Mission of Chess

To provide an environment for graduate research on the design issues necessary for supporting next-generation embedded software systems.

- Model-based design
- Tool-supported methodologies

For
- Real-time
- Fault-tolerant
- Robust
- Secure
- Heterogeneous
- Distributed

Software

We are on the line to create a "new systems science" that is at once computational and physical.

The fate of computers lacking interaction with physical processes.
A Traditional Systems Science – Feedback Control Systems

- Models of continuous-time dynamics
- Sophisticated stability analysis
- But not accurate for software controllers

Discretized Model – A Step Towards Software

- Numerical integration techniques provided sophisticated ways to get from the continuous idealizations to computable algorithms.
- Discrete-time signal processing techniques offer the same sophisticated stability analysis as continuous-time methods.
- But it’s still not accurate for software controllers

In general, \( z \) is an \( N \)-tuple, \( z = (z_1, \ldots, z_N) \), where \( z_i \in \mathbb{R} \). The derivative of an \( N \)-tuple is simply the \( N \)-tuple of derivatives, \( z' = (z_1', \ldots, z_N') \). We know from calculus that

\[
\frac{dz(t)}{dt} = \lim_{\delta \to 0} \frac{z(t + \delta) - z(t)}{\delta}
\]

and so, if \( \delta > 0 \) is a small number, we can approximate this derivative by

\[
\frac{dz(t)}{dt} \approx \frac{z(t + \delta) - z(t)}{\delta}
\]

Using this for the derivative in the left-hand side of (5.50) we get

\[
z(t + \delta) - z(t) = \delta g(z(t), u(t)).
\]
Hybrid Systems – Reconciliation of Continuous & Discrete

- UCB researchers have contributed hugely to the theory and practice of blended discrete & continuous models.

- But it’s still not accurate for software controllers.

Timing in Software is More Complex Than What the Theory Deals With

An example, due to Jie Liu, models two controllers sharing a CPU under an RTOS. Under preemptive multitasking, only one can be made stable (depending on the relative priorities). Under non-preemptive multitasking, both can be made stable.

Where is the theory for this?
Another Traditional Systems Science - Computation, Languages, and Semantics

Everything “computable” can be given by a terminating sequential program.

- Functions on bit patterns
- Time is irrelevant
- Non-terminating programs are defective

\[ f : States \rightarrow States \]

States = Bits*

results + state out

Current fashion – Pay Attention to “Non-functional properties”

- Time
- Security
- Fault tolerance
- Power consumption
- Memory management

But the formulation of the question is very telling:

How is it that when a braking system applies the brakes is any less a function of the braking system than how much braking it applies?
Processes and Process Calculi

Infinite sequences of state transformations are called “processes” or “threads.” Various messaging protocols lead to various formalisms.

In prevailing software practice, processes are sequences of external interactions (total orders).

And messaging protocols are combined in ad hoc ways.

Interacting Processes – Concurrency as Afterthought

Software realizing these interactions is written at a very low level (semaphores and mutexes). Very hard to get it right.

stalled for rendezvous

stalled by precedence

timing dependence
### Interacting Processes - Not Compositional

An aggregation of processes is not a process (a total order of external interactions). What is it?

Many software failures are due to this ill-defined composition.

### What Will Replace This Approach?

- Synchronous languages (e.g. Esterel)?
- Time-driven languages (e.g. Simulink, Giotto)?
- Push/Pull component interactions?
- Hybrid systems?
- Timed process networks?
- Discrete-event formalisms?
- Timed CSP?

We intend to find out.
Ptolemy Project within Chess

- Objective is to unify:
  - modeling
  - specification
  - design
  - programming

- Define modeling & design “languages” with:
  - syntaxes that aid understanding
  - composable abstractions
  - understandable concurrency and time
  - predictable behavior
  - robust behavior

All of these tasks are accomplished by the system designers.

Ptolemy Project Participants

Director:
- Edward A. Lee

Staff:
- Christopher Hylands
- Susan Gardner (Chess)
- Nuala Mansard
- Mary P. Stewart
- Neil E. Turtler (Chess)
- Lea Turpin (Chess)

Postdocs, Etc.:
- Joern Janneck, Postdoc
- Rowland R. Johnson, Visiting Scholar
- Kees Vissers, Visiting Industrial Fellow
- Daniel Lazaro Cuadrado, Visiting Scholar

Graduate Students:
- J. Adam Cataldo
- Chris Chang
- Elaine Cheong
- Sanjeev Kohli
- Xiaojun Liu
- Eleftherios D. Matsikoudis
- Stephen Neuendorffer
- James Yeh
- Yang Zhao
- Haiyang Zheng
- Rachel Zhou
Software Legacy of the Project

- **Gabriel (1986-1991)**
  - Written in Lisp
  - Aimed at signal processing
  - Synchronous dataflow (SDF) block diagrams
  - Parallel schedulers
  - Code generators for DSPs
  - Hardware/software co-simulators

- **Ptolemy Classic (1990-1997)**
  - Written in C++
  - Multiple models of computation
  - Hierarchical heterogeneity
  - Dataflow variants: BDF, DDF, PN
  - C/VHDL/DSP code generators
  - Optimizing SDF schedulers
  - Higher-order components

- **Ptolemy II (1996-2022)**
  - Written in Java
  - Domain polymorphism
  - Multithreaded
  - Network integrated
  - Modal models
  - Sophisticated type system
  - CT, HDF, GI, GR, etc.

Each of these served us, first-and-foremost, as a laboratory for investigating design.

- **PtPlot (1997-??)**
  - Java plotting package
- **Tycho (1996-1998)**
  - Itcl/Tk GUI framework
  - Java GUI framework
**Ptolemy Classic Example**

Relating the problem level with the implementation level

Heterogeneous, problem-level description

Heterogeneous, implementation-level description

Modeling

Synthesis

Ptolemy application developed by Uwe Trautwein, Technical University of Ilmenau, Germany
Foundations

Our contributions:
- Hierarchical Heterogeneity
- Behavioral Types
- Domain Polymorphism
- Responsible Frameworks
- Hybrid Systems Semantics
- Tagged Signal Model
- Discrete-Event Semantics
- Starcharts and Modal Model Semantics
- Continuous-Time Semantics
- Dataflow Semantics (SDF, BDF, DDF, PN, CI)

Giving structure to the notion of “models of computation”

Hierarchical Heterogeneity

In Ptolemy, the semantics of a block diagram is defined by a “director,” which is a component that the model builder places in the model. An “abstract semantics” defines the interaction across levels of hierarchy where the semantics differ.
**Actor-Oriented Design**

**Actors with Ports and Attributes**

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

- **Examples:**
  - Push/Pull
  - Time triggered
  - Process networks
  - Discrete-event systems
  - Dataflow systems
  - Publish & subscribe

*Key idea:* The model of computation is part of the framework within which components are embedded rather than part of the components themselves.

---

**Actor View of Producer/Consumer Components**

- **Basic Transport:**
  - send(0,t)
  - receiver.put(t)
  - get(0)
  - P1
  - E1
  - E2

- **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.*
Examples of Actor-Oriented Component Frameworks

- Easy5 (Boeing)
- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- 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)
- ...

Models of Computation
Principles of Model Driven Architecture

- Continuous-time models
- Dataflow
  - synchronous dataflow
  - boolean/integer dataflow
  - dynamic dataflow
  - heterochronous dataflow
- Push/pull models
- Discrete-event models
- Synchronous/reactive models
- CSP models
- Discrete-time models
- Time-triggered models (TTA, Giotto)

- Modal models are possible in all cases
Ptolemy Project Principles

Basic Ptolemy II infrastructure:

Director from a library defines component interaction semantics.

Large, domain-polymorphic component library.

Continuous-Time Models

Soft Walls Avionics System

Aircraft model
Criticality calculation
Pilot model
Bias control
Synchronous Dataflow (SDF)

SDF offers feedback, multirate, static scheduling, deadlock analysis, parallel scheduling, static memory allocation.

Parallel Scheduling of SDF Models

SDF is suitable for automated mapping onto parallel processors.

Sequential (software)  Parallel (hardware)
Other Dataflow Models
Process Networks

Detection of unknown signal (PSK in this case)

Challenge problem under DARPA Mobies (Model-based design of embedded software),

Output data sequence, at detected baud rate. (not known a priori)

Discrete-Event Models
Sensor Nets Modeling

Ptolemy II model where actor icons depict sensor range and connectivity is not shown with wires

This model shows the results of a power optimization where the green node issues a broadcast signal and the red ones retransmit to relay the signal.
Heterogeneous Models: Periodic/Time-Driven Control Inside Continuous Time

Giotto director indicates a new model of computation.

Domains can be nested and mixed.

Heterogeneous Models
Modal Controller

Periodic, time-driven tasks

Controller task

Modes (normal & faulty)
Heterogeneous Models
Hybrid Systems

HyVisual is a branded tool based on Ptolemy II designed for hybrid system modeling.

Distributed Models, Middleware and Systems of Systems

Currently, components are designed to the middleware APIs. Our objective is to define the components with middleware-polymorphic interfaces that declare precisely the assumptions and guarantees of the components.
Distributed Models Using Mobile Models

Model-based distributed task management:

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

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

A significant challenge here is achieving type safety and security.

MoML XML Schema Used to Transport Models

Ptolemy II designs are represented in XML:

```xml
...<entity name="FFT" class="ptolemy.domains.sdf.lib.FFT">
  <property name="order" class="ptolemy.data.expr.Parameter" value="order">
    </property>
  <port name="input" class="ptolemy.domains.sdf.kernel.SDFIOPort">
    ...</port>
  ...
</entity>
...<link port="FFT.input" relation="relation"/>
<link port="AbsoluteValue2.output" relation="relation"/>
...```
Verification & Validation
What Many People Say They Want

A button that they can push that when pushed will tell them whether or not the design is correct.

Behavioral Types –
A More 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
  - domain type signatures
- Components are
  - data polymorphic
  - domain polymorphic
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.

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

**Key to Domain Polymorphism:**

Receiver Object Model
Behavioral Type System

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

Conclusion - What to Remember

- A new systems science
  - physical + computational
- Actor-oriented design
  - concurrent components interacting via ports
- Models of computation
  - principles of component interaction
- Hierarchical heterogeneity
  - principled mixing of models of computation
- Behavioral types
  - a practical approach to verification and interface definition
- Domain polymorphism
  - defining components for use in multiple contexts

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