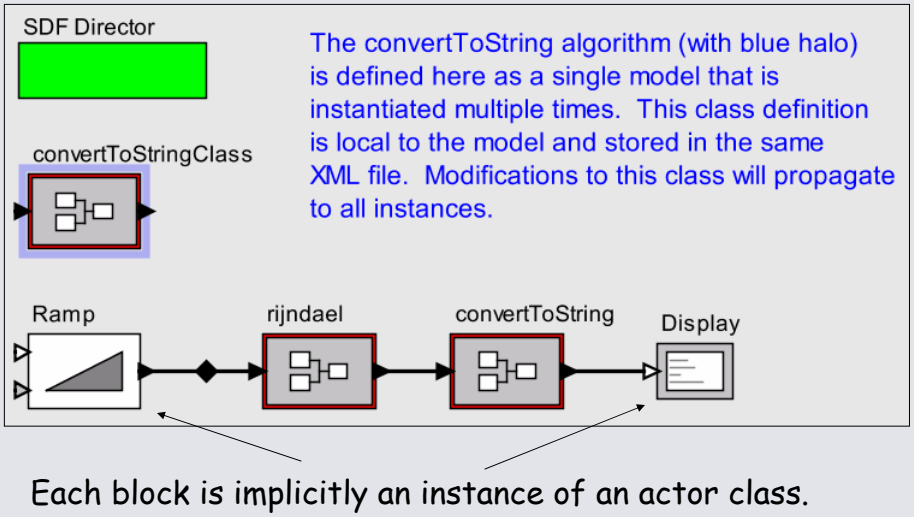
Partially constructed actor-oriented model with a class definition (top) and instance (below).

Bert Sutherland used the first acknowledged object-oriented framework (Sketchpad, created by his brother, Ivan Sutherland) to create the first actor-oriented programming framework.

Key Problems

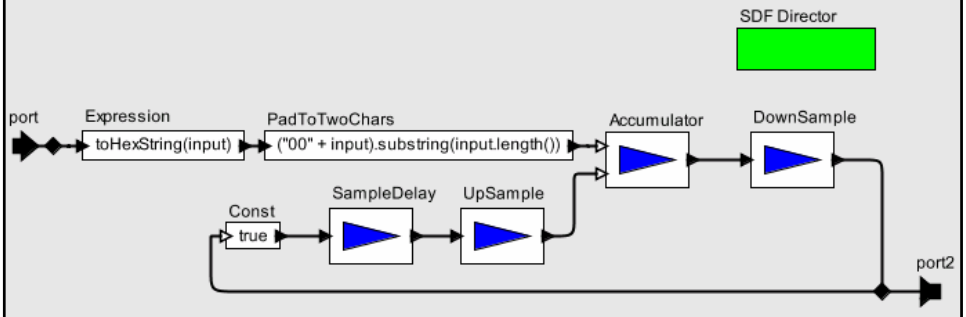
- Direct manipulation user interface of both classes and instances.
  - Maximize syntactic consistency.
- Expressive uses of classes
  - subclasses with extension and overriding
  - nested classes
- Interactive modification of classes
  - Classes might be modified at runtime
- Distinguishing overridden values from inherited default values.

Visual Class Representation



Each block is implicitly an instance of an actor class.

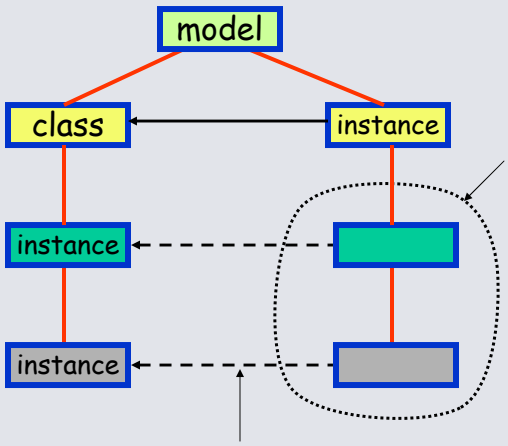
An Actor Class



Every subclass or instance of this actor class contains at least this structure.
Parameter values of actors in a class give default values for all instances of the class

Models with Classes

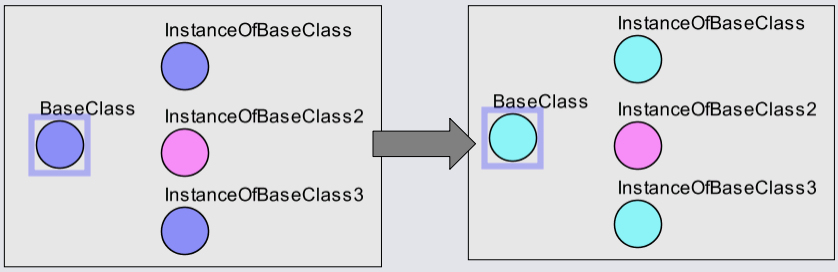
Actor classes are explicitly instantiated.



Derived objects implied by class

Intuition: Modifications to classes propagate to derived objects, as long as local changes have not been made.

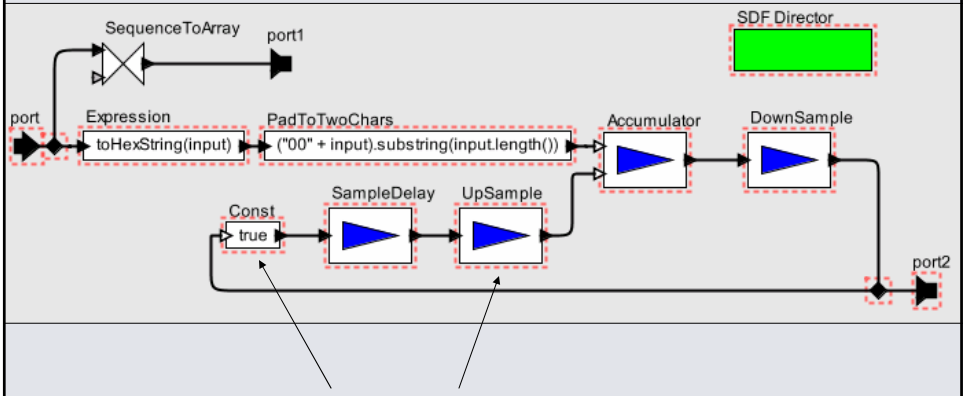
A Simple Example



Here the property of BaseClass is modified, which does not propagate to Instance2.

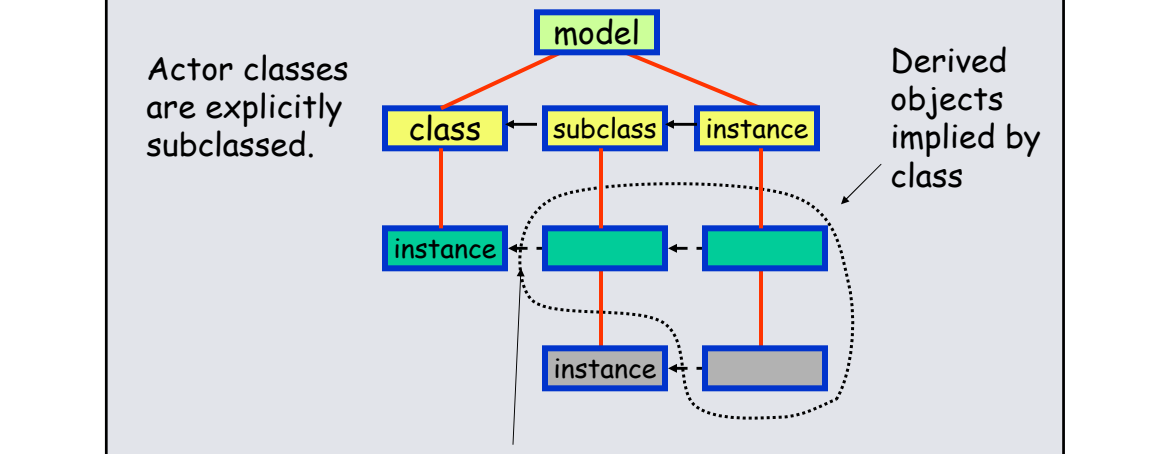
This example is simple because there is only one propagation path.

A Subclass



Dotted lines show inherited objects that cannot be deleted, while allowing syntax-directed editing.

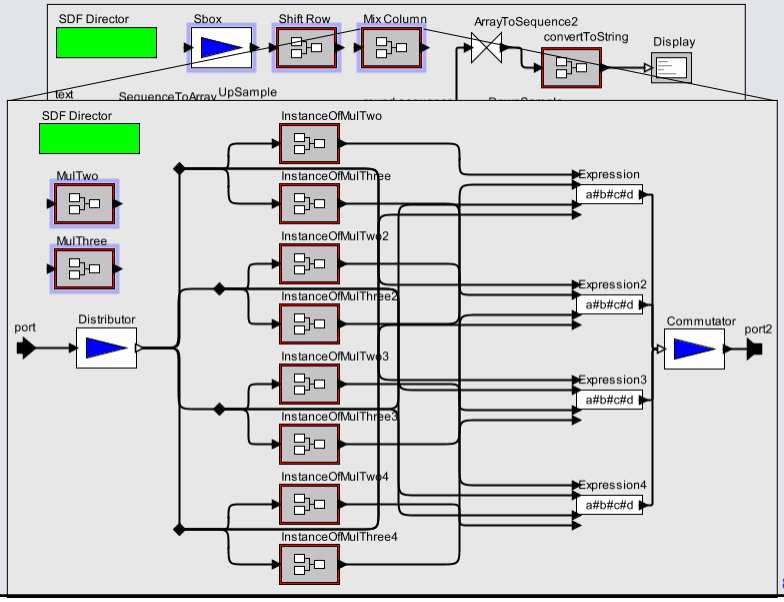
Models with SubClasses



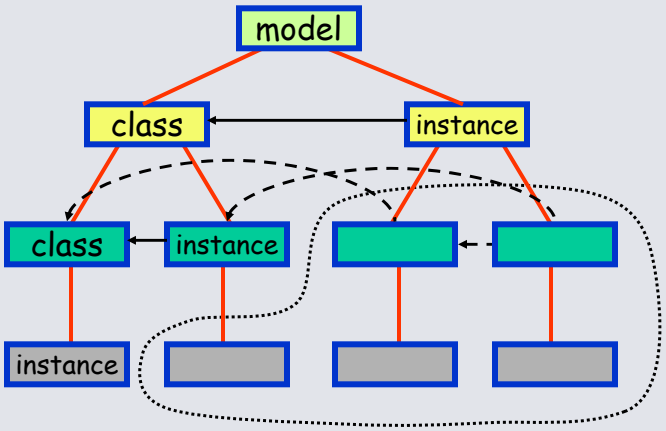
Changes propagate to subclasses similarly to instances.

Subclasses allow for independent extension, while instances do not.

Nested Classes



Models with Nested Classes



Nested classes result in multiple propagation paths.

Approach: prioritize propagation so that localized changes override global changes.

Summary

- Class mechanisms for modularity can be integrated with actor-oriented modeling.
- Inherited changes in a syntactically-driven environment can be tricky:
  - Nested Classes
- Still many open questions..
  - Consistency given multiple propagations?
  - Is "Local Override" property the best one?