Abstract

Model-based design uses models of systems as the specification for software behavior and synthesizes executable software from those models. As with any successful engineering discipline, design techniques must evolve to support development, maintenance, and evolution of designs, and these techniques must be able to handle designs of realistic size and complexity. The discipline of software engineering provides techniques, such as object-oriented design, static analysis, and formal verification, for software designs. This talk explores techniques that support development, maintenance, and evolution of models. In particular, we discuss model transformation, model ontologies, and multimodeling, with applications to construction of large models (e.g. MapReduce), model optimization, and consistency management across multiple models.
The Ptolemy Project

Sponsors:
- Government
  - National Science Foundation
  - Army Research Office
  - Air Force Research Lab
  - Air Force Office of Scientific Research
  - State of California Micro Program
- Industry
  - Agilent
  - Bosch
  - HSBC
  - Lockheed-Martin
  - National Instruments
  - Toyota

History:
The Ptolemy project was started in 1990, though its mission and focus has evolved considerably. An open-source, extensible software framework (Ptolemy II) constitutes the experimental laboratory.

Staffing:
- 1 professor
- 9 graduate students
- 2 postdocs
- 3 staff
- several visitors

Our Premise:
Components are Actors rather than Objects

The established: Object-oriented:
- Class name
- Data
- Methods
- Call
- Call return
- What flows through an object is sequential control
- Things happen to objects

The alternative: Actor oriented:
- Actor name
- Data (state)
- Parameters
- Ports
- What flows through an object is evolving data
- Actors make things happen

Input data  Output data
Ptolemy II: Our Open-Source Laboratory for Experiments with Actor-Oriented Design
http://ptolemy.org

Concurrency management supporting dynamic model structure.
Director from a library defines component interaction semantics
Large, behaviorally-polymorphic component library.
Visual editor supporting an abstract syntax
Type system for transported data

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

- Model transformation
  - with Thomas Feng

- Model ontologies
  - with Mankit Leung & Thomas Mandl (Bosch)

- Multimodeling
  - with Chihong Cheng (TU Munich), Trip Denton (Lockheed), Thomas Huining Feng, Edward Jones (Lockheed-Martin), Reinhard von Hanxleden (Christian-Albrechts-Univ. Kiel),

Model Transformation
Inspirations and Influences

- AGG [Taentzer, 1999]
- AToM3 [Lara, Vangheluwe, 2002]
- FUJABA [Nickel, Niere, Zündorf, 2000],
- GReAT [Agrawal, Karsai, Shi, 2003]
- OMG MOF QVT (Query/Views/Transformations)
- PROGRES [Schürr, Winter, Zündorf, 1995]
- VIATRA2 [Balogh, Varró, 2006]
Key Prior Work from the Ptolemy Project:
1. Higher-Order Components

Examples of HoCs:
- Replicate a submodel over an array of inputs
- Structured dataflow components (case, iterate, recursion)
- Mobile models
- Parameterizing models with models
- Lifecycle models

Key Prior Work from the Ptolemy Project:
2. Composition Languages

*Big Systems with Small Descriptions*

```plaintext
System is {
  Matrix(Component(2),20,3);
}

Component is {
  param n;
  port in[n*2+1];
  port out[n*2+2];
} in {
  Blue(n, in[1..n*2], out[1..n*2]);
  Green(n, in[n*2+1], out[n*2+1]);
}
```

We have released a specification language that we call “Ptalon” for such systems, integrated into Ptolemy II [Cataldo 2006]
Demo: Pattern Matching and Graph Transformation

Model transformation workflow specifies iterative graph rewriting to transform the top-right model into the bottom-left model.

Applications

- Model optimization
  - Support programming idioms
- Scalable model construction
  - Adapt to problem size or parallelism
- Product families
  - A single model transforms to multiple products
- Design refactoring
  - Common model transformations
- Workflow automation
  - Configuration, composition, testing, versioning
Scalable Model Construction:
MapReduce Pattern [Dean, Ghemawat, 2004]

This pattern is intended to exploit parallel computing by distributing computations that fit the structure. The canonical example constructs an index of words found in a set of documents.

A MapReduce Model in Ptolemy II

Inputs of web documents
“contents of the first document”
“contents of the second document”
... “contents of the last document”

End of all documents
false false ... true

Word-value pairs
{“contents”, 1}
{“of”, 1}
... {“document”, 1}

Merged word-counting outputs
{“contents”, 3}
{“first”, 1}
... {“document”, 3}
A configurable application with $m$ Map actors and $n$ Reduce actors has $O(m \times n)$ connections. Inside each actor, parameters need to be configured as well.

Observations

- The visual representation is only helpful for small models.
- The construction process by visual editing is tedious and error prone.
- Adapting the size of the model to varying numbers of compute resources by visual editing is unreasonable.
- Replacing the Map or Reduce actors with alternative functions by visual editing is also not reasonable.
Transformation Rule

Pattern Replacement

A transformation rule to connect a Map actor to a Reduce actor

Map and Reduce are matchers to match arbitrary actors with the specified ports.

A Set of Transformation Rules

CreateMap CreateReduce LinkMapReduce

DE Director

document Split Map Reduce

sendOfTask WaitingStop

merge result

Lee, Berkeley 17

Lee, Berkeley 18
Model is built by repeatedly applying transformation rules drawn from a set

Workflow mechanisms:

- Priority (AGG, AToM3)
- Imperative programs (PROGRES)
- Story diagrams (FUJABA)
- Control flow and data flow (GReAT)
- Abstract State Machines (VIATRA2)
- Ptolemy II models (our approach)

The TransformationRule Actor

Encapsulates a transformation rule

Input:

- \texttt{modelInput} – actor tokens that contain model fragments

Output:

- \texttt{modelOutput} – actor tokens that contain transformation results
- \texttt{matched} – whether the last transformation was successful

This actor may be used in nearly any Ptolemy model (dataflow, process networks, discrete-events, etc.)
Ptolemy II model defines a workflow

This Ptolemy II model creates and executes a MapReduce application for a parameterized number of machines.

Using Other MoCs for Workflows

Here we have used Event-Relationship graphs [Schruben 83] to specify the workflow logic (Ptolemy II domain created by Thomas Huining Feng).
Model Transformation Summary

- Patterns, replacements, and workflows are all expressed using the same target modeling language(s) as the application.
- Mixing of modeling languages / models of computation (via the Ptolemy II framework) is supported in the application, patterns, replacements, and workflows.
- Visual syntaxes become more scalable and flexible.
- Applications
  - Model optimization
  - Scalable model construction
  - Product families
  - Design refactoring
  - Workflow automation
  - …

Outline

- Model transformation
  - with Thomas Feng
- Model ontologies
  - with Mankit Leung & Thomas Mandl (Bosch))
- Multimodeling
  - with Thomas Feng, Trip Denton (Lockheed), Reinhard von Hanxleden (Christian-Albrechts-Univ. Kiel), Chihong Cheng (TU Munich))
Type Lattices

- A *lattice* is a partially ordered set (poset) where every subset has a least upper bound (LUB) and a greatest lower bound (GLB).

- Modern type systems (including the Ptolemy II type system, created by Yuhong Xiong) are based on efficient algorithms for solving inequality constraints on lattices.

Data Ontologies

- Components in a model (e.g. parameters, ports) can have properties drawn from a lattice.

- Components in a model (e.g. actors) can impose constraints on property relationships.

- The type system infrastructure can infer properties and detect errors.
A Simple Example: Constant / Nonconstant

- A port is nonconstant (time-varying), constant, or unknown
- By default, actors impose the constraint that all output ports are greater than or equal to each input port (in the lattice)
- Actor “helpers” can override the constraint definition for any actor.

Property Inference in Action

In the system above, the green indicates constant data and is inferred from constraints imposed by the actors.

Thanks to Thomas Mandl, Research & Technology Center, Bosch, Palo Alto.
Property Inference in Action

In the system above, one of the constant sources has been replaced with a non-constant source. This affects the inferred properties downstream.

Thanks to Thomas Mandl, Research & Technology Center, Bosch, Palo Alto.

Outline

- Model transformation
  with Thomas Feng

- Model ontologies
  with Mankit Leung & Thomas Mandl (Bosch)

- Multimodeling
  with Thomas Feng, Trip Denton (Lockheed), Reinhard von Hanxleden (Christian-Albrechts-Univ. Kiel), Chihong Cheng (TU Munich)
Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling**: hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.

- **multi-view modeling**: distinct and separate models of the same system are constructed to model different aspects of the system.

- **meta modeling**: use of models together with models of the modeling language.

Hierarchical Multimodeling

Hierarchical compositions of models of computation. Maintaining temporal semantics across MoCs is a key challenge. The example here was developed in a collaborative project with Lockheed-Martin.
Background on Hierarchical Multimodeling

- Statecharts [Harel 87]
- Ptolemy Classic [Buck, Ha, Lee, Messerschmitt 94]
- SyncCharts [André 96]
- *Charts [Girault, Lee, Lee 99]
- Colif [Cesario, Nicolescu, Guathier, Lyonnard, Jerraya 01]
- Metropolis [Goessler, Sangiovanni-Vincentelli 02]
- Ptolemy II [Eker, et. al. 03]
- Safe State Machine (SSM) [André 03]
- SCADE [Berry 03]
- ForSyDe [Jantsch, Sander 05]
- ModHelX [Hardebolle, Boulanger07]

Simple Traffic Light Example in Statecharts

**Case study**

- Pred: pedestrian red signal
- Pgrn(0): turn pedestrian green off
- Cgm: car green
- Sec: one second time
- 2 Sec: two seconds time
- Pgo/Pstop: pedestrian go/stop
Traffic Light Example in Ptolemy II

Whereas Statecharts lumps together the state machine semantics and the concurrency model, Ptolemy II separates these.

Here we have chosen the SR Director, which realizes a true synchronous fixed point semantics.

Concurrent State Machines in Ptolemy II

In Ptolemy II, we have implemented an SR Director (for synchronous concurrent models) and an FSM Director (for sequential decision logic). Rather than combining them into one language (like Statecharts), Ptolemy II supports hierarchical combinations of MoCs.
Stepping Outside Statecharts: Modeling the Environment

The above model places the TrafficLight model in a discrete-event testbench that clocks the light and injects faults according to a stochastic model.

What Makes This Possible: The Ptolemy II Actor Abstract Semantics

- Abstract Syntax
- Concrete Syntax
- Type System
- Abstract Semantics
- Concrete Semantics
How Does This Work?  
Ptolemy II Actor Abstract Semantics

Actions invoked on an actor by a director:
- Preinitialization
- Initialization
- Execution
- Finalization

E.g., Partial evaluation (esp. higher-order components), set up type constraints, etc. Anything that needs to be done prior to static analysis (type inference, scheduling, ...)

Lee, Berkeley 39
How Does This Work?
Ptolemy II Actor Abstract Semantics

Actions invoked on an actor by a director:
- Preinitialization
- Initialization
- Execution
- Finalization

E.g., Initialize actors, produce initial outputs, etc.
E.g., set the initial state of a state machine.
Initialization may be repeated during the run (e.g., if the reset parameter of a transition is set and the destination state has a refinement).

In fire(), an FSM first fires the refinement of the current state (if any), then evaluates guards, then produces outputs specified on an enabled transition. In postfire(), it postfires the current refinement (if any), executes set actions on an enabled transition, and takes the transition.
How Does This Work?  
Ptolemy II Actor Abstract Semantics

Actions invoked on an actor by a director:
- Preinitialization
- Initialization
- Execution
- Finalization

A Consequence of Our Abstract Semantics: 
Behavioral Polymorphism

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

- Behavioral polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a synchronous/reactive 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 rendezvous, 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
Hierarchical Models are Behaviorally Polymorphic

Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling**: hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.

- **multi-view modeling**: distinct and separate models of the same system are constructed to model different aspects of the system.

- **meta modeling**: use of models together with models of the modeling language.
Multi-View Modeling:
Distinct and separate models of the same system are constructed to model different aspects of the system.

Background on Multi-View Modeling

- Ptolemy Classic [Buck, Ha, Lee, Messerschmitt 94]
- UML [Various, 90s]
- Model-integrated computing [Sztipanovits, Karsai, Franke 96]
- Colif [Cesario, Nicolescu, Guathier, Lyonnard, Jerraya 01]
- Metropolis [Goessler, Sangiovanni-Vincentelli 02]
- KIEL [Prochnow, von Hanxleden 07]
Model synthesis is one way to maintain model consistency

But Model Synthesis is not always possible. Constructing a Deployment Model

This is the top level of a deployment model, which maps the car light and pedestrian light logic into two distinct compute platforms that communicate via a wireless link. The same models are used for the functional logic, leveraging actor-oriented classes in Ptolemy II.
Inside The Car Light Model

The above model shows the construction of a radio packet for transmission on the wireless link. Inside, it eventually uses the same behavioral model of the traffic light, so changing the behavior in one model is automatically reflected in the other.

Actor-Oriented Classes

[Lee, Liu, Neuendorffer 07]

A class definition (right) has instances in multiple models. Changes to the class definition automatically propagate to the instances.
Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling**: hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.

- **multi-view modeling**: distinct and separate models of the same system are constructed to model different aspects of the system.

- **meta modeling**: use of models together with models of the modeling language.

---

Metamodel of Ptolemy II Abstract Syntax

Using GME (from Vanderbilt) an abstract syntax is specified as an object model (in UML) with constraints (in OCL), or alternatively, with MOF.

Such a spec can be used to synthesize visual editors, models transformers, and code generators.

Meta-model of Ptolemy II abstract syntax, constructed in GME by H. Y. Zheng.
Summary: Model Engineering
the Berkeley View

- Model transformation
  - Scaling to large models
  - Model optimization
  - Maintenance of product families
  - Workflow automation
  - Design refactoring
  - Enhances usability of visual syntaxes

- Model ontologies
  - Systematic management of model properties

- Multimodeling
  - Hierarchical multimodeling
  - Multi-view modeling
  - Metamodeling