Behavioral Types as Interface Definitions for Concurrent Components

Edward A. Lee
Professor
UC Berkeley

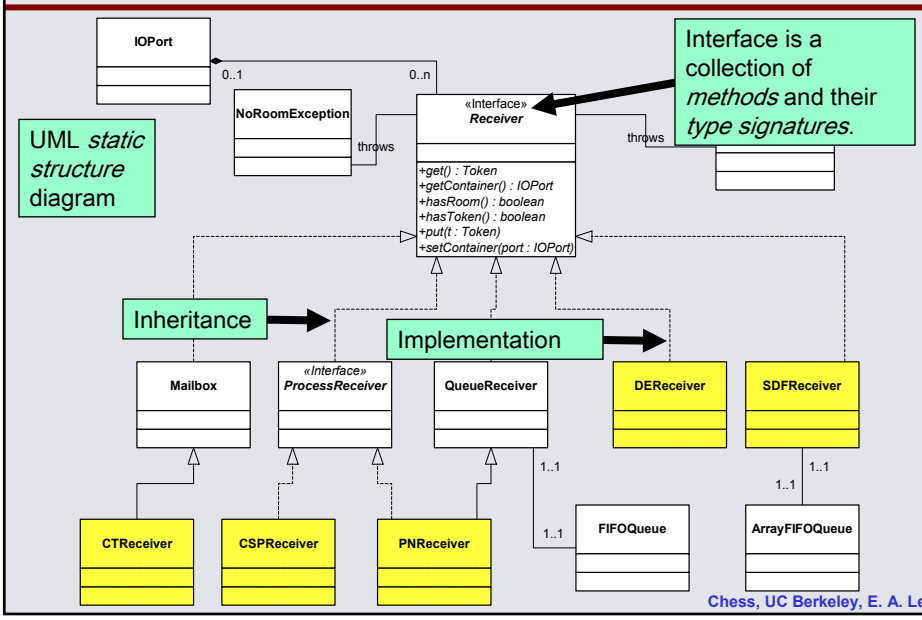
Invited Talk
NEXT TTA Workshop on the Specification of Linking Interfaces
Oct. 12 2003
Philadelphia, PA, USA

Chess

Center for Hybrid and Embedded Software Systems

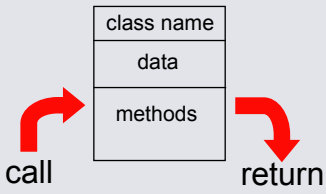
Object-Oriented Design

An Approach to Component Interface Specification



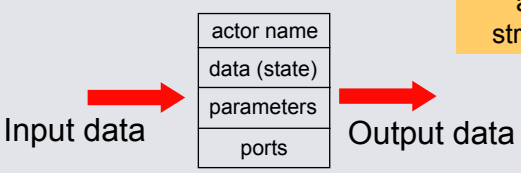
Focus on Actor-Oriented Design

- Object orientation:



What flows through an object is sequential control

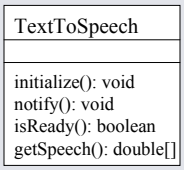
- Actor orientation:



What flows through an object is streams of data

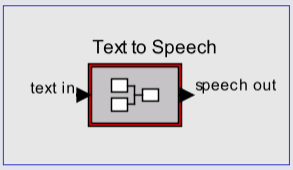
Actor Orientation vs. Object Orientation

Object oriented



OO interface definition gives procedures that have to be invoked in an order not specified as part of the interface definition.

Actor oriented



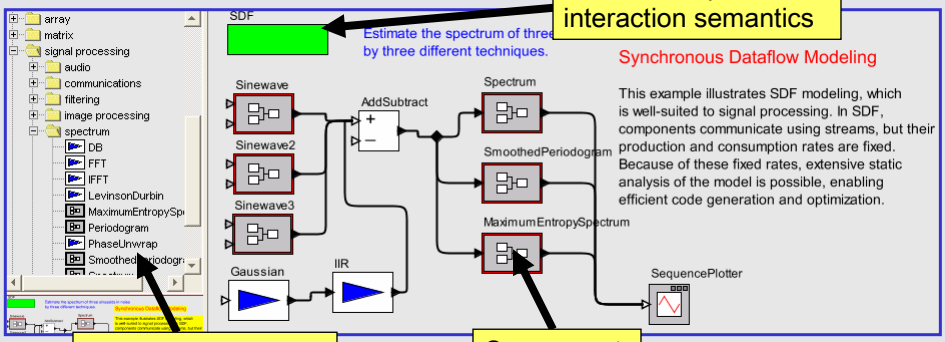
actor-oriented interface definition says "Give me text and I'll give you speech"

- Identified problems with object orientation:
  - Says little or nothing about concurrency and time
  - Concurrency typically expressed with threads, monitors, semaphores
  - Components tend to implement low-level communication protocols
  - Re-use potential is disappointing
- Actor orientation offers more potential for useful modeling properties, and hence for model-based design.

Example of Actor-Oriented Design (in this case, with a visual syntax)

Ptolemy II example:

Director from a library defines component interaction semantics



Large, polymorphic component library.

Component

Key idea: The model of computation is part of the framework within which components are embedded rather than part of the components themselves. Thus, components need to declare behavioral properties.

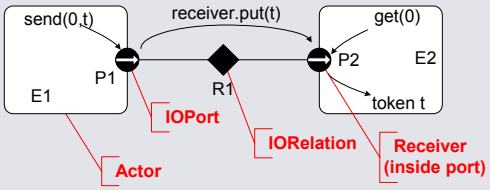
Model of Computation:
- Messaging schema
- Flow of control
- Concurrency

Examples of Actor-Oriented Component Frameworks

- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- Easy5 (Boeing)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- …

Actor View of Producer/Consumer Components

Basic Transport:



Models of Computation:

- push/pull
- continuous-time
- dataflow
- rendezvous
- discrete events
- synchronous
- time-driven
- publish/subscribe
- …

Many actor-oriented frameworks assume a producer/consumer metaphor for component interaction.

Actor Orientation vs. Object Orientation

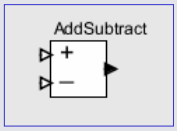
- Object Orientation
  - procedural interfaces
  - a class is a type (static structure)
  - type checking for composition
  - separation of interface from implementation
  - subtyping
  - polymorphism

This is a vision of the future: Few actor-oriented frameworks fully offer this view. Eventually, all will.

- Actor Orientation
  - concurrent interfaces
  - a behavior is a type
  - type checking for composition of behaviors
  - separation of behavioral interface from implementation
  - behavioral subtyping
  - behavioral polymorphism ⟸ Focus on this

Polymorphism

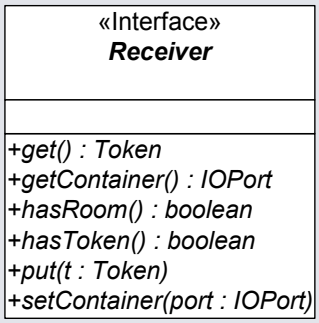
- Data polymorphism:
  - Add numbers (int, float, double, Complex)
  - Add string (concatenation)
  - Add composite types (arrays, records, matrices)
  - Add user-defined types



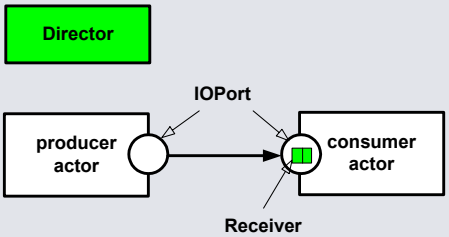
- Behavioral polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a time-triggered model, add when the clock ticks
  - In discrete-event, add when any connected input has data, and add in zero time
  - In process networks, execute an infinite loop in a thread that blocks when reading empty inputs
  - In CSP, execute an infinite loop that performs rendezvous on input or output
  - In push/pull, ports are push or pull (declared or inferred) and behave accordingly
  - In real-time CORBA, priorities are associated with ports and a dispatcher determines when to add

By not choosing among these when defining the component, we get a huge increment in component re-usability. But how do we ensure that the component will work in all these circumstances?

Object-Oriented Approach to Achieving Behavioral Polymorphism

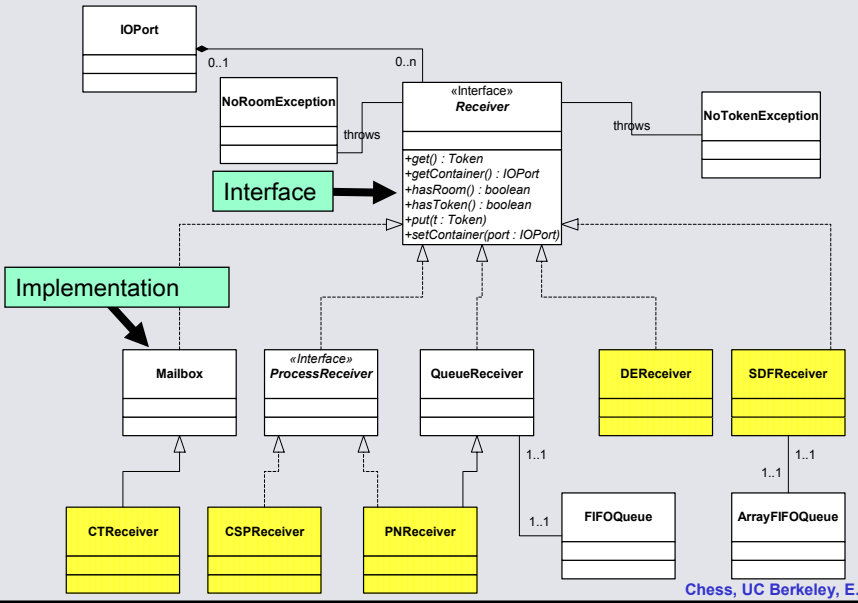


These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is supplied by the director, not by the component.



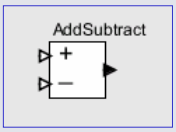
Recall: Behavioral polymorphism is the idea that components can be defined to operate with multiple models of computation and multiple middleware frameworks.

Behavioral Polymorphism
The Object Oriented View



But What If…

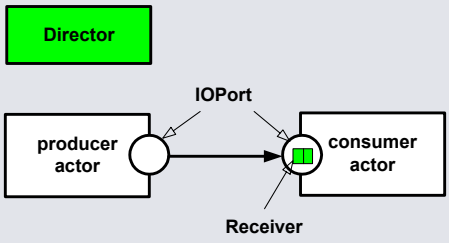
- The component requires data at all connected input ports?
- The component can only perform meaningful operations on 2 successive inputs?
- The component can produce meaningful output before the input is known (enabling it to break potential deadlocks)?
- The component has a mutex monitor with another component (e.g. to access a common hardware resource)?



None of these is expressed in the object-oriented interface definition, yet each can interfere with behavioral polymorphism.

Behavioral Types – A Practical Approach

- Capture the dynamic interaction of components in *types*
- Obtain benefits analogous to data typing.
- Call the result *behavioral types*.

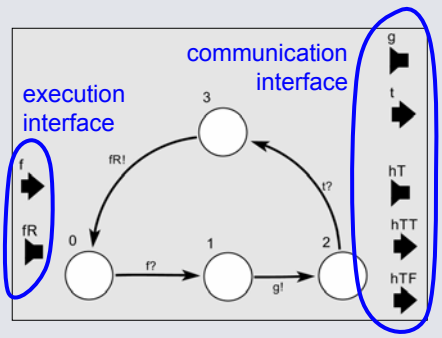


- Communication has
  - data types
  - behavioral types
- Components have
  - data type signatures
  - behavioral type signatures
- Components are
  - data polymorphic
  - behaviorally polymorphic

See Liskov & Wing, ACM, 1994 for an intro to behavioral types.

Behavioral Type System

- We capture patterns of component interaction in a type system framework.
- We describe interaction types and component behavior using extended *interface automata* (de Alfaro & Henzinger)
- We do type checking through *automata composition* (detect component incompatibilities)
- Subtyping order is given by the alternating simulation relation, supporting *behavioral polymorphism*.



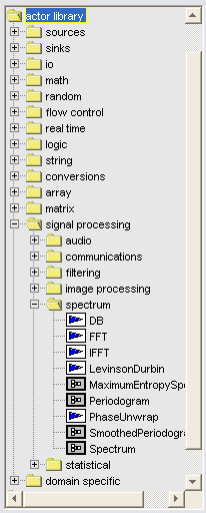
A type signature for a consumer actor.

These behavioral types are an example of an *interface theory* (Henzinger, et al).

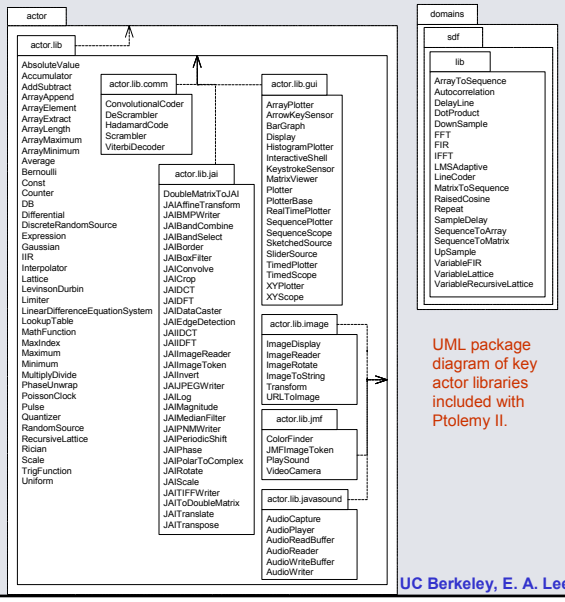
Enabled by a Behavioral Type System

- Checking behavioral compatibility of components that are composed.
- Checking behavioral compatibility of components and their frameworks.
- Behavioral subclassing enables interface/implementation separation.
- Helps with the definition of behaviorally-polymorphic components.

Enabled by Behavioral Polymorphism (1): More Re-Usable Component Libraries



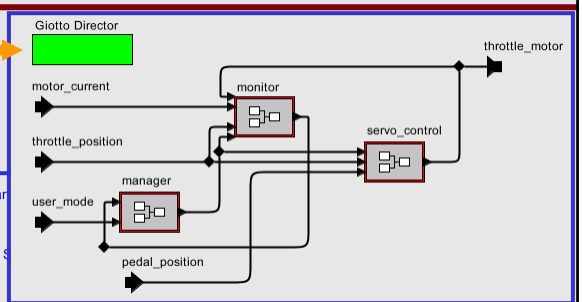
- Data polymorphic components
- Behaviorally polymorphic components



UML package diagram of key actor libraries included with Ptolemy II.

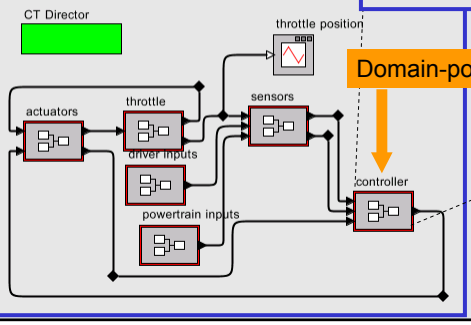
Enabled by Behavioral Polymorphism (2): Hierarchical Heterogeneity

Giotto director indicates a new model of computation.



Heterogeneous model of the UC Berkeley Vehicle Dynar
Electronic Throttle Controller.

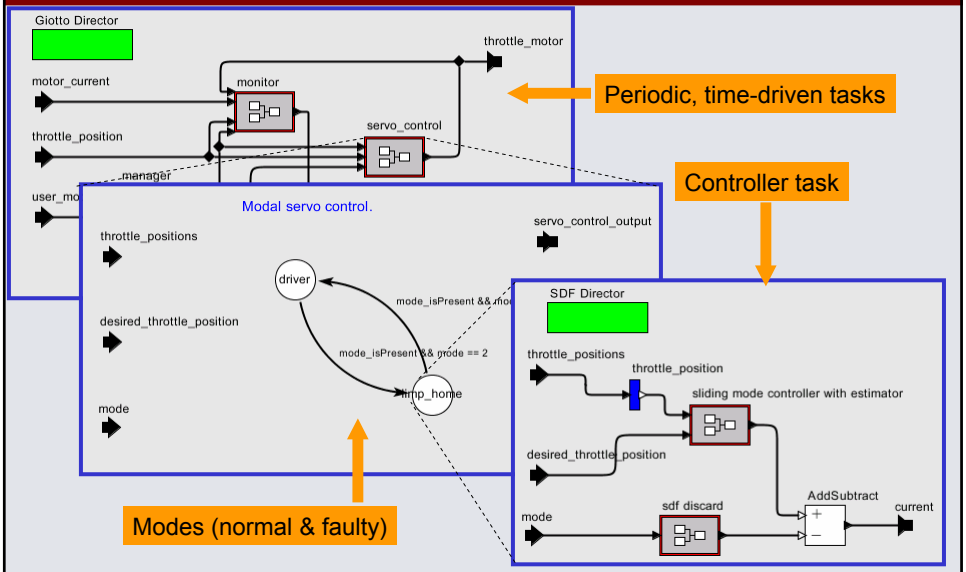
by Paul Griffiths, Christoph Kirsch, Tunc Simsek, Jason S
Last updated January 15, 2002



Domain-polymorphic component.

Domains can be nested and mixed.

Enabled by Behavioral Polymorphism (3): Modal Models



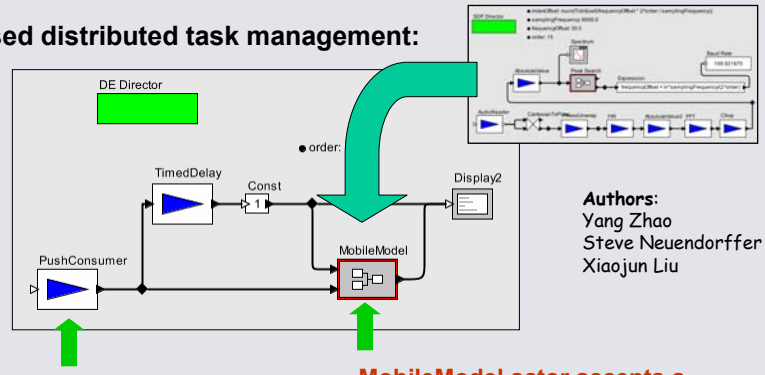
Periodic, time-driven tasks

Controller task

Modes (normal & faulty)

Enabled by Behavioral Polymorphism (4): Mobile Models

Model-based distributed task management:



Authors:
Yang Zhao
Steve Neuendorffer
Xiaojun Liu

PushConsumer actor receives pushed data provided via CORBA, where the data is an XML model of a signal analysis algorithm.

MobileModel actor accepts a StringToken containing an XML description of a model. It then executes that model on a stream of input data.

Data *and* behavioral type safety will help make such models secure

Conclusion – What to Remember

- Actor-oriented design
  - concurrent components interacting via ports
- Models of computation
  - principles of component interaction
- Behavioral types
  - a practical approach to verification and interface definition
- Behavioral polymorphism
  - defining components for use in multiple contexts

http://ptolemy.eecs.berkeley.edu
http://chess.eecs.berkeley.edu