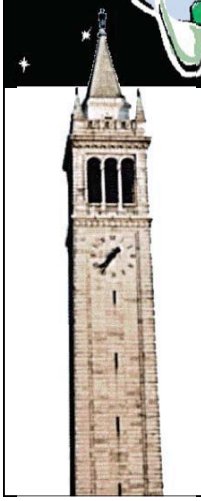
Ninth Biennial Ptolemy Miniconference

February 16, 2010 http://ptolemy.eecs.berkeley.edu/ptconf

8:00 am to 8:30 am	Continental Breakfast
8:30 am to 8:45 am	*Ptolemy Miniconferences*, **Edward Lee** (*Berkeley*)
8:45 am to 9:10 am	*Distributed Execution Architectures in Kepler* **Jianwu Wang,** Daniel Crawl, Ilkay Altintas, Chad Berkley, & Matthew B. Jones (*San Diego Supercomputer Center and UC Santa Barbara*)
9:10 am to 9:35 am	*Modeling Distributed Real-Time Systems with Ptolemy II*, **Patricia Derler,** Jia Zou, Slobodan Matic, John Eidson (*Berkeley*)
9:35 am to 9:55 am	*Semantics of Modal Models in Ptolemy*, **Stavros Tripakis** & Edward A. Lee (*Berkeley*)
9:55 am to 10:15 am	Break
10:15 am to 11:10 am	*Static Analysis using the Ptolemy II Ontologies Package*, **Charles Shelton,** Elizabeth Latronico, & Ben Lickly (*Bosch & Berkeley*)
11:10 am to 11:35 am	*To Meet or Not to Meet the Deadline*, **Jan Reineke,** Isaac Liu, Gage Eads, Stephen Edwards, Sungjun Kim, Hiren Patel (*Berkeley, Columbia, Waterloo*)
11:35 am to 12:15 pm	*Poster Tweets*
12:15 pm to 2:30 pm	Working Lunch and Poster Session
2:30 pm to 2:55 pm	*The Dataflow Interchange Format: Towards Co-Design of DSP-oriented Dataflow Models and Transformations*, **Shuvra S. Bhattacharyya** (*Univ. of Maryland*)
2:55 pm to 3:20 pm	*Workflow Recovery for Different Models of Computation and Models of Provenance*, **Sven Koehler,** Bertram Ludaescher, Timothy McPhillips, Anandarup Sarkar (*UC Davis*)
3:20 pm to 3:45 pm	*Design, Analysis, and Implementation of Static Dataflow Models for Hardware Targets*, **Kaushik Ravindran, Murali Parthasarathy et. al**, (*National Instruments*)
3:45 pm to 4:00 pm	Break
4:00 pm to 4:25 pm	*Kepler/G-Pack: A Kepler Package Using the Google Cloud for Interactive Scientific Workflows*, **Gongjing Cao, Lei Dou, Quinn Hart, Bertram Ludaescher**, (*UC Davis*)
4:25 pm to 4:50 pm	*Context Aware Actors*, **Anne H.H. Ngu** & George Chin Jr. (*Texas State Univ. & Pacific NW National Lab*)
4:50 pm to 5:15 pm	*Modular Code Generation,* **Dai Bui** & Stavros Tripakis (*Berkeley*)
5:15 pm to 5:30 pm	*The Ptolemy Project: Advancing System Design*, **Edward A. Lee** (*Berkeley*)
6:00 pm to 8:00 pm	Reception and Dinner, The Faculty Club

Posters		
Gage Eads	Berkeley	*Deadline Instructions in a PRET Architecture*
Shanna-Shaye Forbes	Berkeley	*Error Handling in Model-Based design for Real-Time Systems*
Soheil Ghiasi, Matin Hashemi	UC Davis	*Malleable Dataflow Specification: An Essential Ingredient for Resource-Scalable Implementations*
Isaac Liu, Jan Reineke	Berkeley	*A PRET Architecture Supporting Concurrent Programs with Composable Timing Properties*
Slobodan Matic, Ilge Akkaya, and John Eidson	Berkeley	*The Distributed Power System Test Case for Distributed Real-Time Systems*
Christian Motika, Hauke Fuhrmann, Miro Spönemann Reinhard von Hanxleden	Kiel University	*KIELER Actor Oriented Modeling*
Deepak Shankar	Mirabilis	*Using Ptolemy/VisualSim for Internet-based model Sharing and Communication.*
Chris Shaver	Berkeley	*Alternative Syntactic Representations of Graph-Based Models*
Chris Shaver, Dai Bui**,** Stavros Tripakis	Berkeley	*Multidimensional Dataflow Models*
Elizabeth Latronico, Charles Shelton, Ben Lickly	Bosch & Berkeley	*Lattice Composition for Ontology Analysis*
Ben Lickly, Charles Shelton, Elizabeth Latronico	Bosch & Berkeley	*Practical Ontologies with Infinite Lattices*
Andreas Thuy	University of Paderborn	*Towards flexible and robust cyber-physical-systems through self organization*
Stavros Tripakis, Marc Geilen, Maarten Wiggers	Berkeley, TU Eindhoven	*The Earlier the Better: A Theory of Timed Actor Interfaces*
Mike Wirthlin	Brigham Young University	*Automated Bit-Width Analysis Using Ptolemy*
Michael Zimmer	Berkeley	*IEEE 1588 Time Synchronization for Real-Time Distributed Systems*
Jia Zou, Jeff Jensen, Slobodan Matic	Berkeley	*The Tunneling Ball Device (TBD) Test Case for Real-Time Systems*
Jia Zou, Slobodan Matic, John Eidson	Berkeley	*From PTIDES to PtidyOS: Programming Distributed Real-Time Embedded Systems*

Ptolemy Miniconferences



Edward A. Lee

Robert S. Pepper Distinguished Professor

Ninth Biennial Ptolemy Miniconference

February 16, 2011
Berkeley, CA, USA

Ptolemy Project

February, 2011

Staff:
- Christopher Brooks
- Edward A. Lee (PI)
- Stavros Tripakis
- Mary P. Stewart

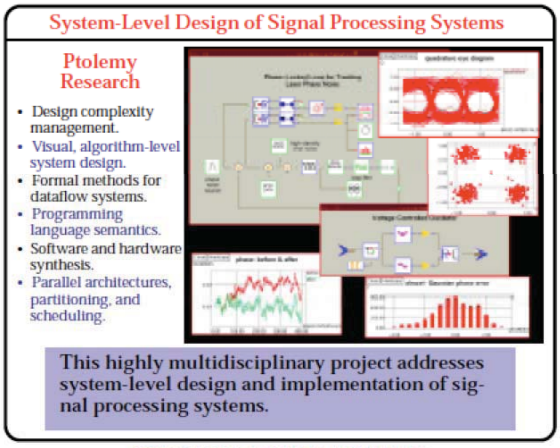
Postdocs:
- Patricia Derler
- Slobodan Matic
- Eleftherios Matsikoudis
- Jan Reineke

Grad Students:
- Ilge Akkaya
- Dai Bui
- Shanna-Shaye Forbes
- Ben Lickly
- Isaac Liu
- Chris Shaver
- Jia Zou
- Mike Zimmer

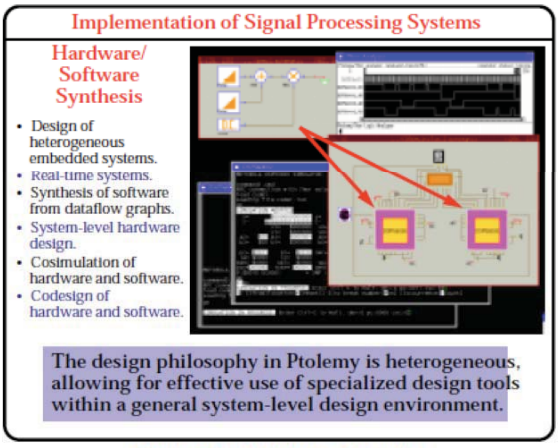
Visiting Scholars:
- Hugo Andrade
- Janette Cardoso
- John Eidson

Photo by Chamberlain Fong

The 1st Biennial Ptolemy Miniconference: 1995



The 1st Biennial Ptolemy Miniconference: 1995

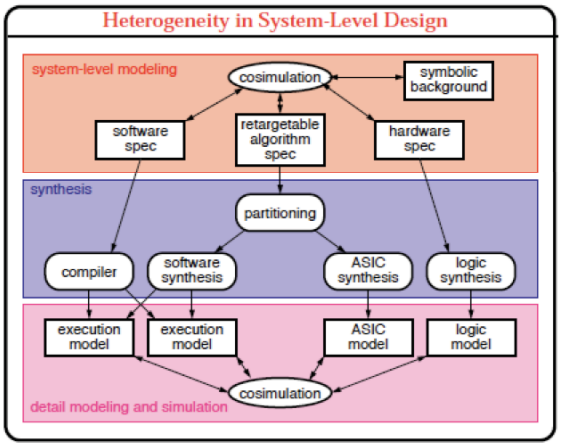


The 1st Biennial Ptolemy Miniconference: 1995

The Ptolemy Project

Shuvra Bhattacharyya
Joseph T. Buck
Wan-Teh Chang
Brian L. Evans
Steve X. Gu
Sangjin Hong
Christopher Hylands
Asawaree Kalavade
Alan Kamas
Allen Lao
Bilung Lee
Edward A. Lee
David G. Messerschmitt
Praveen K. Murthy
Thomas M. Parks
José Luis Pino
Farhana Shiekh
S. Sriram
Juergen Teich
Warren W. Tsai
Patrick J. Warner
Michael C. Williamson

A AT BERKELEY



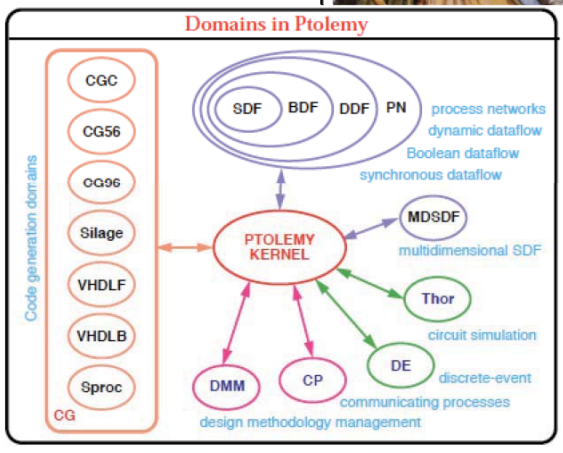
UNIVERSITY OF CALIFORNIA AT BERKELEY

The 1st Biennial Ptolemy Miniconference: 1995

The Ptolemy Project

Shuvra Bhattacharyya
Joseph T. Buck
Wan-Teh Chang
Brian L. Evans
Steve X. Gu
Sangjin Hong
Christopher Hylands
Asawaree Kalavade
Alan Kamas
Allen Lao
Bilung Lee
Edward A. Lee
David G. Messerschmitt
Praveen K. Murthy
Thomas M. Parks
José Luis Pino
Farhana Shiekh
S. Sriram
Juergen Teich
Warren W. Tsai
Patrick J. Warner
Michael C. Williamson

A AT BERKELEY



UNIVERSITY OF CALIFORNIA AT BERKELEY

The 1st Biennial Ptolemy Miniconference: 1995

The Ptolemy Project

Shuvra Bhattacharyya
Joseph T. Buck
Wan-Teh Chang
Brian L. Evans
Steve X. Gu
Sangjin Hong
Christopher Hylands
Asawaree Kalavade
Alan Kamas
Allen Lao
Bilung Lee
Edward A. Lee
David G. Messerschmitt
Praveen K. Murthy
Thomas M. Parks
José Luis Pino
Farhana Shiekh
S. Sriram
Juergen Teich
Warren W. Tsai
Patrick J. Warner
Michael C. Williamson

A AT BERKELEY

Where to From Here?

- Real-time scalable computing.
- Scalable embedded systems design.
- Design migration from abstract to concrete.
- Formal methods based on partial orders.
- Hybrid systems: combining FSM with dataflow.
- Modeling and analysis of random systems.
- Design of nondeterminate systems.
- Complexity management.
- Design visualization and documentation.
- Partial evaluation and incremental compilation.
- Models for back-end signal interpretation.
- Heterogeneous scheduling.

UNIVERSITY OF CALIFORNIA AT BERKELEY

Organizational

Staff

Diane Chang, administrative assistant
Kevin Chang, programmer
Christopher Hylands, programmer analyst
Edward A. Lee, professor and PI
Mary Stewart, programmer analyst

Postdocs

Praveen Murthy
Seehyun Kim
Raja Nagarajan
John Reekie
Dick Stevens (on leave from NRL)

Students

Sunil Bhave
Cliff Cordeiro
John Davis
Stephen Edwards
Ron Galicia
Mudit Goel
Michael Goodwin
Luis Gutierrez
Bilung Lee

Michael C. Williamson
Yuhong Xiong

Key Outside Collaborators

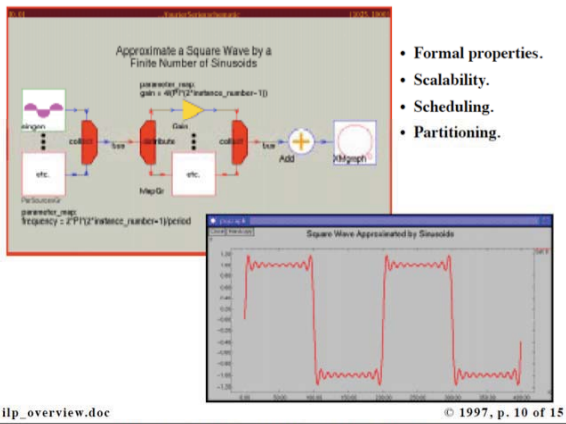
Shuvra Bhattacharyya (Hitachi)
Joseph T. Buck (Synopsys)
Brian L. Evans (UT Austin)
Soonhoi Ha (Seoul N. Univ.)
Tom Lane (SSS)
Thomas M. Parks (Lincoln Labs)
José Luis Pino (Hewlett Packard)

ilp_overview.doc

UNIVERSITY OF CA

The 2nd Biennial Ptolemy Miniconference: 1997

Visual Design

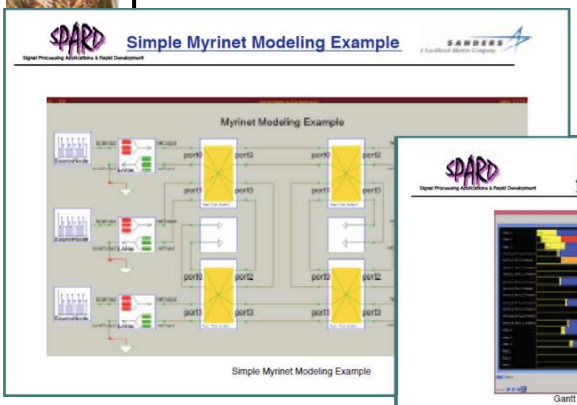


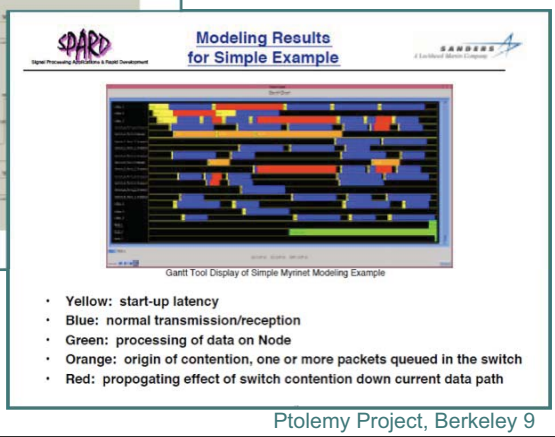
- Formal properties.
- Scalability.
- Scheduling.
- Partitioning.

ilp_overview.doc © 1997, p. 10 of 15

UNIVERSITY OF CALIFORNIA AT BERKELEY

The 2nd Biennial Ptolemy Miniconference: 1997

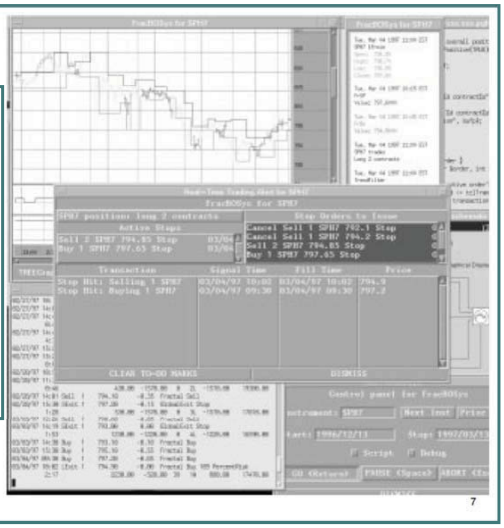




High-Performance Scalable Computing (HPSC) modeling by Sanders, a Lockheed-Martin Company.

The 2nd Biennial Ptolemy Miniconference: 1997

Live trading example



Applications of Ptolemy in Securities Trading

or,
Playing the Markets with Ptolemy

Tom Lane

Structured Software Systems, Inc.

The 2nd Biennial Ptolemy Miniconference: 1997

Modeling Free Space Optoelectronic Systems Using Ptolemy

Steven P. Levitan
Donald M. Chiarulli
Tim P. Kurzweg
Mark A. Rempel
Departments of Electrical Engineering & Computer Science
steve@ee.pitt.edu
http://kona.ee.pitt.edu/steve
University of Pittsburgh

Philippe J. Marchand
Chi Fan
Fredrick B. McCormick
Department of Electrical & Computer Engineering
pmarchand@ucsd.edu
http://soliton.ucsd.edu
University of California, San Diego

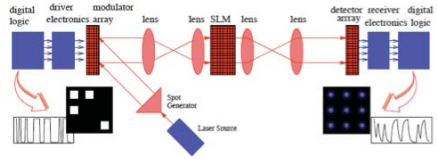
Funding: National Science Foundation- MIP-9421777

Overview

Chatoyant is a computer aided design tool for the design of *Free Space Optoelectronic Information processing (FSOI) Systems.*

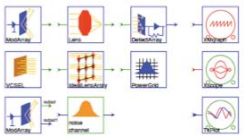
- Simulation - Analysis - Synthesis - Interface

Enable the modeling of FSOI systems without costly prototyping



chatoyant

Chatoyant Stars in Ptolemy



Modulators	Detectors	Lenses	Lenslets
Area	Detector Size	Focal Length	Focal Length
Spacing	Detector Spacing	Diameter	Diameter
Lambda	Distance	Distance	Distance
Spotsize	x, y offsets	x, y offsets	x, y offsets
Filename	Radius of Integration		Spacing
Gauss/Ray	R, C, A		Number

chatoyant

The 2nd Biennial Ptolemy Miniconference: 1997

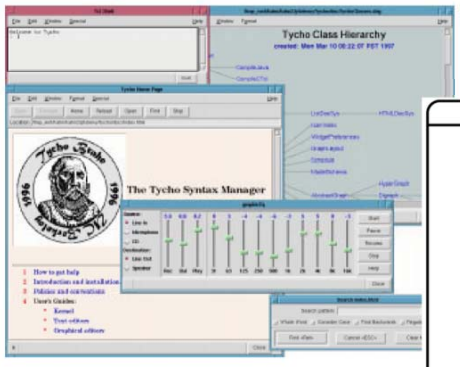
Tycho

Christopher Hylands
Edward A. Lee
H. John Reekie

Contributors:
Kevin Chang
Wan-Teh Chang
Cliff Cordeiro
Wei-Jen Huang
Joel King
Farhana Sheikh
Mario Jorge Silva

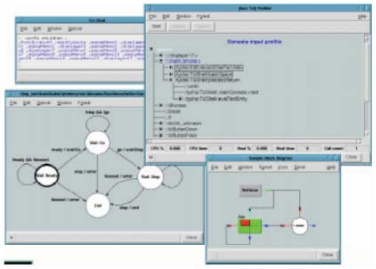
UNIVERSITY OF CALIFORNIA AT BERKELEY

Tycho (1)



UNIVERSITY OF CALIFORNIA AT BERKELEY

Tycho (2)

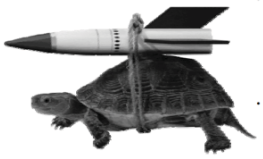


UNIVERSITY OF CALIFORNIA AT BERKELEY

3rd Biennial PtConf 1999

The switch to Ptolemy II

Modeling in Java ?!?!?!?!?



- Choosing the best modeling technique can have a far bigger impact than using a faster modeling tool.
- Mixing modeling techniques permits multi-domain modeling using the best available modeling techniques.

- Threads, objects, and UI infrastructure helps with both.
- Network integration of Java promotes sharing of modeling methods.
- Java performance and infrastructure is rapidly improving.

Ptolemy Classic vs Ptolemy II

C++	Java
Mature platform	Experimental
Does code generation	All Java (now)
Monolithic tool	Modular packages
Standalone	Networked
Sequential	Multi-threaded
GUI-centric	Applet-centric
Ad-hoc development	Good software practice
Dynamically linked	Reflective
Astronomical lexicon	Boring lexicon

Ptolemy Miniconference - 1

The Ptolemy Project

Heterogeneous Modeling and Design

Principal Investigator
Edward A. Lee

Staff
Jennifer Basler
Christopher Hylands
Mary P. Stewart

Postdocs/Researchers
Bart Kienhuis
James Lundblad
John Reekie

Students
John Davis, II
Ron Galicia
Mudit Goel
Bilung Lee
Michael Leung
Jie Liu
Xiaojun Liu
Lukito Muliadi
Steve Neuendorffer
Neil Smyth
Jeff Tsay
William Wu
Yuhong Xiong

Ptolemy Miniconference - 1

3rd Biennial PtConf 1999

Algorithm Analysis and Mapping Environment for Adaptive Computing Systems

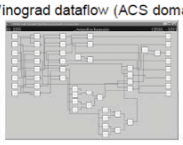
Eric Pauer, Cory Myers, Ken Smith, and Paul Fiore
er,cory.jmsmtn,pmorej@sanders.com
Lockheed Martin Company
Nashua, NH 03061

SANDERS
A Lockheed Martin Company

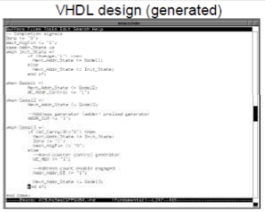
Third Bi-Annual Ptolemy Miniconference - 1999

ACS Domain - CGFPGA Target

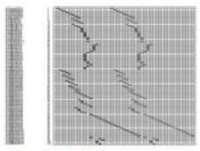
Winograd dataflow (ACS domain)



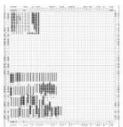
VHDL design (generated)



CGFPGA target yields: VHDL design and schedule



The results are sent to synthesis and place/route, yielding complete FPGA implementation!



Dataflow/Hardware schedule

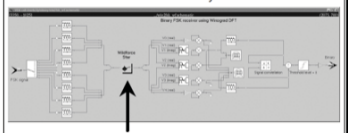
SANDERS
A Lockheed Martin Company

Adaptive Computing System Design and Implementation using the ACS Domain

PUBS-98-MC1_001-P
2/12/99 11

Hardware-in-the-loop

SDF Galaxy



SDF Wildforce star executes complete FPGA design in hardware on Annapolis Wildforce FPGA board

Adaptive Computing System Design and Implementation using the ACS Domain

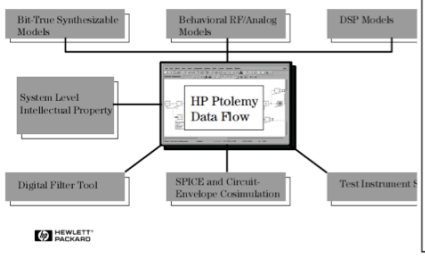
3rd Biennial
PtConf 1999

Cosimulating Synchronous DSP Designs with Analog RF Circuits

José Luis Pino and Khalil Kalbasi

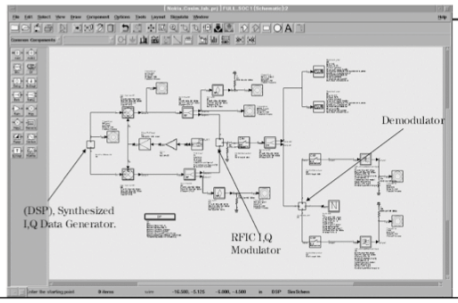
HEWLETT PACKARD

HP Ptolemy



HEWLETT PACKARD

Example: 16 QAM Tx/Rx



Ptolemy Project, Berkeley 15

4th

2001

4th 2001

ESPADON EUROFINDER

Rapid Prototyping of RADAR Signal Processing Systems using Ptolemy Classic

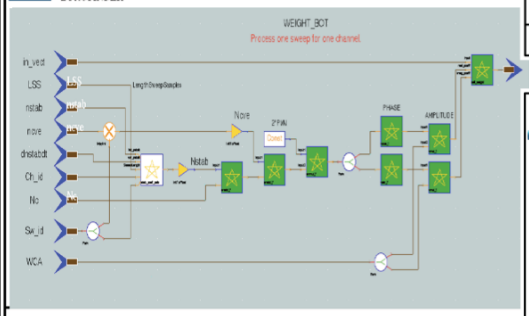
Ptolemy MiniConference UCB

Denis Aulagnier, Patrick Meyer, Hans Schurer, Xavier Warzee, THALES

THALES Metra BAe Dynamics BAE SYSTEMS

ESPADON EUROFINDER

BENCHMARK FINAL DESIGN (2)



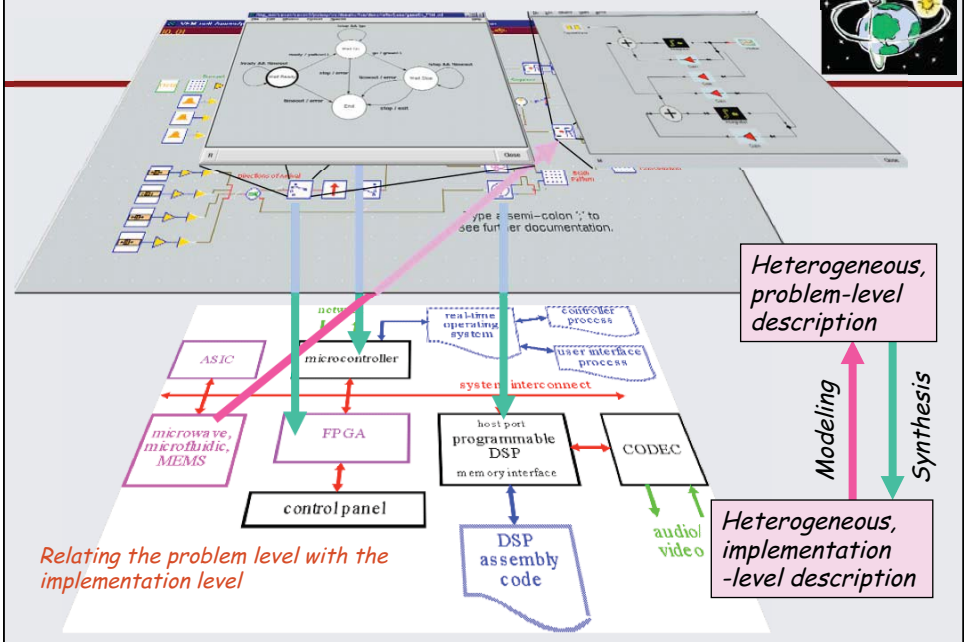
THALES Metra BAe Dynamics BAE SYSTEMS

ESPADON EUROFINDER

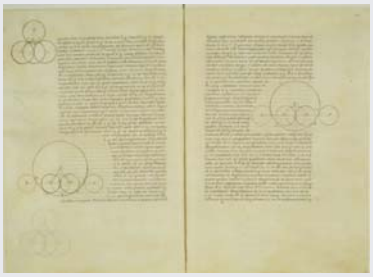
CONCLUSIONS (1)

- Main functional requirements are met by the final design (12 of the 19 requirements)
- Throughput and latency requirements are almost met; expected to be met in case of full speed G4 daughter cards and/or VSIPL functions redesign
- Review of graphical Ptolemy designs seems faster and more efficient than code reviews
  - Disadvantage is parameter handling and scope.
  - Design is highly multi-rate, but this is difficult to see
  - Some functionality is inside stars (hidden)
- Total design, validate & test time for bare beamformer was 354.5 hours, while normal development takes 481 hours: Approximately 36% faster (improvement ~1.36)

THALES Metra BAe Dynamics BAE SYSTEMS



Foundations

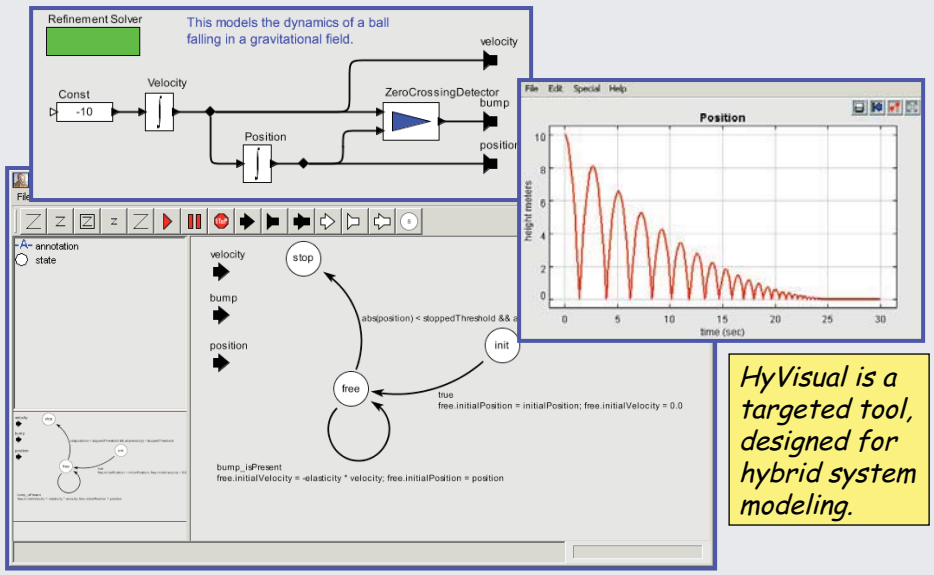


Our contributions:

- Behavioral Types
- Domain Polymorphism
- Responsible Frameworks
- Hybrid Systems Semantics
- Dataflow Semantics
- Tagged Signal Model
- Starcharts and Modal Model Semantics
- Discrete-Event Semantics
- Continuous-Time Semantics

Giving structure to the notion of "models of computation"

HyVisual – Hybrid System Modeling Tool Based on Ptolemy II, Released Jan. 2003



HyVisual is a targeted tool, designed for hybrid system modeling.

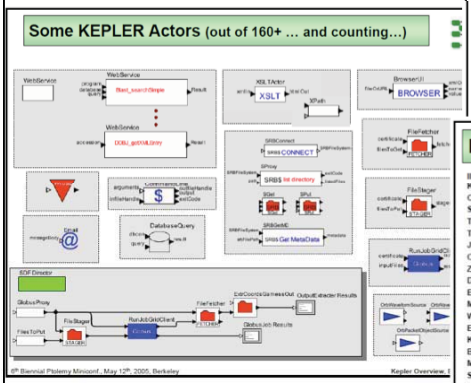
6th 2005

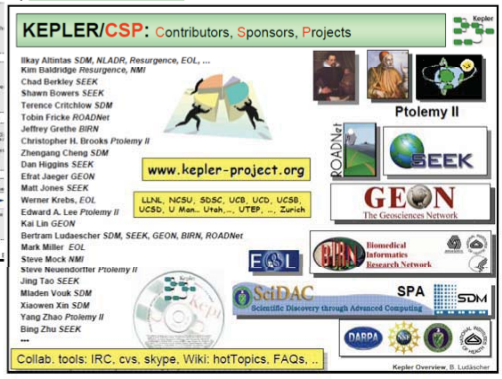
KEPLER: Overview and Project Status

Kepler

Bertram Ludäscher
ludaesch@ucdavis.edu

Associate Professor
Dept. of Computer Science & Genome Center
University of California, Davis





6th 2005

Growth of the Cal actor language

XILINX®

Programming with actors

Jörn W. Janneck
Xilinx Research Labs

driver application

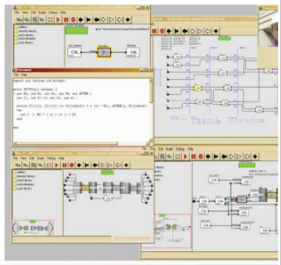
MPEG-4 decoder

metrics

- 60 atomic actors
- 22 atomic actor classes
- 3307 LOC (Cal)
- LOC per actor class between 7 and 2054

actor constructs

- variable token rates
- static/cyclostatic rates
- data-dependent choice
- test for absence of tokens
  - non-prefix-monotonic actors

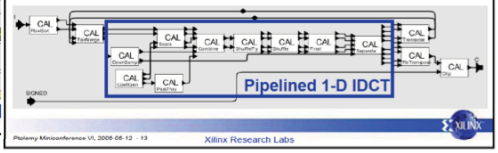


Ptolemy Miniconference VI, 2005-05-12 - 9 Xilinx Research Labs

code generation

2D-IDCT, version 2

- interleave row and column streams
- pipelined 1D-IDCT
- result:
  - 6 multipliers with 46% utilization
    - more operator re-use costly in terms of operand routing
  - >100 Mhz clock



Ptolemy Miniconference VI, 2005-05-12 - 13 Xilinx Research Labs

Ptolemy Project, Berkeley 23

7th 2007

Kepler

The Kepler Project

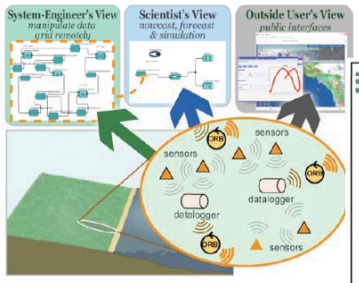
Overview, Status, and Future Directions

Matthew B. Jones
on behalf of the Kepler Project team

National Center for Ecological Analysis and Synthesis
University of California, Santa Barbara

NCEAS SEEK

REAP breakdown



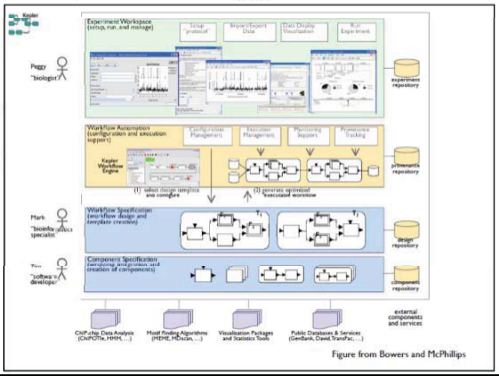


Figure from Bowers and McPhillips

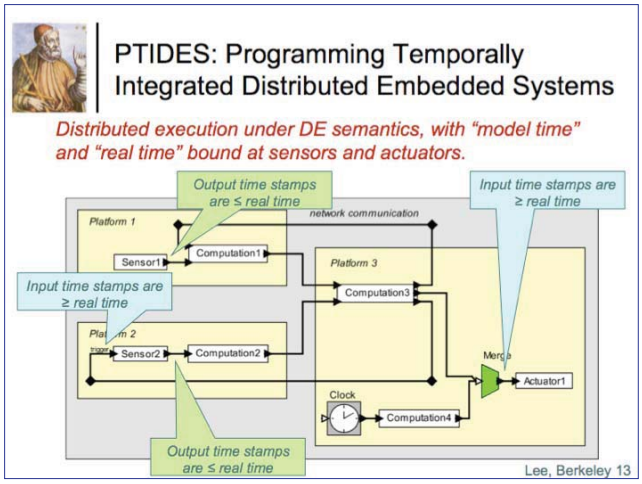
24

8th 2009

Hugo Andrade (NI)
Christopher Brooks
Dai Bui
Yasemin Demir
John Eidson (Agilent)
Thomas Feng
Shanna Shaye Forbes
Jeff Jensen
Edward Lee
Jackie Leung
Ben Lickly
Isaac Liu
Thomas Mandl (Bosch)
Slobodan Matic
Eleftherios Matsikoudis
Hiren Patel
Stephan Resmerita
Bert Rodiers
Yang Zhao
Jia Zou

8th 2009

Hugo Andrade (NI)
Christopher Brooks
Dai Bui
Yasemin Demir
John Eidson (Agilent)
Thomas Feng
Shanna Shaye Forbes
Jeff Jensen
Edward Lee
Jackie Leung
Ben Lickly
Isaac Liu
Thomas Mandl (Bosch)
Slobodan Matic
Eleftherios Matsikoudis
Hiren Patel
Stephan Resmerita
Bert Rodiers
Yang Zhao
Jia Zou

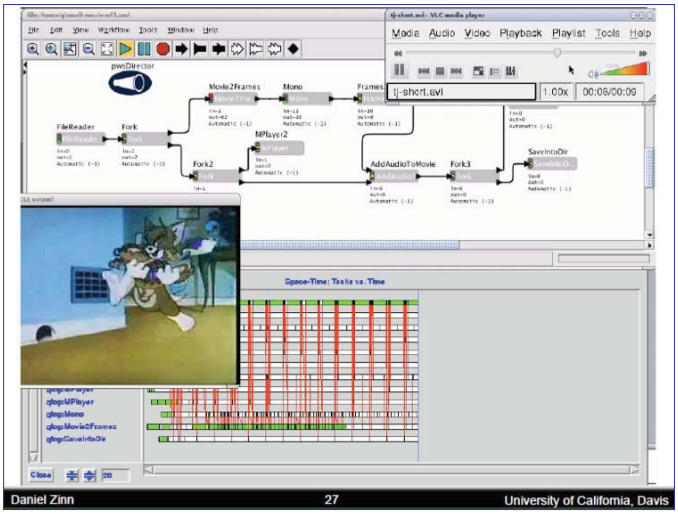


8th 2009

Parallel Virtual Machines in Kepler

Daniel Zinn
Xuan Li
Bertram Ludaescher

UC Davis



9th 2011

Let the show begin!

Distributed Execution Architectures in Kepler

Jianwu Wang[1], Daniel Crawl[1],
Ilkay Altintas[1], Chad Berkley[2], Matthew B. Jones[2]

[1] *San Diego Supercomputer Center, UCSD*
[2] *National Center for Ecological Analysis and Synthesis, UCSB*

Outline

- Distributed Execution Architectures in Kepler
- Master-Slave Distributed Execution Architecture in Kepler
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between the Above Two Architectures

Part I

- **Distributed Execution Architectures in Kepler**
- Master-Slave Distributed Execution Architecture in Kepler
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures

Distributed Execution Requirements in Kepler

- Various requirements on distributed execution in different environments, examples:
  - Ad-hoc network resources
  - Web service resources
  - Cluster resources
  - Grid resources
  - Cloud resources
  - …

Distributed Execution Supports in Kepler

- Kepler integrated frameworks and libraries to support the requirements
  - Remote method invocation (RMI) for ad-hoc network resources
  - Axis Web service libraries for Web Service invocation
  - Ssh session libraries (JSch) for remote execution and job submission on clusters
  - Globus libraries for Grid computing

Three Distributed Execution Levels in Kepler

- **Workflow level:** the whole workflow can be executed in distributed environments
  - Example: Web service for Kepler workflow execution
- **Actor level:** distributed computing and data resources can be utilized in an actor
  - Example: Web service actor in Kepler
- **Sub-workflow level:** sub-workflows can be executed in distributed environments
  - Example: Master-Slave and MapReduce Distributed Execution

Advantages of Using Workflow System for Distributed Execution

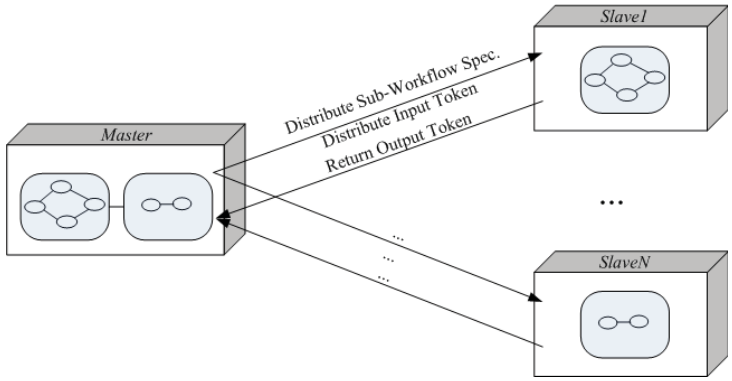
- **Reuse existing workflows**
  - Easily transform workflow from centralized execution to distributed execution
- **Transparent implementation**
  - Hide diverse distributed techniques from users, such as different job schedulers
  - Just drag-and-drop, no coding is needed
- **Optimal execution**
  - (Semi-)automatically get the best execution plan
- **Provenance support**
- **Fault tolerance**

Part II

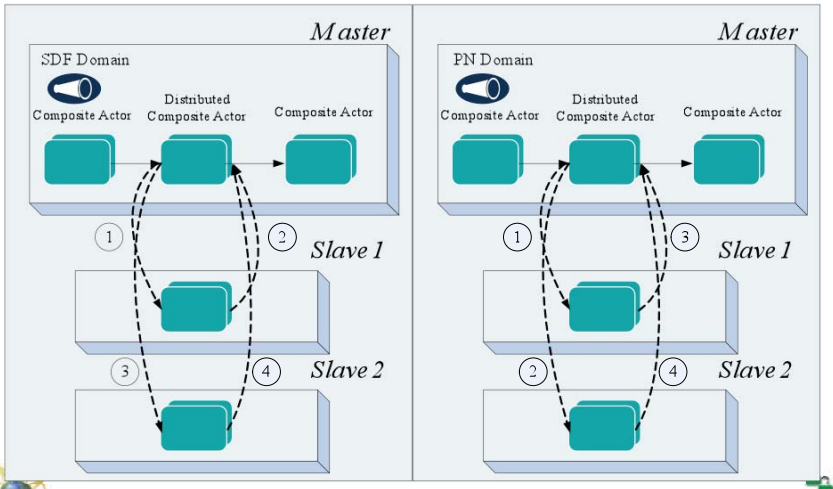
- Distributed Execution Architectures in Kepler
- **Master-Slave Distributed Execution Architecture in Kepler**
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures

Distributed Composite Actor

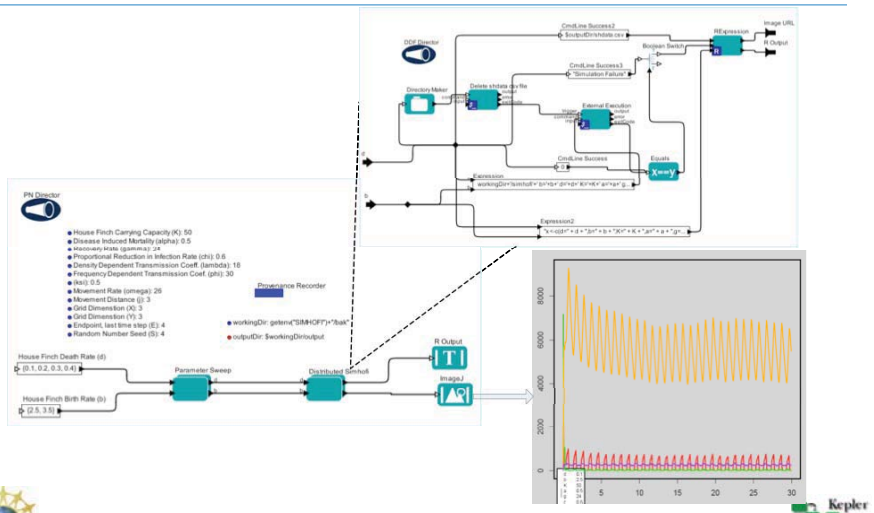
- As the role of Master, each token received by Distributed Composite Actor is distributed to a Slave node, executed, and the results returned.



Distributed Composite Actor Behaviors with Different Computation Models

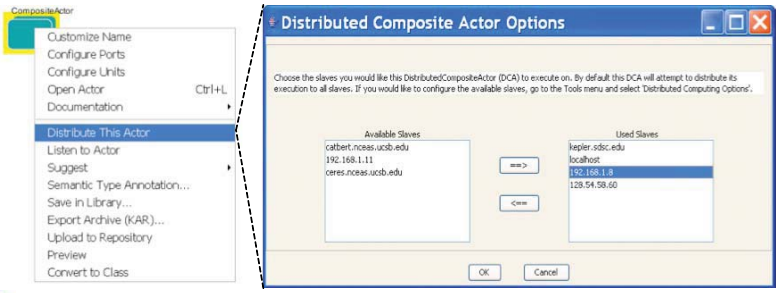


Demo Workflow

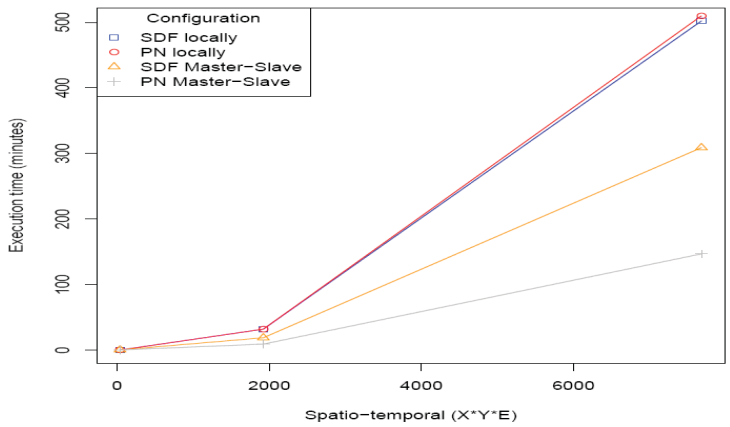


Usability

- Users use the DistributedCompositeActor just like the common composite actor
- Interaction for execution environment transition



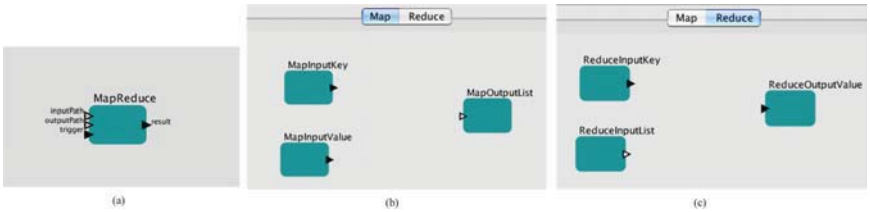
Performance Experiment



Part III

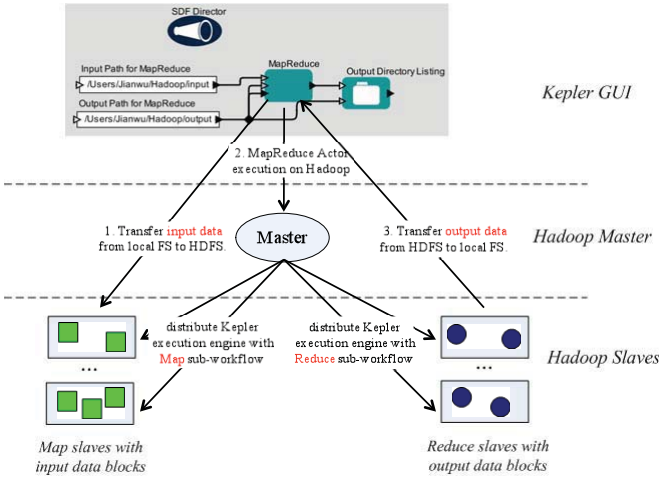
- Distributed Execution Architectures in Kepler
- Master-Slave Distributed Execution Architecture in Kepler
- **MapReduce Distributed Execution Architecture in Kepler**
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures

MapReduce Actor in Kepler

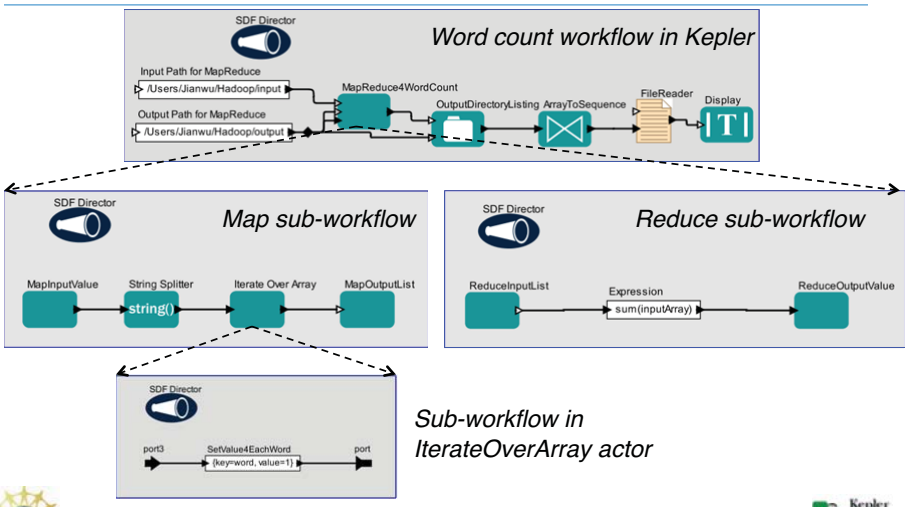


(a) MapReduce actor. (b) Map sub-workflow in MapReduce actor.
(c) Reduce sub-workflow in MapReduce actor.

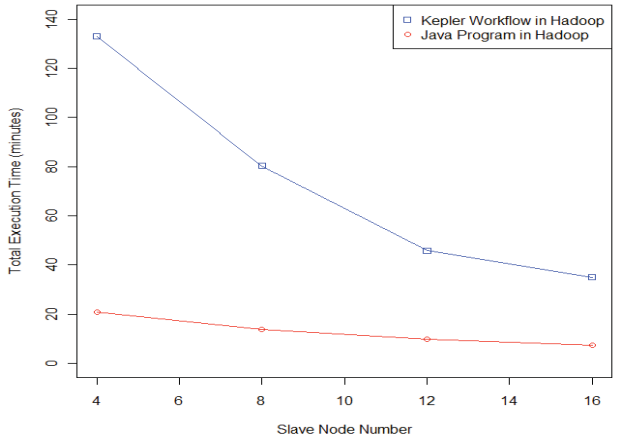
MapReduce Actor Execution in Hadoop



Using MapReduce Actor for Word Count



Performance Experiment for Word Count



Part IV

- Distributed Execution Architectures in Kepler
- Master-Slave Distributed Execution Architecture in Kepler
- MapReduce Distributed Execution Architecture in Kepler
- **Comparison between Master-Slave and MapReduce Distributed Execution Architectures**

Commonalities

- **Both have distributed data + distributed programs**
- **Both have master and slaves**
- **Both have execution engines on slaves**

Main Differences

- **MapReduce**
  - Usually, all input data needs to be staged in beforehand and outputs is only accessible when the whole execution is finished
  - More suitable for large data sets, and has good scalability on clusters with numerous nodes
- **Master-Slave**
  - Inputs can be provided dynamically and get its result gradually once it is generated
  - More suitable for dynamic data distribution cases

Thanks!

- **Papers for the Above Work**
  - J. Wang, D. Crawl, I. Altintas. *Kepler + Hadoop – A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems*. In Proc. of the 4th Workshop on Workflows in Support of Large-Scale Science (WORKS09) at Supercomputing 2009 (SC2009) Conf..
  - J. Wang, I. Altintas, P. R. Hosseini, D. Barseghian, D. Crawl, C. Berkley, M. B. Jones. *Accelerating Parameter Sweep Workflows by Utilizing Ad-hoc Network Computing Resources: an Ecological Example*. In Proceedings of IEEE 2009 Third International Workshop on Scientific Workflows (SWF 2009), 2009 Congress on Services (Services 2009).
- **More Information:**
  - Distributed Execution Interest Group of Kepler: https://dev.kepler-project.org/developers/interest-groups/distributed
  - Contact: jianwu@sdsc.edu

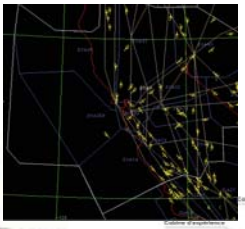
MODELING DISTRIBUTED REAL-TIME SYSTEMS WITH PTOLEMY II

Patricia Derler, Jia Zou, Slobodan Matic, John Eidson, Edward A. Lee

University of California, Berkeley

DISTRIBUTED REAL-TIME SYSTEMS

Multiple computers, comprising of sensors and actuators, connected on a network that act and react on events to meet timing constraints.

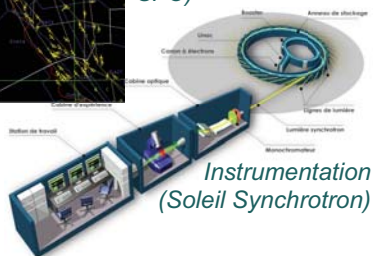


Transportation (Air traffic control at SFO)

Automotive

E-Corner, Siemens

Telecommunications



Instrumentation (Soleil Synchrotron)

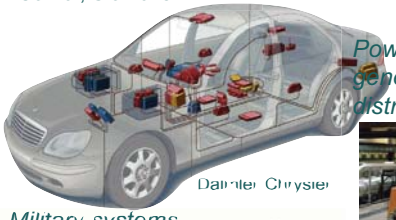
Building Systems

Factory automation

Courtesy of Kuka Robotics Corp.

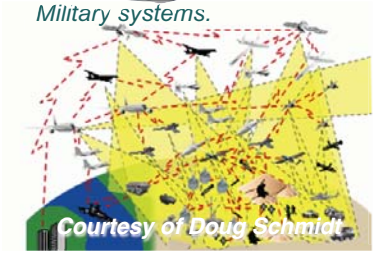
Power generation and distribution

Courtesy of General Electric



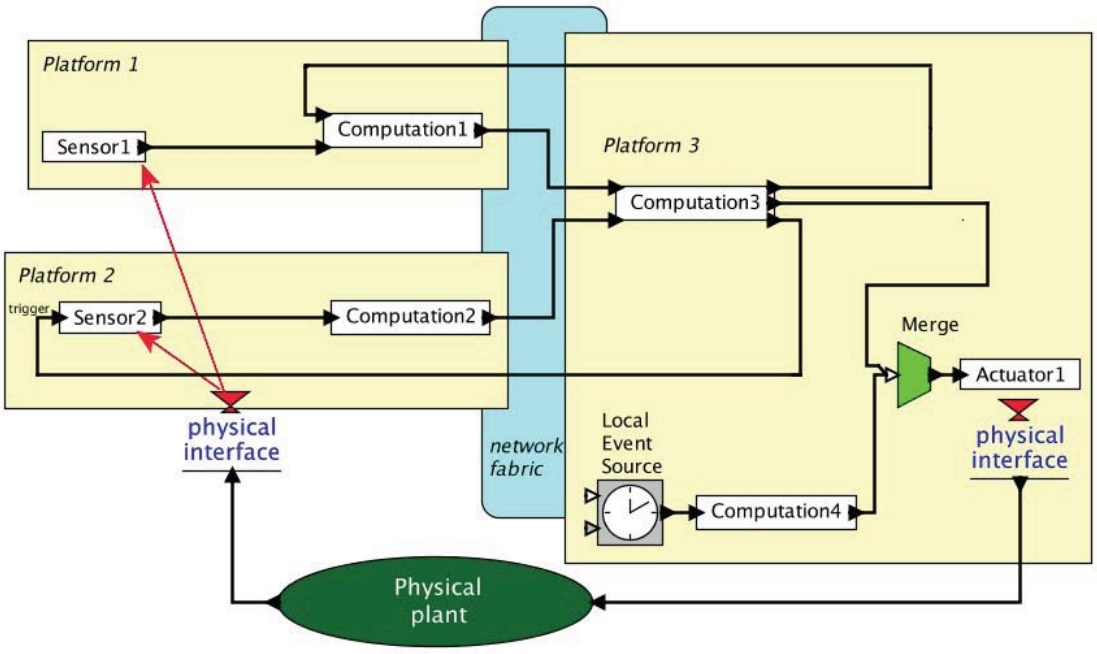
Daimler Chrysler

Military systems.



Courtesy of Doug Schmidt

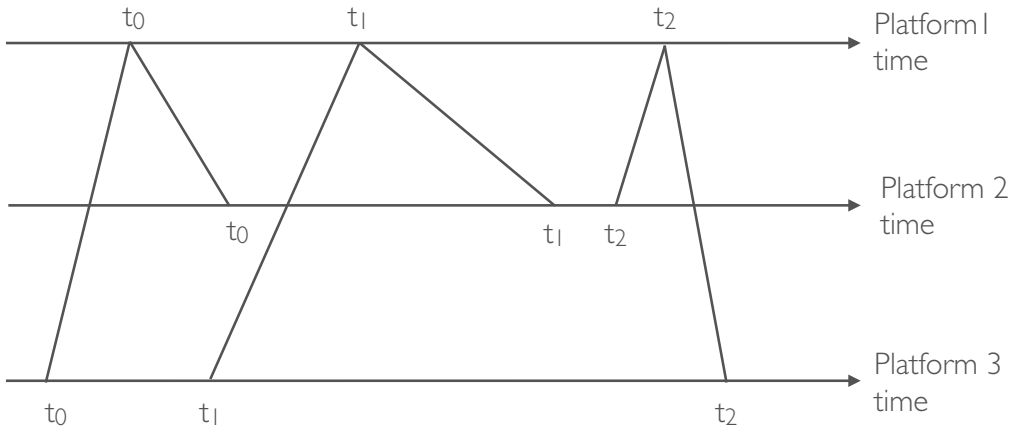
MODELING DISTRIBUTED REAL-TIME SYSTEMS



OVERVIEW

- Challenges: How to model
  - Time
  - Network
  - Execution time
  - Execution semantics
- Address modeling challenges in PTIDES

THE TIME CHALLENGE



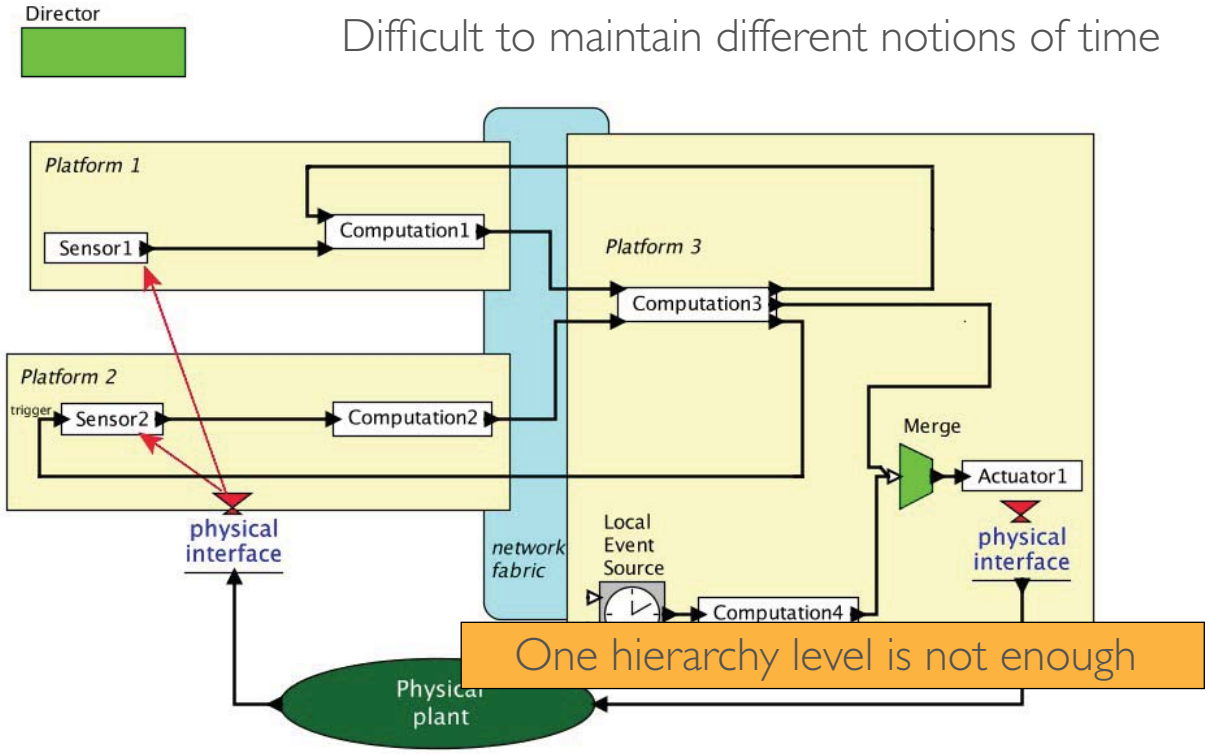
Distributed platforms have different notions of time

Platform clocks drift

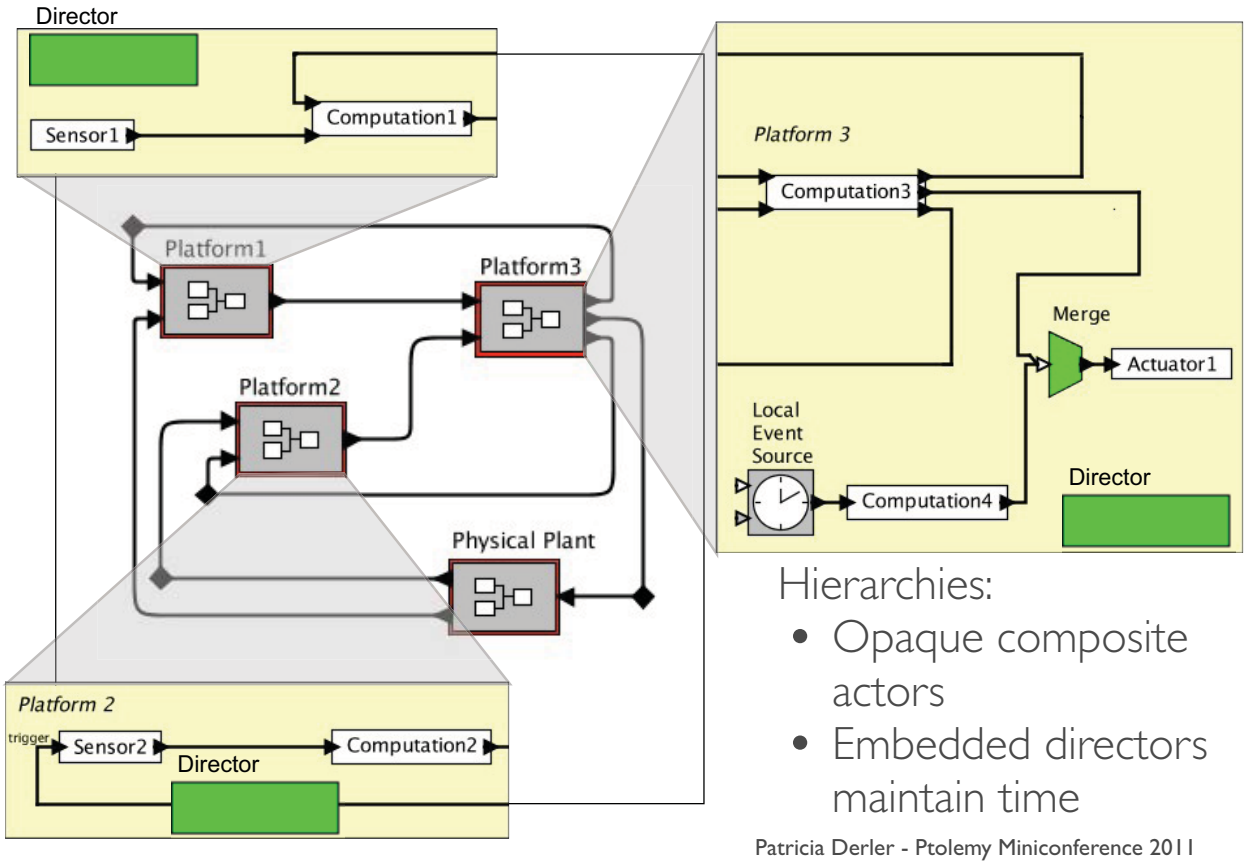
Platform clocks drift at varying rates

MODELING DISTRIBUTED SYSTEMS

Director mediates between actors
Difficult to maintain different notions of time



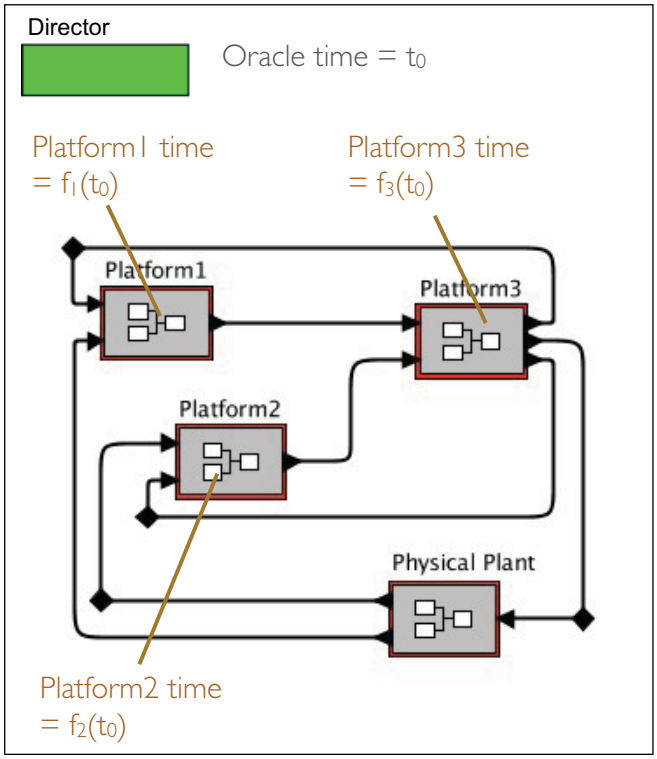
MODELING DISTRIBUTED SYSTEMS



Hierarchies:

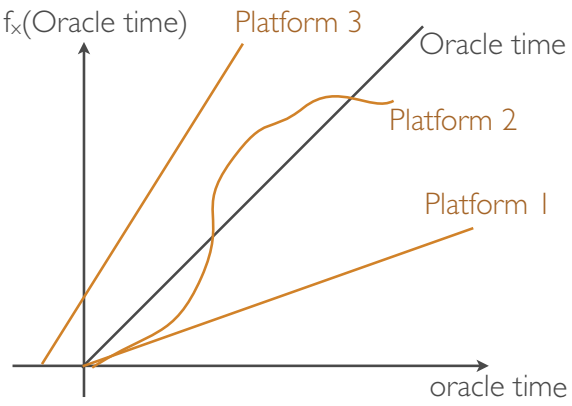
- Opaque composite actors
- Embedded directors maintain time

MODELING DISTRIBUTED SYSTEMS

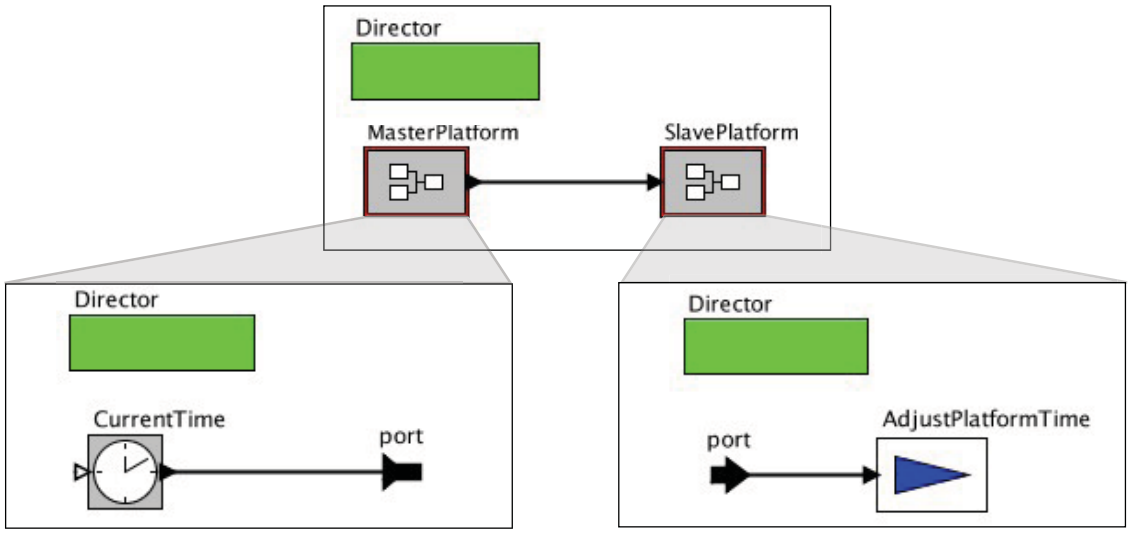


Top level: Oracle time

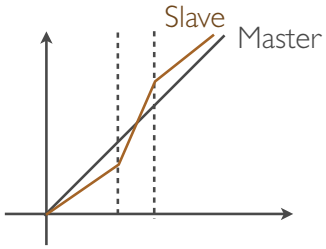
Every platform time is defined with respect to oracle time



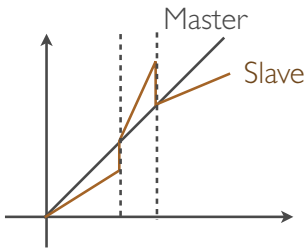
CLOCK SYNCHRONIZATION



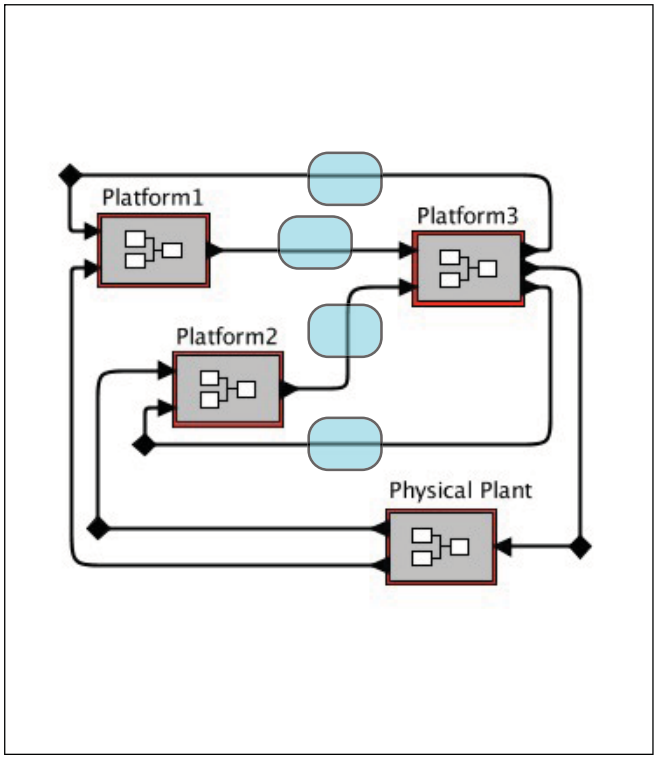
adjust clock rate:



adjust clock value:



MODELING NETWORKS

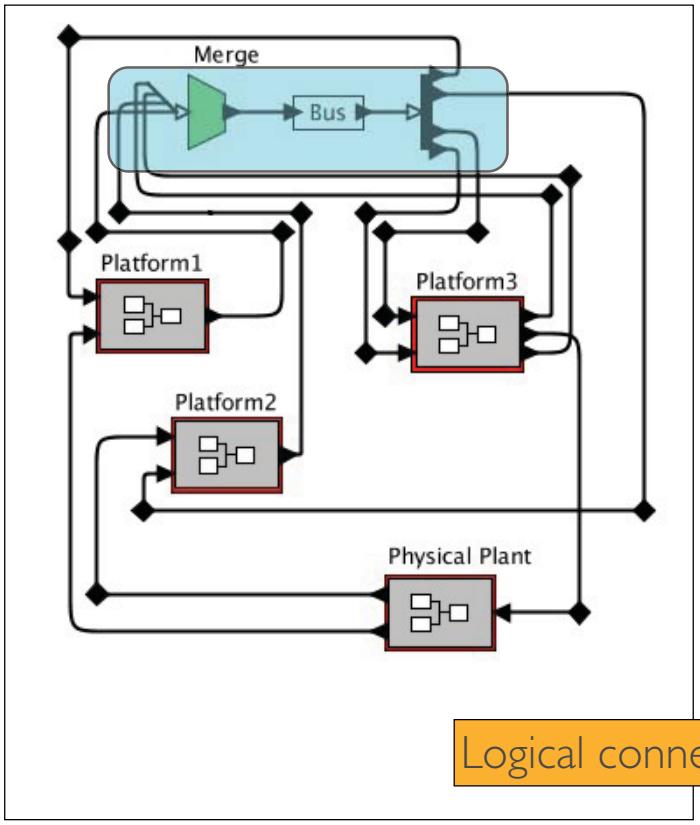


Distributed platforms communicate via networks

Networks have latencies

e.g. CAN Bus, TTEthernet

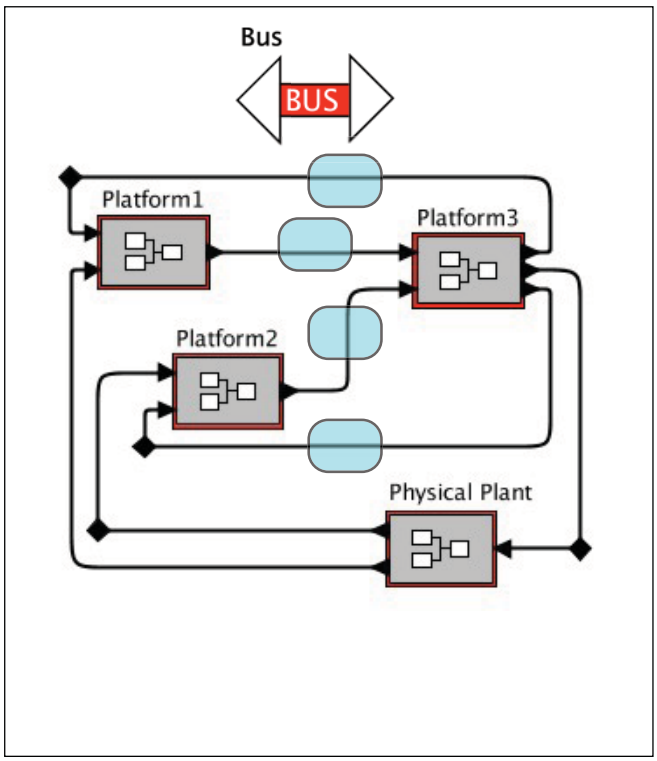
MODELING NETWORKS



Physical connections vs. Logical connections

Logical connections are lost

MODELING NETWORKS

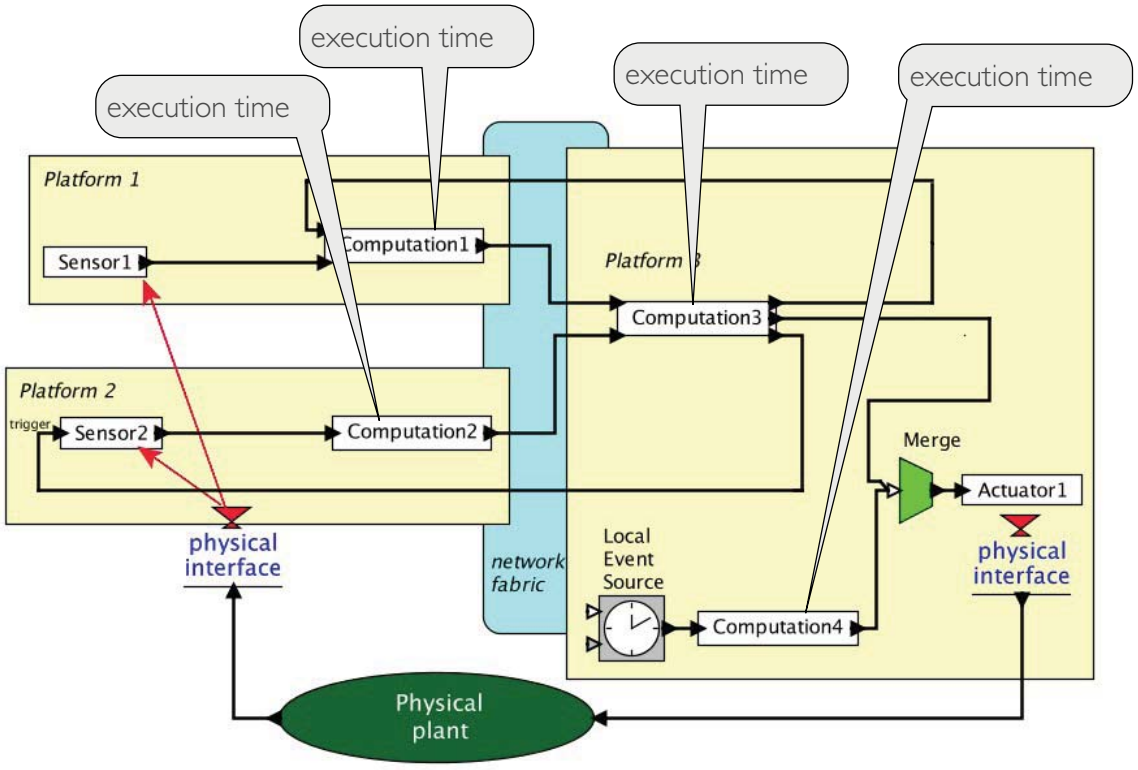


Aspect-oriented modeling

Quantity managers [Balarin03] and schedulers to simulate network latency

[Balarin03] F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. L. Sangiovanni-Vincentelli, and Y. Watanabe. Metropolis: an integrated electronic system design environment. Computer, 36(4), 2003.

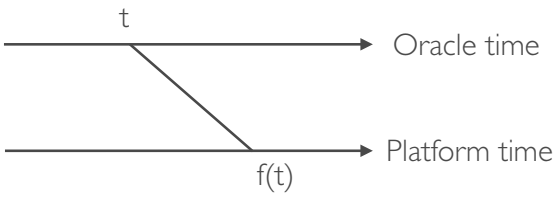
MODELING EXECUTION TIME

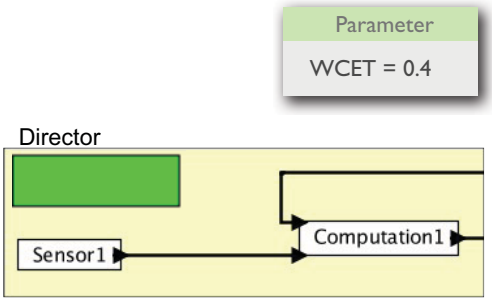


MODELING EXECUTION TIME

How is execution time computed?

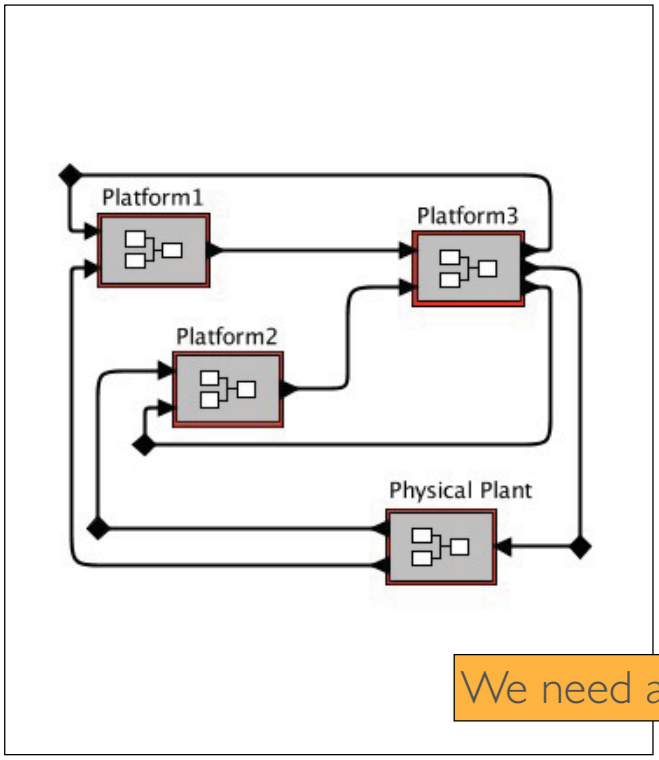
Which time line to use for specifying execution time?





Aspect-oriented programming

DISTRIBUTED DISCRETE-EVENT MODELS

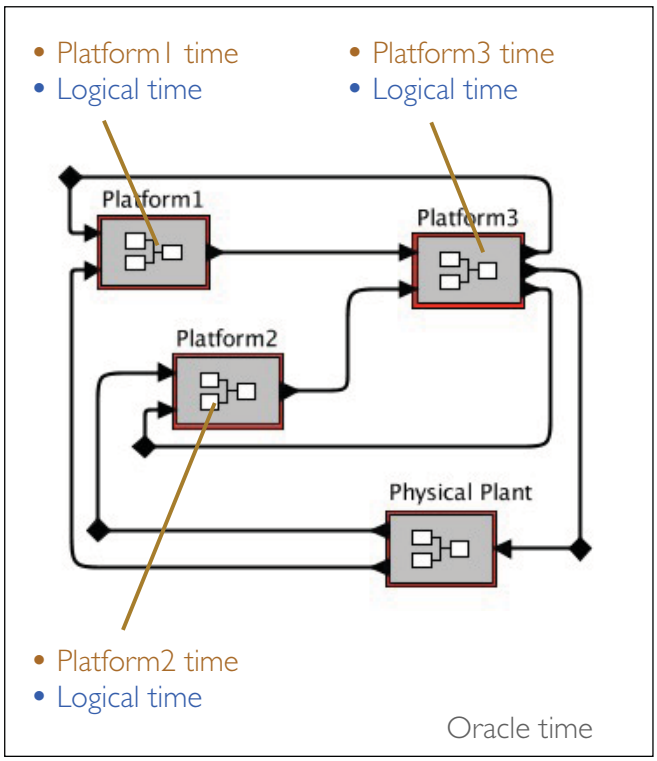


Discrete-Event (DE) for simulation

DE as a application specification language which serves as a semantic basis for obtaining determinism in distributed real-time systems.

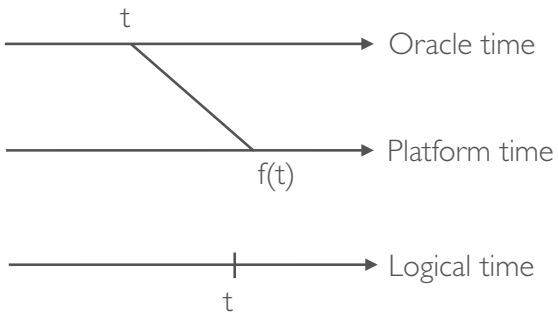
We need another time line

DISTRIBUTED DISCRETE-EVENT MODELS



Logical time describe the execution semantics

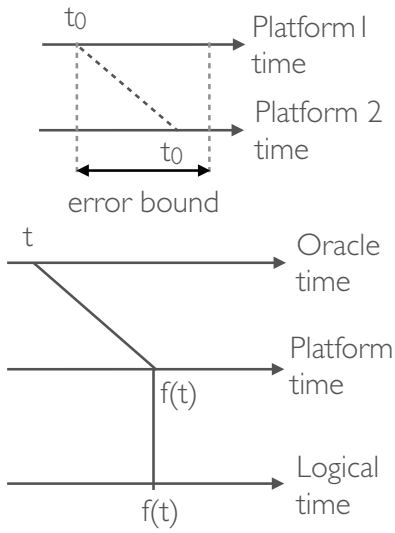
New time line: logical time



PTIDES: AN APPLICATION

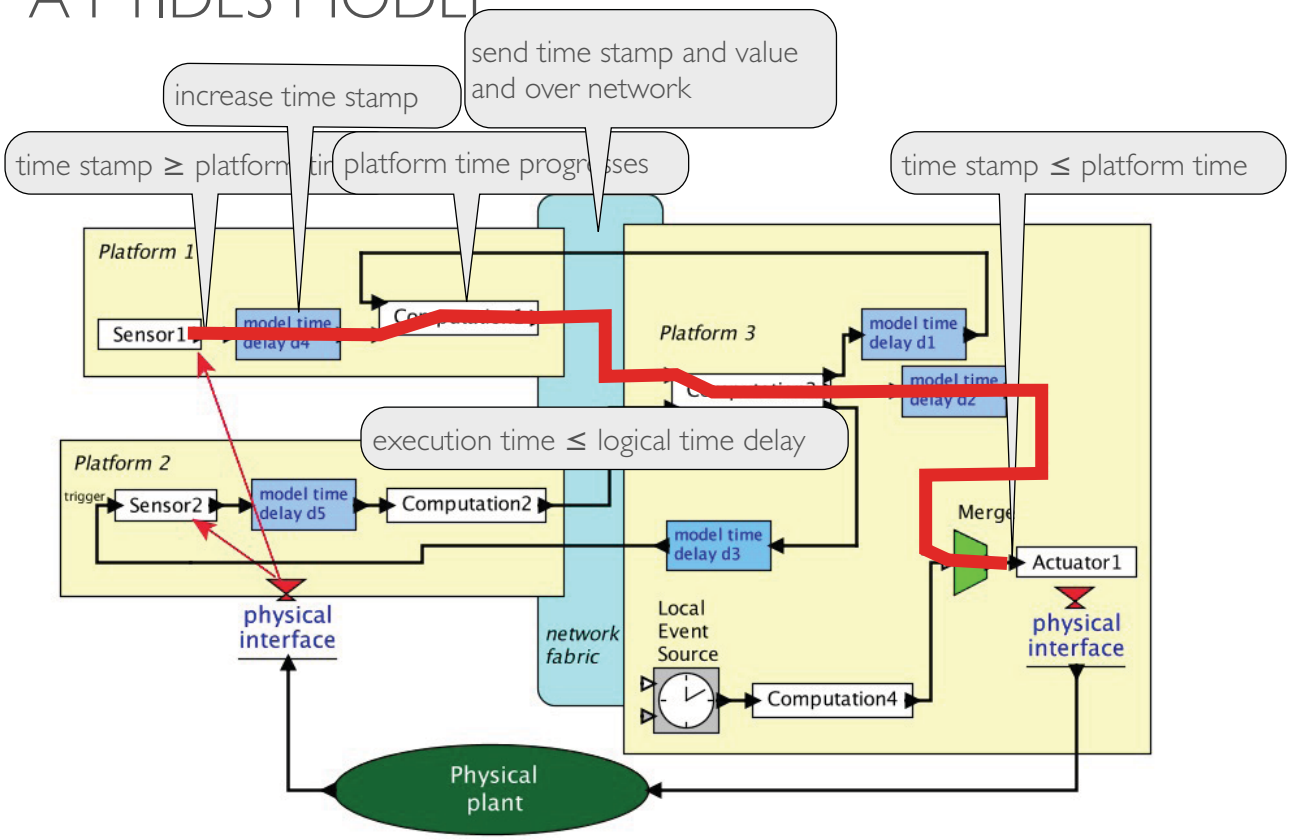
Programming temporally integrated distributed event systems [Zhao07]

- Discrete event model for execution
- Relates logical time to platform time whenever necessary
- Requires bounded error between platform clocks: Relies on clock synchronization
- Events are processed in time-stamped order



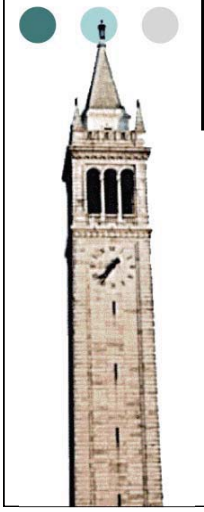
[Zhao07] Y. Zhao, J. Liu, and E. A. Lee. A programming model for time-synchronized distributed real-time systems. In Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 07), pages 259–268, Bellevue, WA, USA, Apr 2007.

A PTIDES MODEL



SUMMARY

- Distributed embedded systems
- Each distributed platform has its own notion of time
- Modeling distributed systems with different notions of time and clock drifts
- Clock synchronization
- Modeling networks
- Modeling distributed discrete event systems
- PTIDES



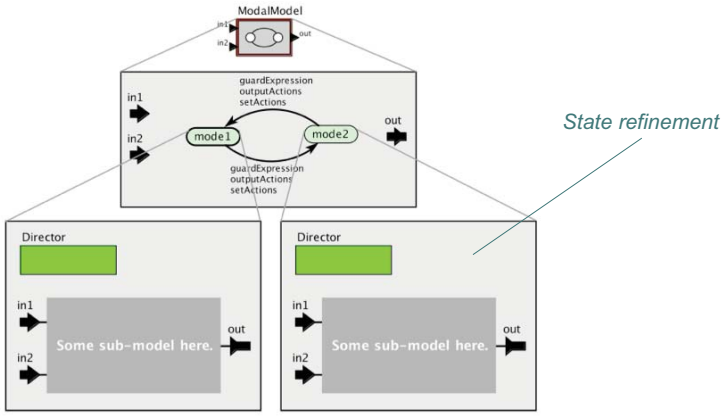
Semantics of Modal Models in Ptolemy II

Edward A. Lee
Stavros Tripakis
UC Berkeley

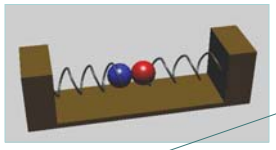
Ninth Biennial Ptolemy Miniconference
February 16, 2011

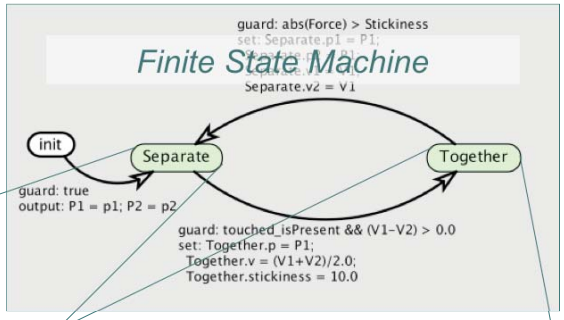
What are modal models?

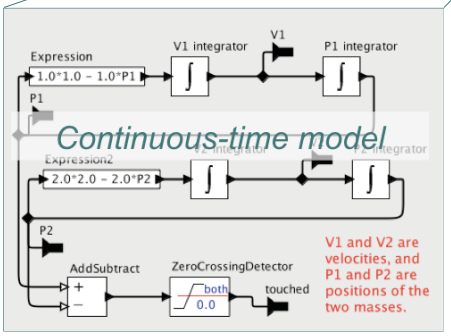
- Modal models = hierarchical models mixing FSMs (Finite State Machines) and other models

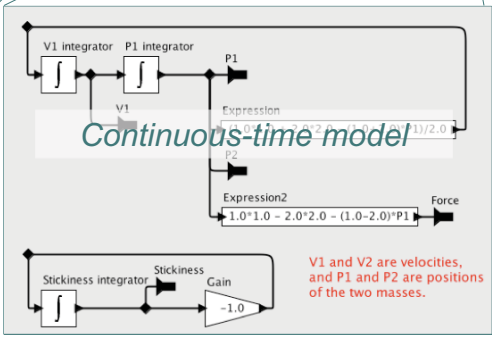


Example: Hybrid System







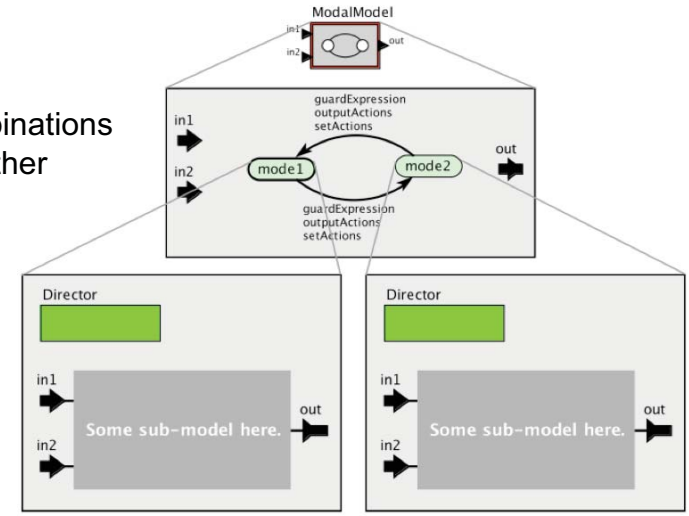


Influences for this work

- Statecharts [Harel 87]
- Argos [Maraninchi 91]
- Esterel [Berry & Gonthier 92]
- Abstract state machines [Gurevich 93]
- Hybrid systems [Puri & Varaiya 94, Henzinger 99]
- Timed automata [Alur & Dill 94]
- SyncCharts [Andre 96]
- I/O Automata [Lynch 96]
- *Charts [Girault, Lee, & Lee 99]
- UML State machines

Ptolemy II goes one step further

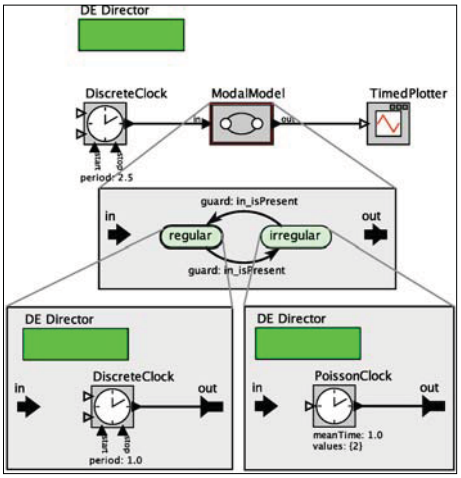
Arbitrary combinations of FSMs with other domains



How to give meaning to modal models?

- Not always trivial:

What happens to the events produced by the discrete clock while system is at mode "irregular"?



How to give meaning to modal models?

- Approach 1:
  - Give a meaning to every possible combination of models:
    - Hierarchical state machines (Statecharts, UML, …)
    - Timed automata (timed models within state machines)
    - Hybrid automata (continuous models within state machines)
    - Mode automata (synchronous/reactive within state machines)
    - …
  - Scalable?

How to give meaning to modal models?

- Approach 2 [Ptolemy]:
  - Modular semantics
    - semantics of composite blocks = function of semantics of sub-blocks
  - Compositionality
  - Heterogeneity

This talk

- A formal semantics for Ptolemy
  - operational semantics
    - close enough to the Java implementation to be faithful
    - but not too close (fits in a few pages)
- A formal semantics for modal models

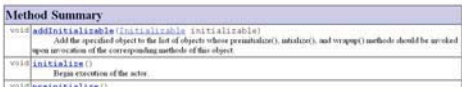
A FORMAL SEMANTICS FOR PTOLEMY

Abstract semantics

- Actor = State Machine
- Actor = Inputs + Outputs + States + Initial state
  + Fire + Postfire
- Fire = output function: produces outputs given current inputs + state $F : S \times I \rightarrow O$
- Postfire = transition function: updates state given current inputs + state $P : S \times I \rightarrow S$

Implemented as Java interfaces

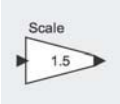
Interface "Initializable"



Interface "Executable"

Method Summary	
void	**fire**() Fire the actor.
boolean	**isFireFunctional**() Return true if this executable does not change state in either the prefire() or the fire() method.
boolean	**isStrict**() Return true if this executable is strict, meaning all inputs must be known before iteration.
int	**iterate**(int count) Invoke a specified number of iterations of the actor.
boolean	**postfire**() This method should be invoked once per iteration, after the last invocation of fire() in that iteration.
boolean	**prefire**() This method should be invoked prior to each invocation of fire().
void	**stop**() Request that execution of this Executable stop as soon as possible.
void	**stopFire**() Request that execution of the current iteration complete.
void	**terminate**() Terminate any currently executing model with extreme prejudice.

Examples



Single state ("stateless").
P : trivial (state never changes).
*F : out := in*1.5*

State: the current value
F : out := state
P: state := input

Behaviors – untimed

$$F : S \times I \rightarrow O$$
$$P : S \times I \rightarrow S$$

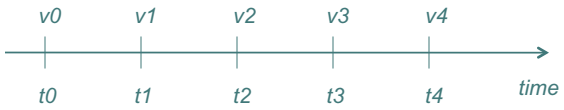
- Set of untimed traces:

$$s_0 \xrightarrow{x_0, y_0} s_1 \xrightarrow{x_1, y_1} s_2 \xrightarrow{x_2, y_2} \cdots$$

- such that for all i :

$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i, x_i)$$

What about “timed” actors?

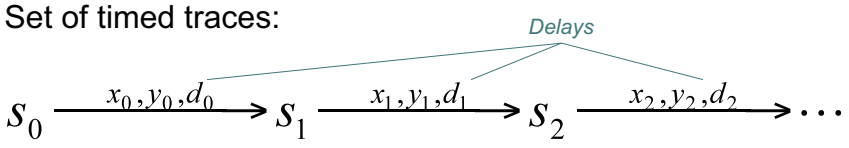


- States include special ***timer*** variables:
  - Set to some positive value, “expire” when they reach 0, can be “frozen” and “resumed”, …

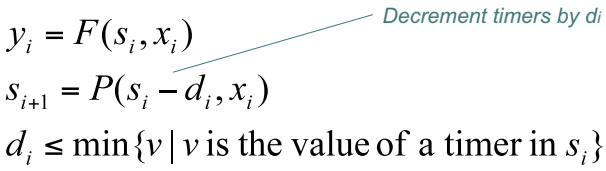
Behaviors – timed

$$F : S \times I \rightarrow O$$
$$P : S \times I \rightarrow S$$

- Set of timed traces:

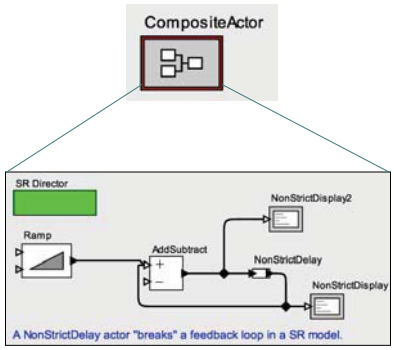


- such that for all i :



What about hierarchy?

How to give semantics to a hierarchical model?
i.e.,
How to give semantics to a composite actor?



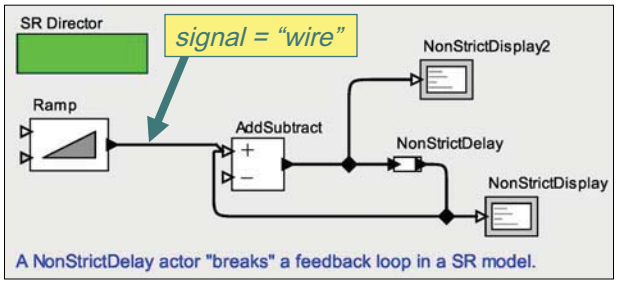
Directors = composition operators

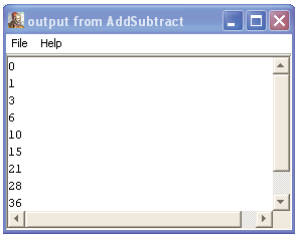
- Given a composite actor with a set of subactors
  A1, A2, …, with fire & postfire functions
  F1/P1, F2/P2, …
- … its director defines a new pair of fire & postfire functions *F* and *P.*
- *F* and *P* define a new, composite actor *A.*
- *A* can be used like an atomic actor (black-box).

Modular, compositional semantics

[Synchronous languages: Esterel, Lustre, Signal, …]

Example: Synchronous/Reactive (SR)



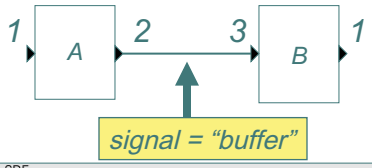


- *To compute the composite F, director solves a **fixpoint**:*
  - *Keep on evaluating Fi's until the values of all signals stabilize (this includes output signals in particular)*
  - *State remains unchanged during computation of the fixpoint!*

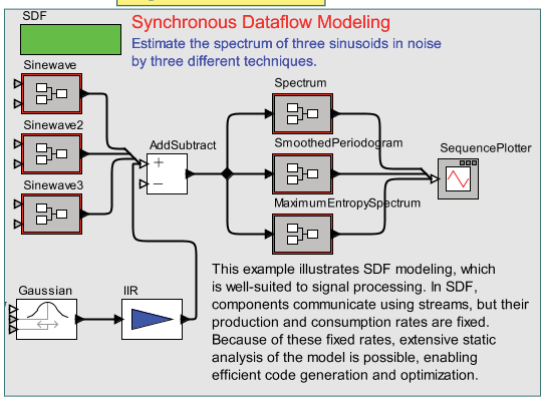
    c.f. separation of fire and postfire
- *To compute the composite P, just execute all Pi's*

[Lee-Messerschmitt '87]

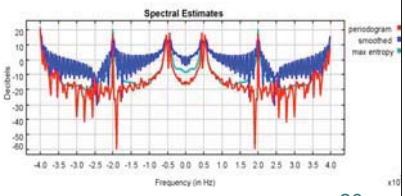
Example: Synchronous Data Flow (SDF)



In each firing, actors consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.



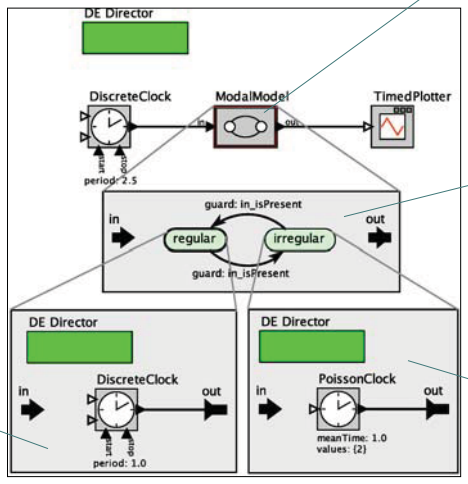
- *SDF Director computes periodic schedule, e.g., A,A,A,B,B*
- *Composite fire() fires all internal actors according to schedule*



MODAL MODEL SEMANTICS

Giving semantics to modal models

Goal:
define F, P functions
for the modal model

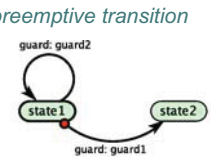


F_c, P_c functions
already defined
for the "controller"
automaton

F_1, P_1 functions
already defined
for this refinement

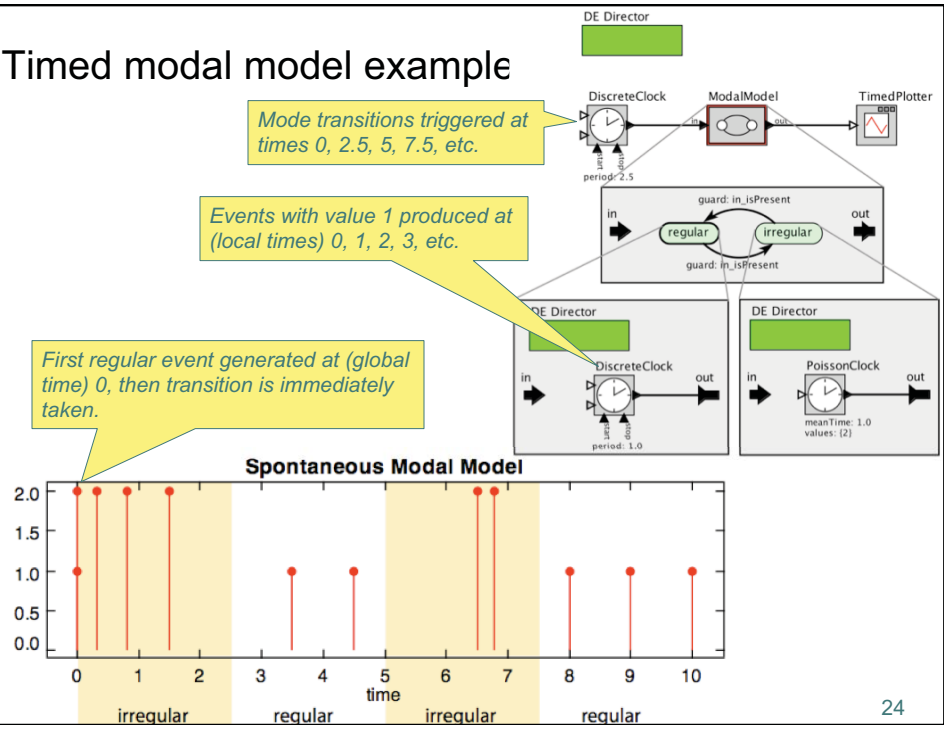
F_2, P_2 functions
already defined
for this refinement

Rough description of semantics



- Given current controller state *si* :
- If no outgoing transitions from *si* are enabled:
  - Use *Fi* and *Pi* to compute *F* and *P*
- If preemptive outgoing transitions from *si* are enabled:
  - Use the actions of these transitions to compute *F* and *P*
- If only non-preemptive outgoing transitions from *si* are enabled:
  - First fire refinement, then transition, i.e.:
    - *F* is the composition of *Fi* and the output action of a transition
    - *P* is the composition of *Pi* and the state update action of a transition
- Timers of refinements suspended and resumed when exiting/entering states
- Details in paper “Modal Models in Ptolemy” [EOOLT 2010]

Timed modal model example



Conclusions, ongoing work and future challenges

- Modular formal semantics for Ptolemy II
  - Directors = composition operators over state machines
- Semantics worked out for modal models [EOOLT 2010]
  - Currently extending it to other domains: Synchronous-Reactive, SDF, Discrete-Event, Continuous-Time, …
- Meta-model to describe semantics?

Thank you

- Questions?

Static Analysis using the Ptolemy II Ontologies Package

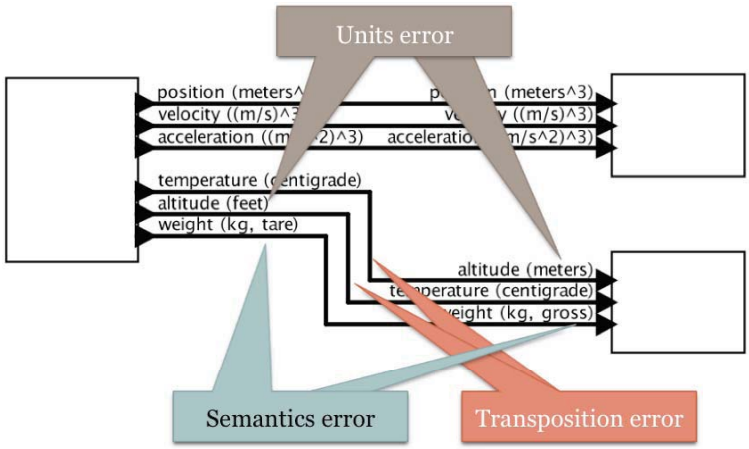
Charles Shelton, Elizabeth Latronico (Bosch Research and Technology Center)
Ben Lickly, Edward Lee (UC Berkeley)

Ptolemy Mini-Conference, February 16, 2011

Motivation

- Cars are networked software systems
  - Up to 70 Electronic Control Units
  - Software crucial for many features
    - Electronic stability control
    - Parking assist
    - Emissions control
    - Engine Start/Stop
    - Active and passive safety
  - ***Bosch makes all of these systems for auto manufacturers***
- How can we manage increasing complexity and interconnectedness of software models?
- Analysis approaches promising, but hand-annotation has drawbacks
  - Time intensive to develop and maintain
  - People are inconsistent, make errors
  - Repeat for every composition

Examples of Model Construction Errors

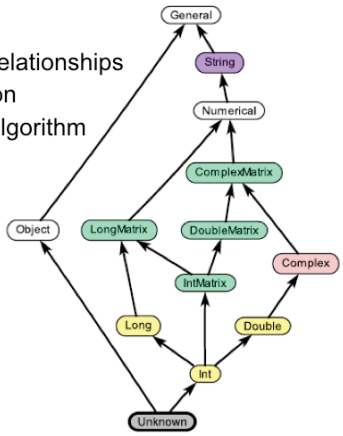


Static Analysis Using Ontologies

- An ***Ontology*** consists of:
  - A set of ***Concepts***
  - ***Relationships*** between those concepts
- Ontologies are used for representation of semantic information
  - General ontology frameworks (eg. OWL) focus on expressiveness
  - Arbitrary ontologies represent complex relationships as a graph
- Restrict Ptolemy ontologies to ***lattice*** graph structure
  - Lattice elements form a complete partial order
  - Existing scalable analysis algorithms
  - Existing work from compiler static analysis

Ontology Example: Ptolemy Type System

- Ptolemy type system implementation
  - Types organized in a lattice
  - Edges represent "can be converted to" relationships
  - Automatic type inference and propagation
    - Rehof-Mogensen constraint solving algorithm
- Users define type constraints in their actors
  - eg. An actor's output port type must be ***"greater than or equal to"*** *(higher in the lattice)* the input port type
  - Connections between actors imply that the sink type **≥** the source type

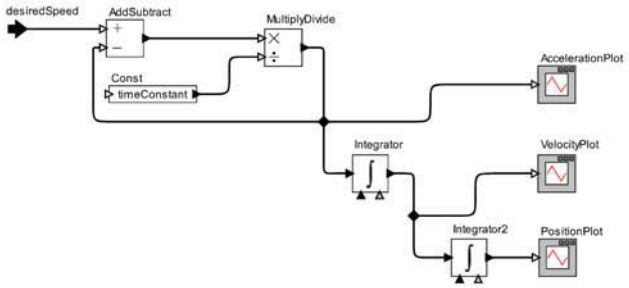


Ptolemy Ontologies Framework

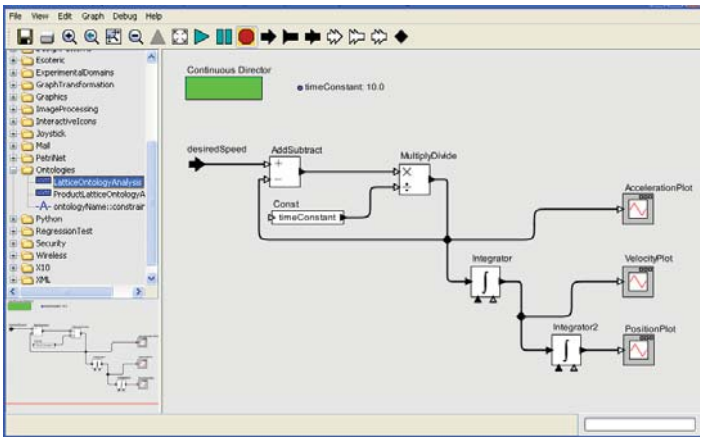
- Ontologies package generalizes the Ptolemy type system framework
  - Users can define their own ontology
  - Must also define the rules that determine ontology concept resolution
    - Constraints between model elements
    - Constraints between actor input and output ports
  - Reuses existing Ptolemy code
- Constraints are specified as inequalities between concepts assigned to each model element
  - $c_{output} \geq c_{input}$
  - $c_{output} \geq f(c_{input})$ where f is a monotonic function in the ontology domain ($c_a \geq c_b$ ***implies*** $f(c_a) \geq f(c_b)$)

Demo: Dimensional Analysis

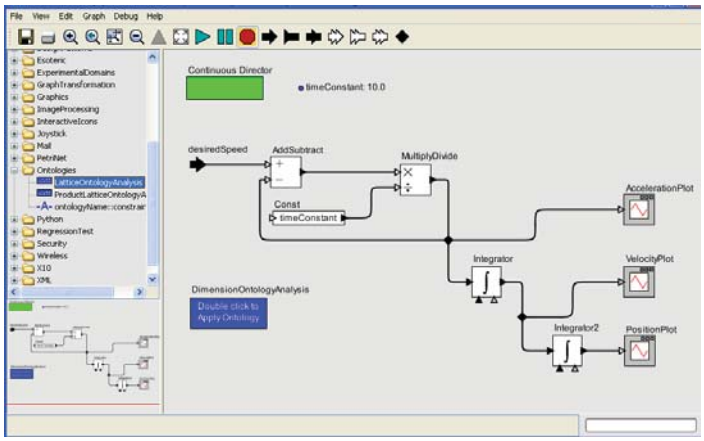
- Use the Ontology static analysis to infer dimensional properties
  - Position, Velocity, Acceleration, Time
- Ptolemy Model Example: Simple Car Dynamics Model
  - There is an error in this model that leads to incorrect results



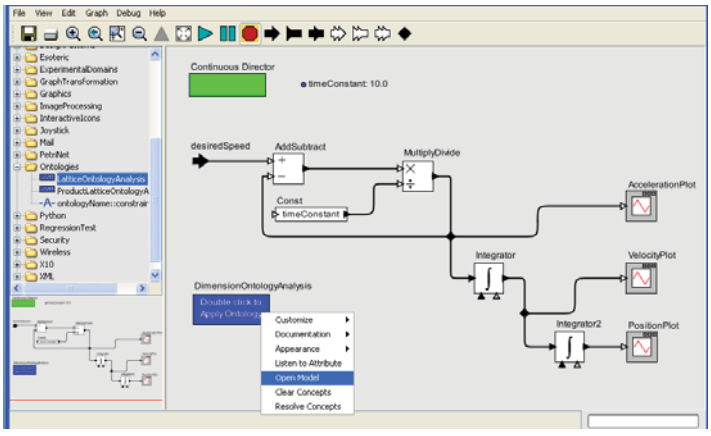
Step 1: Drag in a Lattice Ontology Solver



Step 1: Drag in a Lattice Ontology Solver

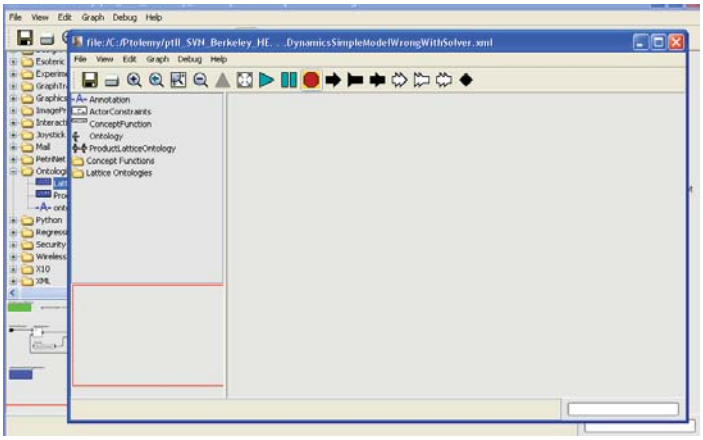


Step 2: Open the Solver Model



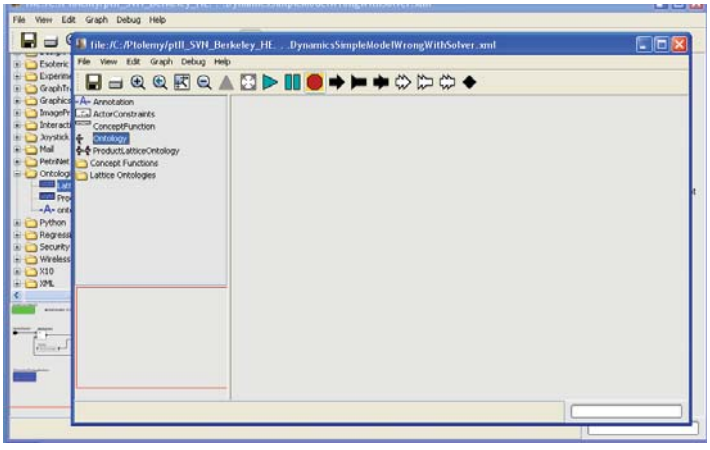
Static Analysis Using Ptolemy II Ontologies

Step 2: Open the Solver Model

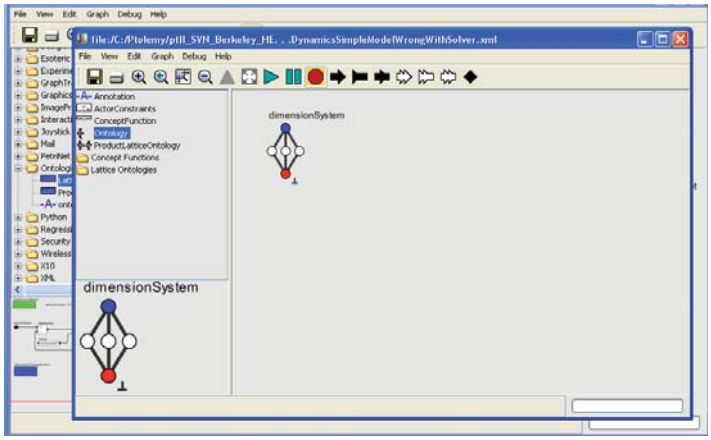


Static Analysis Using Ptolemy II Ontologies

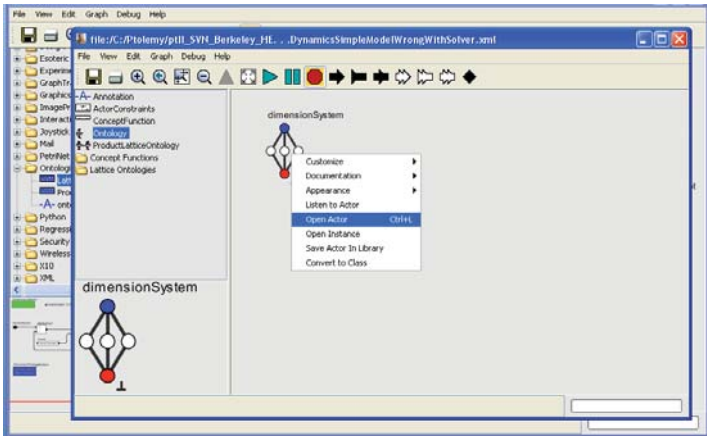
Step 3: Drag an Ontology into the Solver Model



Step 3: Drag an Ontology into the Solver Model

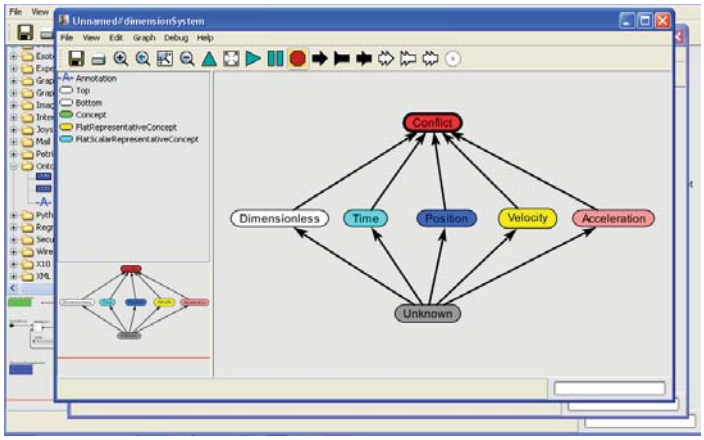


Step 4: Create the Dimensional Analysis Ontology



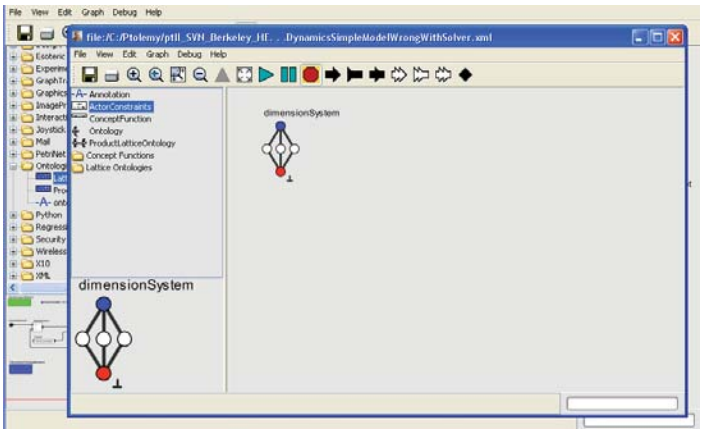
Static Analysis Using Ptolemy II Ontologies

Step 4: Create the Dimensional Analysis Ontology

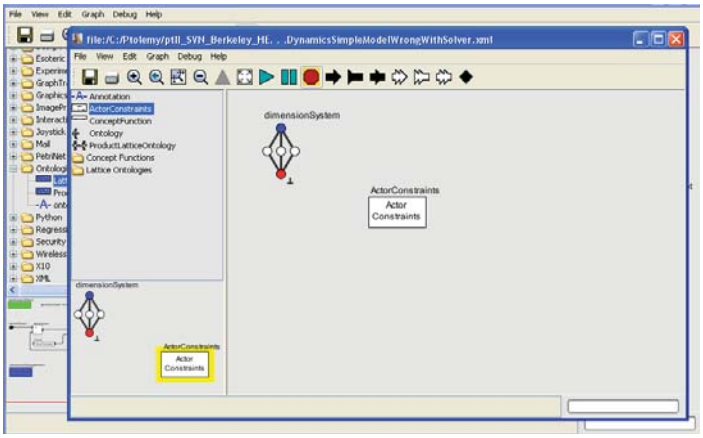


Static Analysis Using Ptolemy II Ontologies

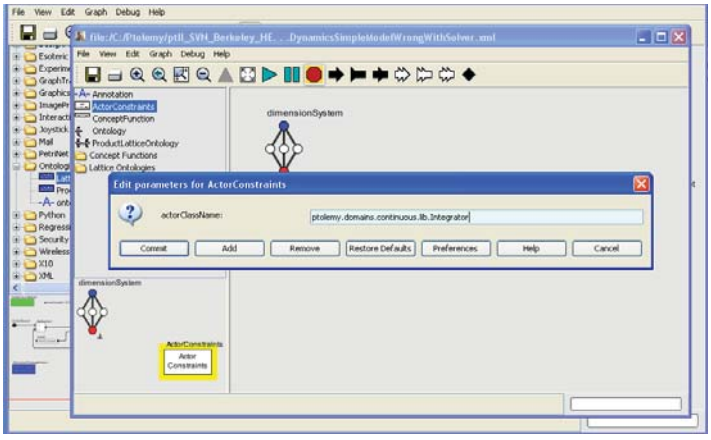
Step 5: Add Actor Constraints to the Solver Model



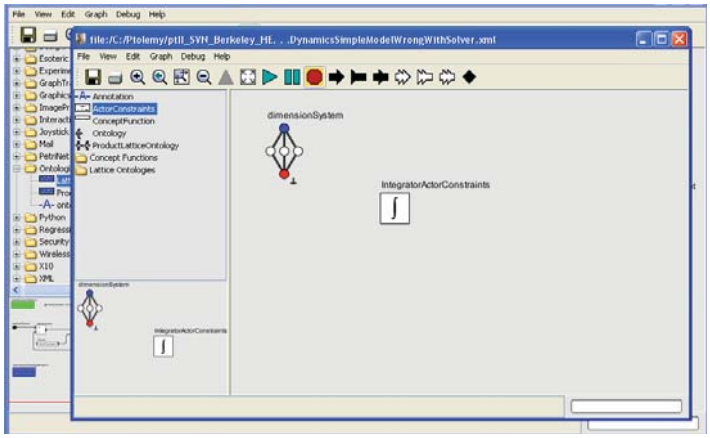
Step 5: Add Actor Constraints to the Solver Model



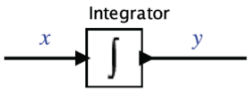
Step 5: Add Actor Constraints to the Solver Model

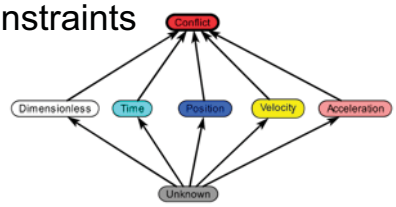


Step 5: Add Actor Constraints to the Solver Model



Defining Actor-Specific Constraints





Actor	Elements	Constraints
Integrator	input port ***derivative*** (x), output port ***state*** (y)	$c_x \geq f_I(c_y)$ $c_y \geq f_O(c_x)$

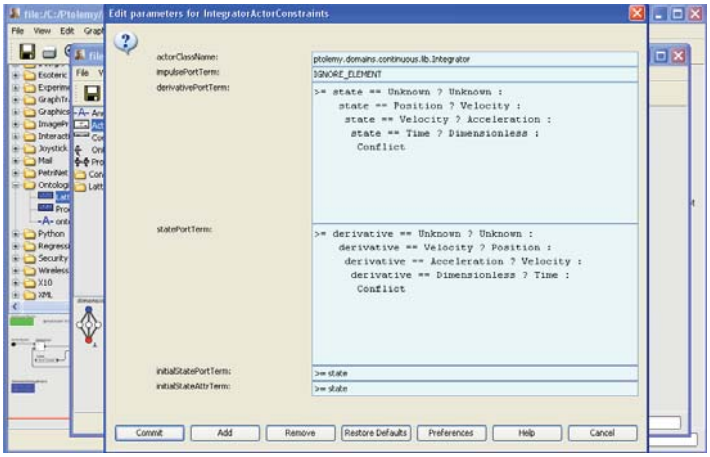
$f_I(c_y) =$

Unknown	If c_y = Unknown
Velocity	If c_y = Position
Acceleration	If c_y = Velocity
Dimensionless	If c_y = Time
Conflict	Otherwise

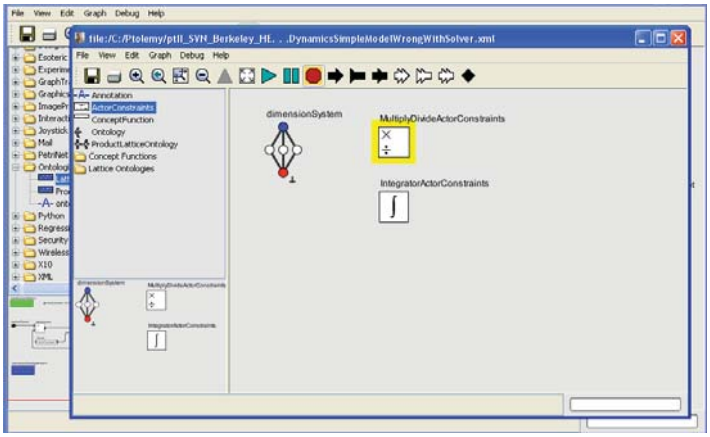
$f_O(c_x) =$

Unknown	If c_x = Unknown
Position	If c_x = Velocity
Velocity	If c_x = Acceleration
Time	If c_x = Dimensionless
Conflict	Otherwise

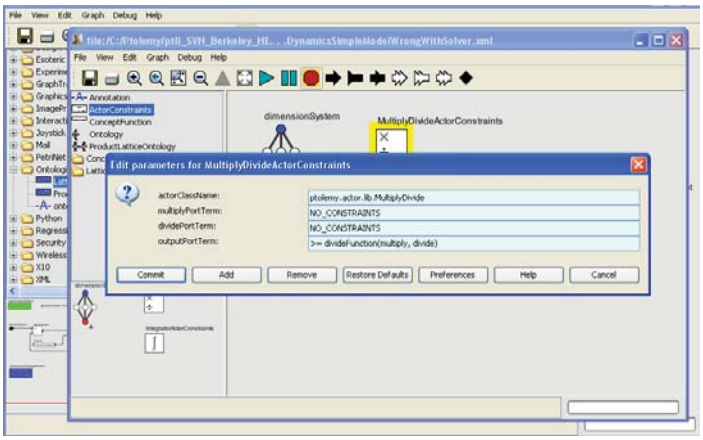
Step 5: Add Actor Constraints to the Solver Model



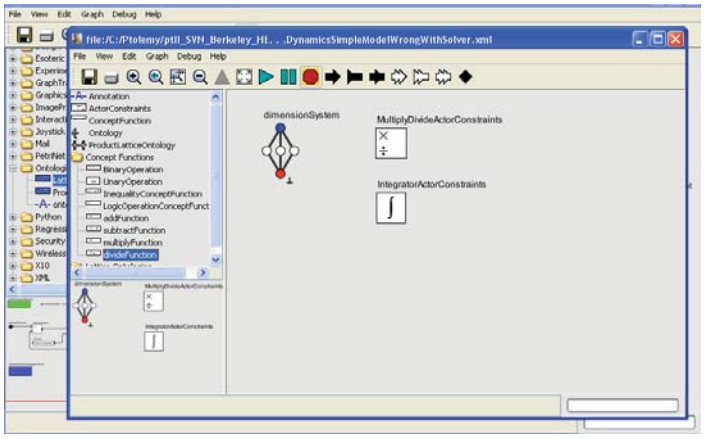
Step 5: Add Actor Constraints to the Solver Model



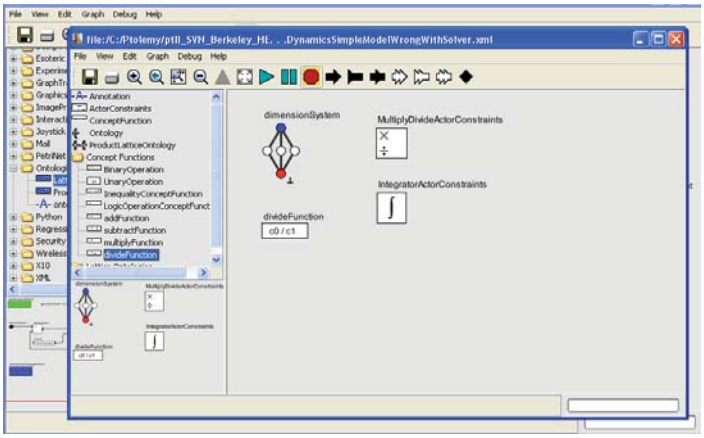
Step 5: Add Actor Constraints to the Solver Model



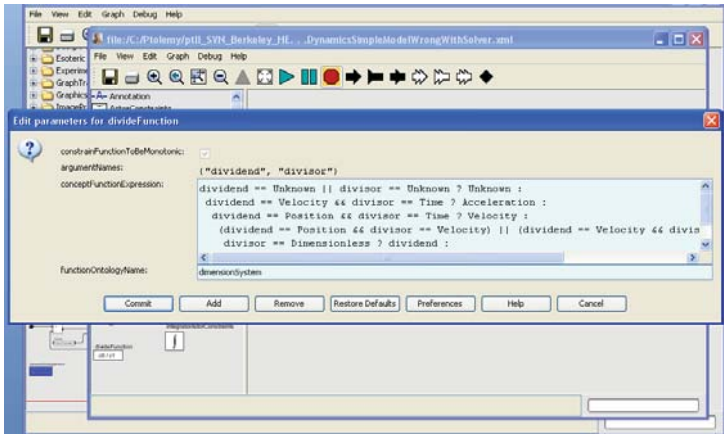
Step 5: Add Actor Constraints to the Solver Model



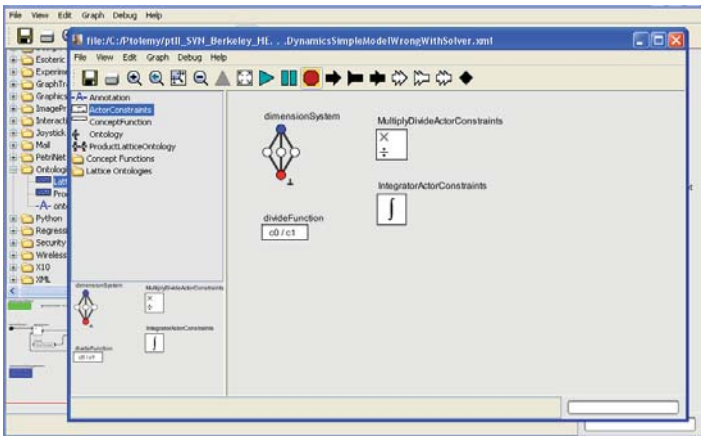
Step 5: Add Actor Constraints to the Solver Model



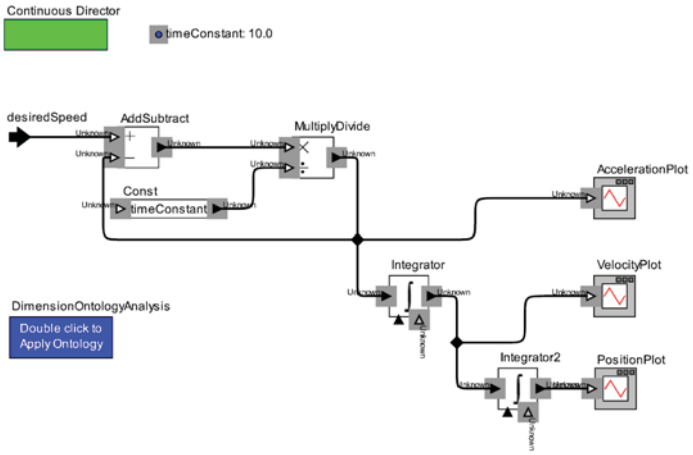
Step 5: Add Actor Constraints to the Solver Model



Step 5: Add Actor Constraints to the Solver Model

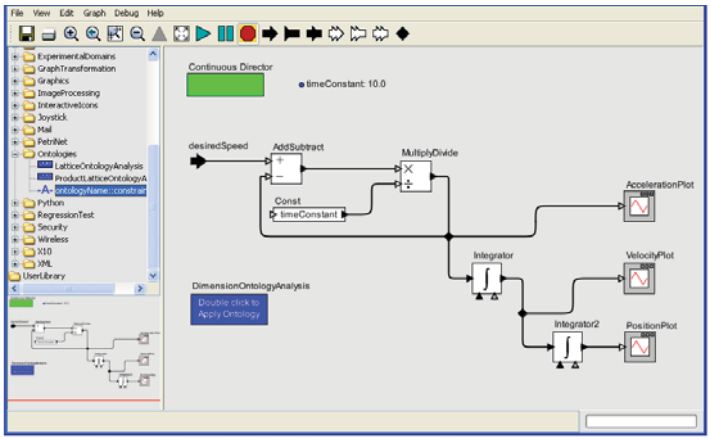


Execute the Lattice Ontology Analysis



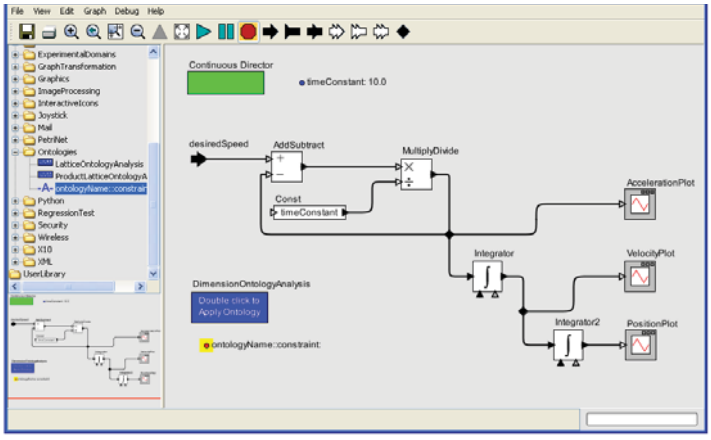
Static Analysis Using Ptolemy II Ontologies

Step 6: Add Initial Constraints to the Model



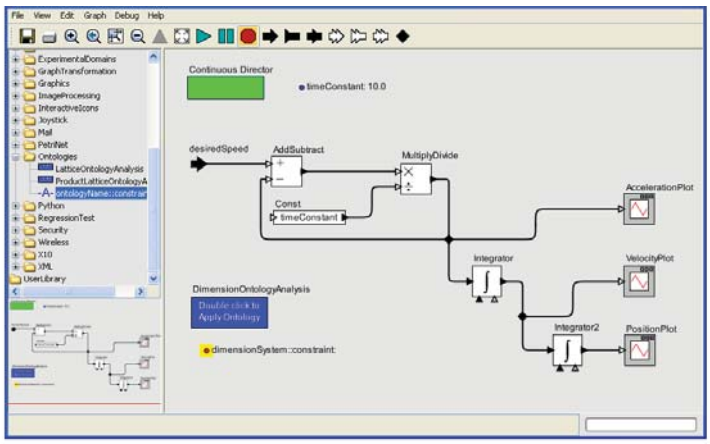
Static Analysis Using Ptolemy II Ontologies

Step 6: Add Initial Constraints to the Model



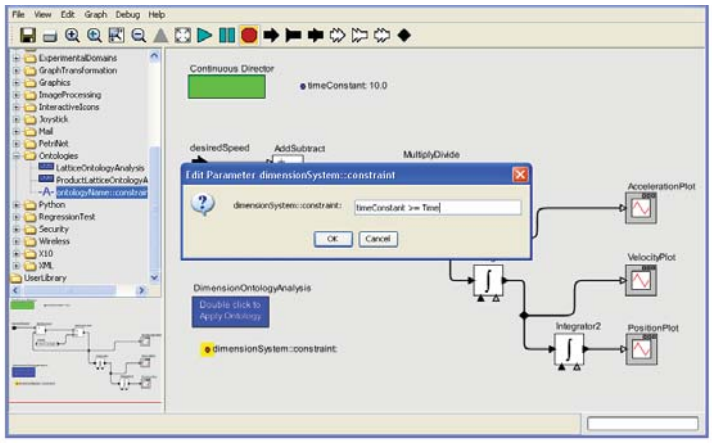
Static Analysis Using Ptolemy II Ontologies

Step 6: Add Initial Constraints to the Model

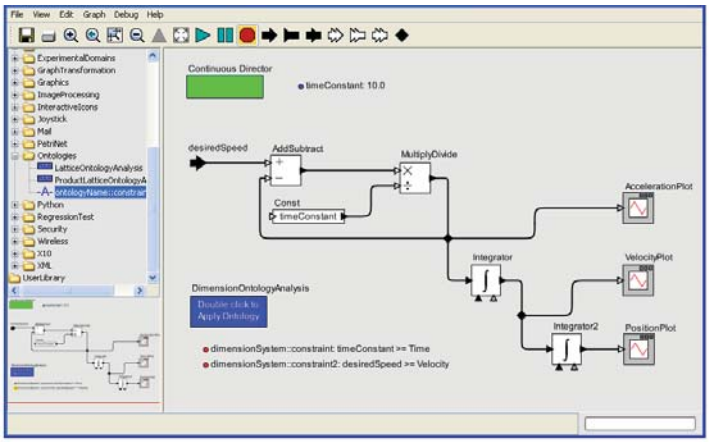


Static Analysis Using Ptolemy II Ontologies

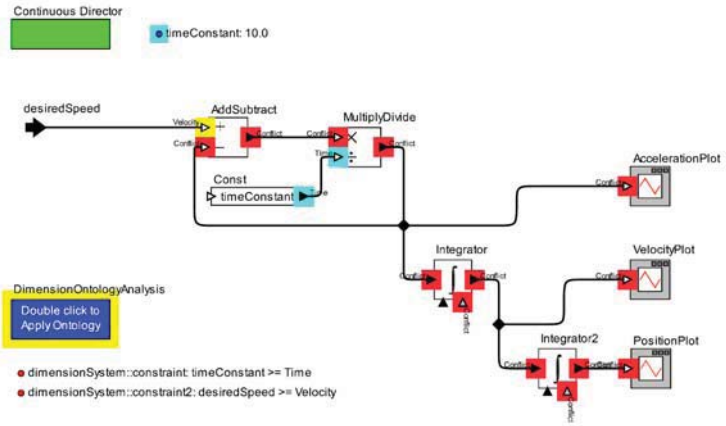
Step 6: Add Initial Constraints to the Model



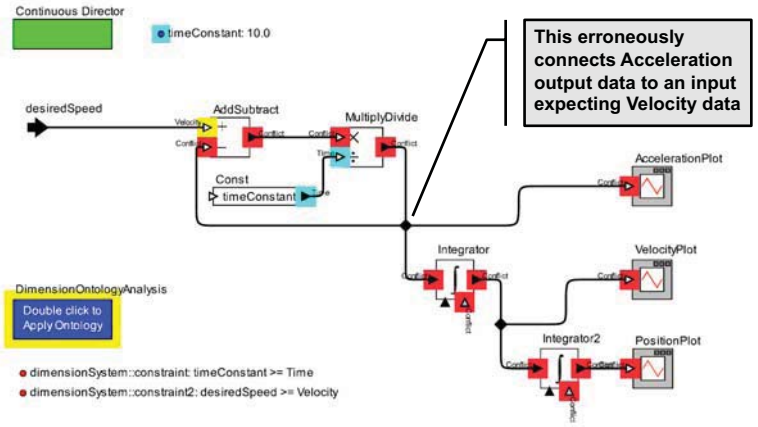
Step 6: Add Initial Constraints to the Model



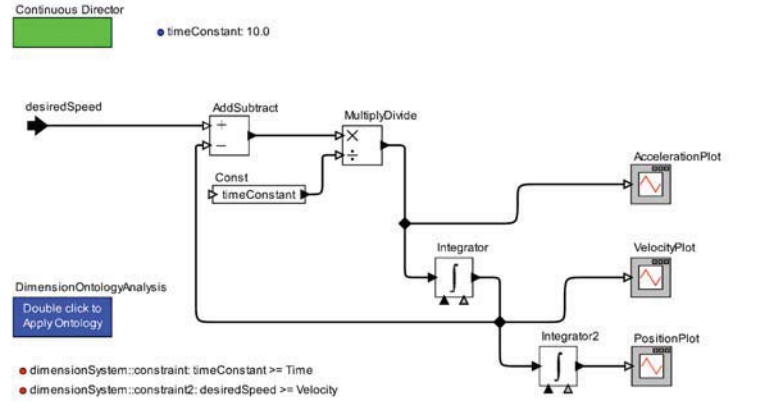
Reexecute the Lattice Ontology Analysis



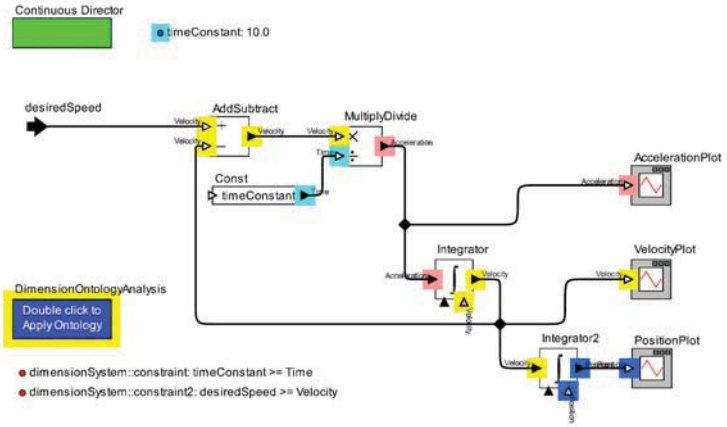
Fix the Model Error and Reanalyze



Fix the Model Error and Reanalyze



Done!



Potential Uses for Ontology-Based Analyses

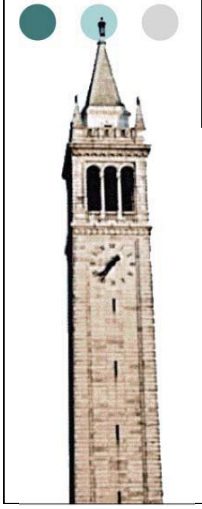
- Type/Semantics Checking
  - Signal Data type Propagation
  - Signal Physical Dimension Propagation
  - Signal Physical/Logical Propagation
  - Signal Data/Control Propagation
- Constant/Non-Constant Propagation
- Reachability
- Observability
- Identify and Propagate Diagnostic/Functional Model Elements

Ongoing Work

- Combining multiple ontology frameworks for integrated analyses
  - **POSTER: Elizabeth Latronico**
- Ontologies with Infinite Lattice Elements
  - Constant value propagation
  - Representing and propagating records of lattice elements
  - **POSTER: Ben Lickly**
- Concept function monotonicity analysis
  - Automatically determine whether or not a function is monotonic
  - Enable easier development of ontology frameworks
- Ontology Error Analysis
  - Identify errors in the model by finding specific constraint conflicts

Conclusions

- Lattice-based ontologies enable automatic static analysis
  - Models can be verified for structural and semantic properties
  - Guaranteed sound analysis given:
    - The ontology is a lattice
    - All constraint functions are monotonic
  - Analysis algorithm scales with the number of constraints
    - # constraints scales with # model elements
- Ontologies Package Demos in the Ptolemy Repository
  - `/ptolemy/data/ontologies/demo`
  - `/ptolemy/data/ontologies/demo/DimensionSystemExample`
  - `/ptolemy/data/ontologies/demo/CarTracking`
- Thanks!
  - Charles.Shelton@us.bosch.com



To Meet or Not to Meet the Deadline

Gage Eads	UC Berkeley
Stephen A. Edwards	Columbia University
Sungjun Kim	Columbia University
Edward A. Lee	UC Berkeley
Ben Lickly	UC Berkeley
Isaac Liu	UC Berkeley
Hiren D. Patel	University of Waterloo
Jan Reineke <speaker>	UC Berkeley

Ninth Biennial Ptolemy Miniconference
Berkeley, CA, February 16, 2011

Abstractions are Great

… if they abstract the right thing

Abstracts from execution time →

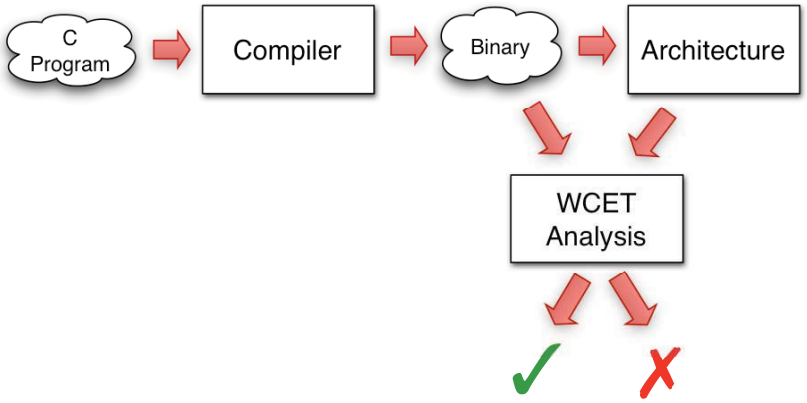
- Higher-level Model of Computation
- C-level programming language
- Instruction Set Architecture (ISA)
- Hardware Realizations

Code Generation

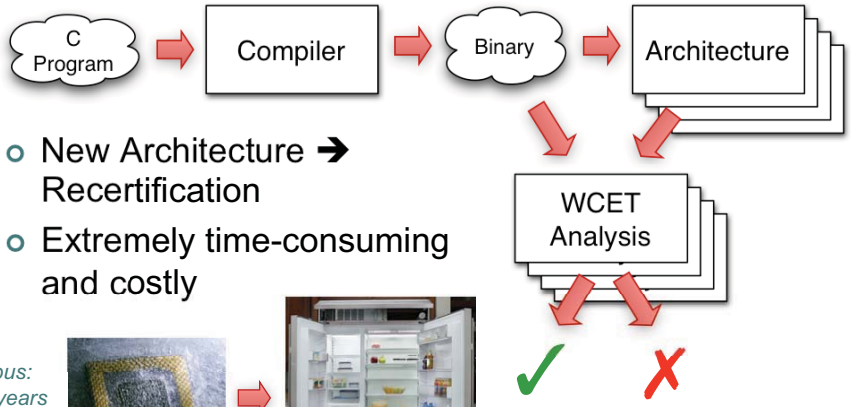
Compilation

Execution

Current Timing Verification Process



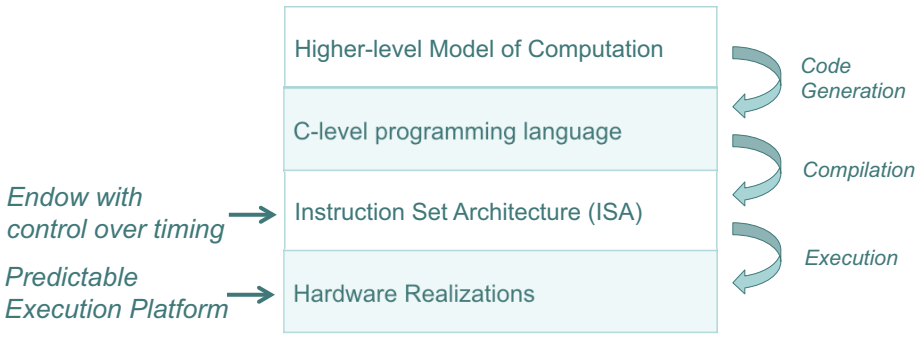
Current Timing Verification Process



- New Architecture ➔ Recertification
- Extremely time-consuming and costly

Airbus: 40 years supply of

Agenda of PRET



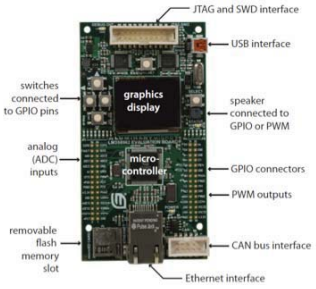
PRET Machines

Make Timing a Semantic Property of Computers

Precision-Timed (PRET) Machines

Timing precision with performance: Challenges:

- Memory hierarchy (scratchpads?)
- Deep pipelines (interleaving?)
- ISAs with timing (deadline instructions?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Precision networks (TTA? Time synchronization?)



See our posters!

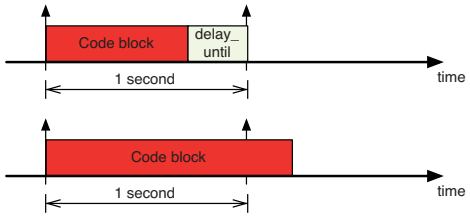
Agenda of this Talk

Adding Control over Timing to the ISA

Variant 1: “delay until”

Some possible capabilities in an ISA:

- [V1] Execute a block of code taking at least a specified *time* [Ip & Edwards, 2006]

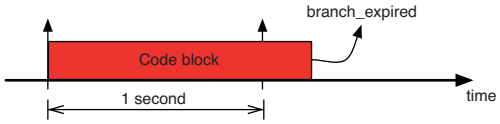


Where could this be useful?

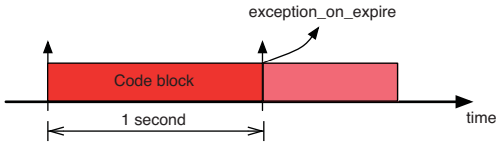
- Finishing early is not always better:
  - Scheduling Anomalies (Graham’s anomalies)
  - Communication protocols may expect periodic behavior
  - …

Adding Control over Timing to the ISA
Variants 2+3: “late” and “immediate miss detection”

- [V2] Do [V1], and then conditionally branch if the specified *time* was exceeded.



- [V3] Do [V1], but if the specified *time* is exceeded during execution of the block, branch immediately to an exception handler.



Applications of Variants 2+3
“late” and “immediate miss detection”

- [V3] “immediate miss detection”:
  - Runtime detection of missed deadlines to initiate error handling mechanisms
  - Anytime algorithms
  - However: unknown state after exception is taken
- [V2] “late miss detection”:
  - No problems with unknown state of system
  - Change parameters of algorithm to meet future deadlines

PRET Assembly Instructions Supporting these Four Capabilities

set_time %r, <val>
- loads current time + <val> into %r

delay_until %r
- stall until current time >= %r

branch_expired %r, <target>
- branch to target if current time > %r

exception_on_expire %r, <id>
- arm processor to throw exception <id> when current time > %r

deactivate_exception <id>
- disarm the processor for exception <id>

Controlled Timing in Assembly Code

[V1] Delay until:

```
set_time r1, 1s
// Code block
delay_until r1
```

[V2] Late miss detection

```
set_time r1, 1s
// Code block
branch_expired r1, <target>
delay_until r1
```

[V3] Immediate miss detection

```
set_time r1, 1s
exception_on_expire r1, 1
// Code block
deactivate_exception 1
delay_until r1
```

[V2] + [V3] could all have a variant that does not control the minimum execution time of the block of code, but only controls the maximum.

Application: Timed Loops

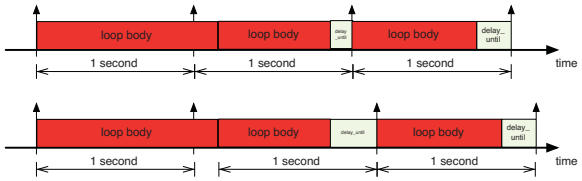
Fixed Period

```
set_time r1, 1s
loop:
// Code block
delay_until r1
r1 = r1 + 1s
b loop
```

Lower bound for each iteration

```
set_time r1, 1s
loop:
// Code block
delay_until r1
set_time r1, 1s
b loop
```

The two loops above have different semantics:



Timed Loop with Exception Handling

Exact execution time (no jitter)

```
set_time r1, 1s
exception_on_expire r1, 0
loop:
// Code block
deactivate_exception 0
delay_until r1
r1 = r1 + 1s
exception_on_expire r1, 0
b loop
```

This code takes exactly 1 second to execute each iteration. If an iteration takes more than 1 second, then as soon as its time expires, the iteration is aborted and an exception handler is activated.

Exporting the Timed Semantics to a Low-Level Language (like C)

```
tryin (500ms) {
  // Code block
} expired {
   patchup();
}
```

```
set_time r1, 500ms
// Code block
branch_expired r1, patchup
```

This realizes variant 2, "late miss detection."

The code block will execute to completion.
If 500ms have passed, then the patchup procedure will run.

Exporting the Timed Semantics to a Low-Level Language (like C)

```
tryin (500ms) {
  // Code block
} catch {
   panic();
}
```

```
jmp_buf  buf;

if ( !setjmp(buf) ){
  set_time r1, 500ms
  exception_on_expire r1, 0
  // Code block
  deactivate_exception 0
} else {
   panic();
}

exception_handler_0 () {
    longjmp(buf)
}
```

This pseudo-code is neither C-level nor assembly, but is meant to explain an assembly-level implementation.

Variant with Exact Execution Times: tryfor

```
tryfor (500ms) {
  // Code block
} catch {
   panic();
}
```

```
jmp_buf  buf;

if ( !setjmp(buf) ){
  set_time r1, 500ms
  exception_on_expire r1, 0
  // Code block
  deactivate_exception 0
  delay_until r1
} else {
   panic();
}

exception_handler_0 () {
    longjmp(buf)
}
```

This is the same, except for the added delay_until

MTFD – Meet the F(inal) Deadline

- Variant [V1] ensure that a block of code takes **at least** a given time.
- Variants [V2, V3] allow to act upon deadline misses.
- [V4] "MTFD": Execute a block of code taking **at most** the specified time.

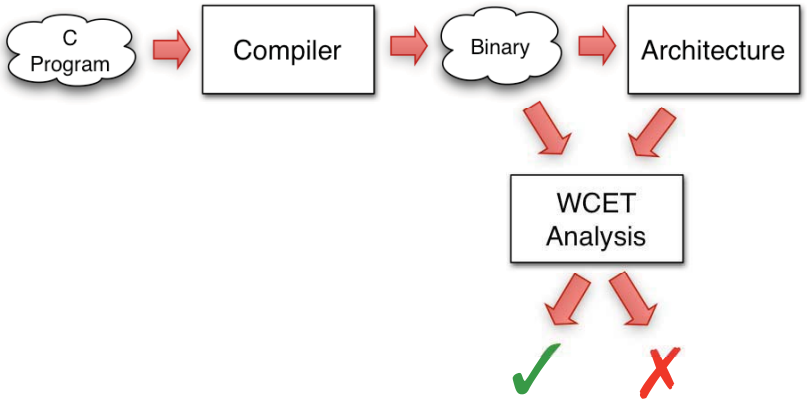
Being arbitrarily "slow" is always possible and "easy".

But what about being "fast"?

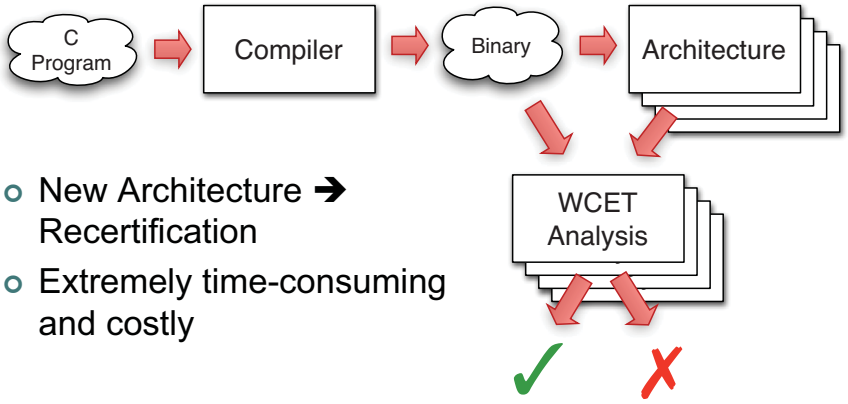
[V4] Exact execution:

```
set_time r1, 1s
// Code block
MTFD r1
delay_until r1
```

Current Timing Verification Process

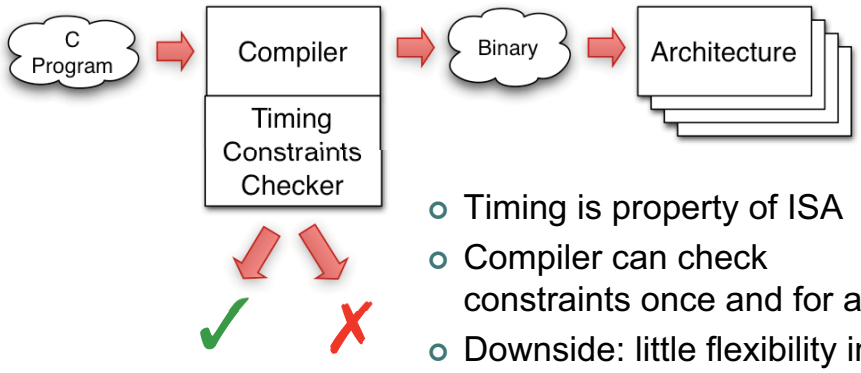


Current Timing Verification Process



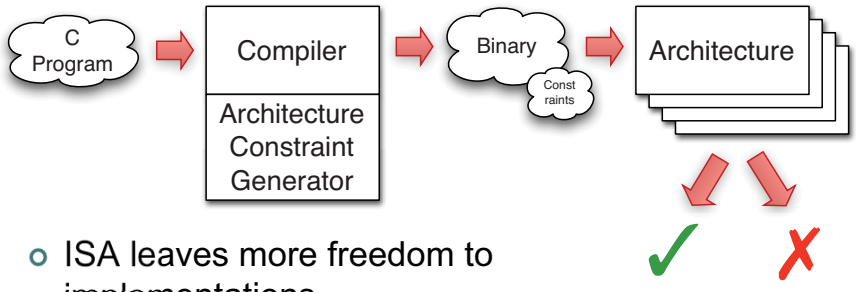
- New Architecture ➔ Recertification
- Extremely time-consuming and costly

The Future (?) Timing Verification Process



- Timing is property of ISA
- Compiler can check constraints once and for all
- Downside: little flexibility in architecture development

The Future (?) Timing Verification Process: More Realistic?



- ISA leaves more freedom to implementations
- Compiler generates constraints on architecture to meet timing constraints

Conclusions

- Abstractions are great, if they are the right abstractions
- Real-time computing needs different abstractions

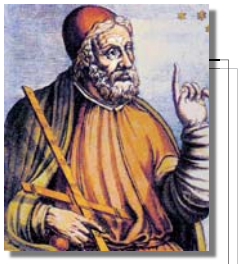
Raffaello Sanzio da Urbino – The Athens School

February 16, 2011
Berkeley, CA

Ptolemy Miniconference

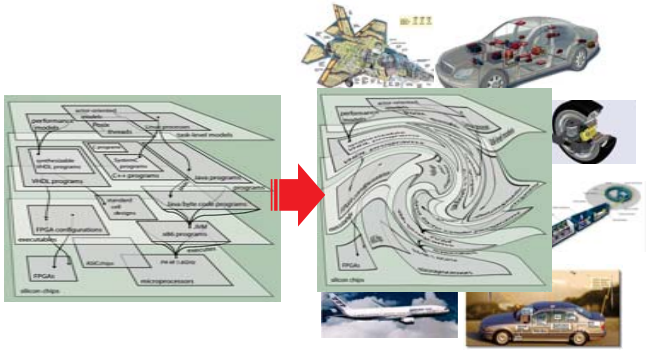
Deadline Instructions in a PRET Architecture

Gage Eads, Edward A. Lee, Isaac Liu, Hiren D. Patel, Jan Reineke



PRET Philosophy

The traditional computing abstractions only concern themselves with the "functional" aspects of a program and not its timing properties. This allows the use of techniques like speculative execution, caches, interrupts, and dynamic compilation that offer improved average-case performance at the expense of predictable execution times. The PRET project aims to improve the timing predictability at all layers of abstraction by carefully reexamining and reworking various architectural and compiler advancements with an eye toward their effects on timing behavior and worst-case bounds.



C Level Constructs

Real-time software engineers require expressive timing constructs at the programming language level. We present three possible control blocks, constructed from PRET deadline instructions: **tryin expired{}**, **tryin catch{}**, and **tryfor{}**.

tryin{} - expired{}

```
tryin(500ms) {
    // Code block to execute
    read_sensor();
}
expired{ patchup(); }
```

Upper Bound (Late Miss Detection)

```
ldr r1, 1                ; Set seconds register to 1
set_time r1, r2          ; Set r1 to current time + r1,
...                      ;Code block to execute
ldr r3, <TARGET>         ; Load r3 with TARGET address
branch_expired r1, r2, r3  ; Jump to TARGET if
                           ; deadline was missed
```

tryin{} - catch{}

```
tryin(500ms) {
    //Code block to execute
    control_actuator();
}
catch{ panic(); }
```

Upper Bound (Immediate Miss Detection)

```
ldr r1, 1                ; Set seconds register to 1
set_time r1, r2          ; Set r1 to current time + r1,
exception_on_expire r1, 1 ; Enable exception ID #1
...                      ;Code block to execute
deactivate_exception 1   ; Deactivate exception ID #1
```

tryfor{}

```
tryfor(500ms) {
    //Code block to execute
    read_sensor();
}
catch{ panic(); }
```

Lower Bound

```
ldr r1, 1                ; Set seconds register to 1
set_time r1, r2          ; Set r1 to current time + r1,
...                      ;Code block to execute
delay_until r1, r2       ; Stall until lower bound met
```

Assembly Timing Instructions

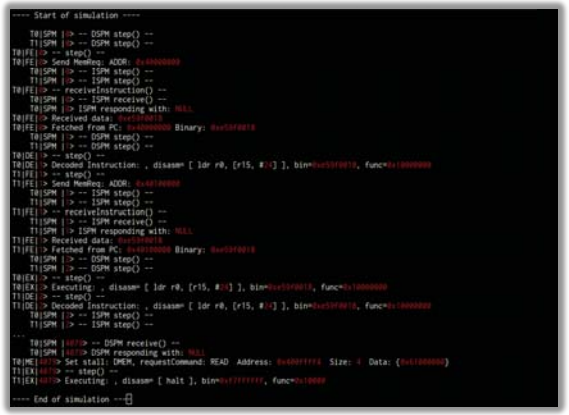
We augment the ARM ISA with five powerful timing instructions. The instructions are implemented as coprocessor accesses to the 64-bit timer coprocessor, which is partitioned into a 32-bit second counter and a 32-bit nanosecond counter. Register operands *rd* and *rm* contain the **second** and **nanosecond** values, respectively.

Assembly Instruction	Behavior
SET_TIME rd, rm	Loads the current timer value plus register contents into the register.
DELAY_UNTIL rd, rm	Stall CPU until timer value > *rd* + *rm*
BRANCH_EXPIRED rd, rm, rn	Branch to target address in register *rn* if timer value > *rd* + *rm*
EXCEPTION_ON_EXPIRE rd, <id>	Arm the processor to throw an exception immediately with id = <id> when the timer > *rd* + *rm*
DEACTIVATE_EXCEPTION <id>	Deactivate exceptions of id = <id>

We design timing constructs such as lower and upper bounded execution through a combination of PRET timing instructions. A fourth construct, MTFD, would statically determine whether a code block can meet its deadline on a given predictable system.

PRET Simulator

We released the PRET Simulator v1.0 in February 2009. A number of class projects and papers leveraged the simulator to explore precision timed software design, such as an automated mapping from a timed functional specification (Giotto) to a PRET architecture.



Debug output from PRET Simulator running threads 0 and 1

Since then, we have made significant changes to the PRET architecture: the ISA changed from SPARCv8 to ARMv4, a predictable memory controller subsumed the memory wheel, and the core became a 4-stage pipeline. Version 2.0 of the PRET Simulator is functional and features all of the aforementioned deadline instructions. It will be released for public use under an open-source license in the near future.

Acknowledgments

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota. The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program.

CHESS

CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

Error Handling in Model-Based design for Real-Time Systems

Shanna-Shaye Forbes
Edward A. Lee

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

Work In Progress

Introduction

Designing effective error handling in an embedded software systems is essential for acceptable and reliable functionality in cases of errors and for the recovery from faults.

Errors in the error handling system can cause catastrophic failures of the software and can endanger human life. We take a principled approach of extending a model of computation (MOC) with timing semantics for embedded systems by an error handling mechanism for timing errors in model-based design.

Background

It is extremely important that one identifies and is capable of handling error cases before deploying embedded software.

One ineffective solution that has been used in the past is to have the system reset itself for every error encountered. If a system resets itself too often this can lead to significant loss of productivity, possible loss of data, and in extreme cases, possible loss of life.

Examples of the importance of error handling:

In 1996 the European Space Agency's Ariane 5 rocket self-destructed 40 seconds after launch. The underlying cause of the self-destruct sequence was a 64-bit floating point to 16-bit integer conversion exception. This occurred because of reuse of code designed for the much smaller Ariane IV [1] .

Image Source: http://www.josefdobler.de/Neuer%20Ordner/launch-site/ami33h.jpg

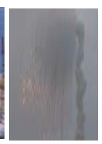


Image Source: http://sspg1.bnsc.rl.ac.uk/SEG/Ariane%205%20Explosion%20Report.htm

More recently, the SPIRIT Mars rover encountered a "reboot loop" shortly after landing, where a fault during the booting process caused the system to reboot again Luckily, a software patch solved the problem and the mission continued successfully[2].

Image Source: http://en.wikipedia.org/wiki/File:NASA_Mars_Rover.jpg

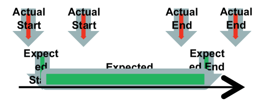
A Royal Airforce pilot accidentally dropped a practice bomb on the flight deck of the Royal Navy's aircraft carrier. It missed it's intended target and several sailors were injured. The cause was attributed to a timing delay in the software.

In an effort to avoid possible similar mistakes with newly designed systems and to provide a more systematic means of dealing with timing errors, we present preliminary work that extends a model of computation (MOC) for embedded systems which features timing semantics.

Methodology

The focus of this work is on timing errors.

A timing error occurs when the specification says one thing and the implementation does something else. Often times, this is caused by execution at a time that violates a specification.



Model-based design, simulation, and synthesis is being used more than before in lieu of hand writing code and testing it [4].

There is also a resurgence in Cyber Physical System design due to renewed interest in the area

If these trends continue, we will see:

1. **More use of model-based design with timing specifications in the design of Cyber Physical Systems;**
2. **The desire to include error handling explicitly in a model instead of in an ad hoc manner.**

Objective

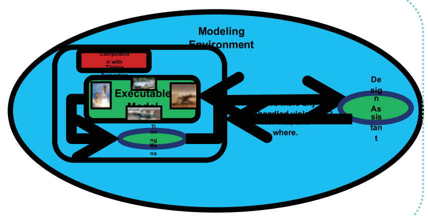
Add meaning to what is done in the event of a timing error. We achieve this by :

1) Extend concepts from real-time programming languages to model-based design.

* Exception handling

2) Adding concepts to hierarchical state machines

* Error Transitions



Timing Manager

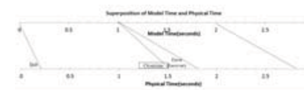
1. Notes desire to incorporate physical time into the model
2. Annotates actors with a execution time estimate parameter. Values are expected to be provided by the designer or by an external execution time analysis tool
3. Simulates execution time of actors as a probabilistic variant of annotated execution time estimate parameter value
4. If a timing error is detected the timing manager passes the error up the model hierarchy. If the specification is included in a modal model the timing manager enables the first applicable error transition. If not it moves further up the hierarchy and attempts to enable an error transition
5. If there is not error transition to catch the error in the hierarchy an exception the simulation of the model is stopped and the user is informed of the unhandled error

Discussion and Future Work

Error Handling in Model-Based design for Real-Time Systems is still a work in progress. Preliminary results indicate that the current mechanism is able to detect and appropriately transition after simulating a timing overrun.

Features Added to Ptolemy II

1. Error Transition for timing errors into modal models[3].
2. Timing manager to introduce a secondary notion of time and handle errors hierarchically.
3. Design Assistant to aid the user
4. Preliminary code generation support for the timing manager.



Future Work

2) incorporating representative probabilistic distributions into the timing
3) expanding the preliminary work in C and Java code generation
Allowing the user to specify a recovery transition
Adding in other types of timing error transitions

Code generation support for all features of the timing manager.

Timing manager extensions to provide suggestions for the scheduling strategy used with the MOC.

References

[1] ARIANE 5: Flight 501 Failure. http://sspg1.bnsc.rl.ac.uk/SEG/Ariane%205%20Explosion%20Report.htm
[2] Mars Spirit Wiki. Mars Spirit Software Problem. http://c2.com/cgi/wiki?MarsSpiritSoftwareProblem.
[3]C. J. Murray. Automakers opting for model-based design. Design News: http://www.designnews.com/article/511392-Automakers_Opting_for_Model_Based_Design.php, November 5 2010.
[4] Edward A. Lee, Stavros Tripakis. "Modal Models in Ptolemy". Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2010), 1-11, October, 2010.

Acknowledgements

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota.

This work was also supported by the Jenkins Pre-doctoral Fellowship program as well as a Jenkins Pre-doctoral Fellowship Program Mini Grant

FORMLESS: Scalable Utilization of Embedded Manycores in Streaming Applications

Matin Hashemi, Soheil Ghiasi
Electrical and Computer Engineering Department
University of California, Davis

http://leps.ece.ucdavis.edu

Throughput Estimation:

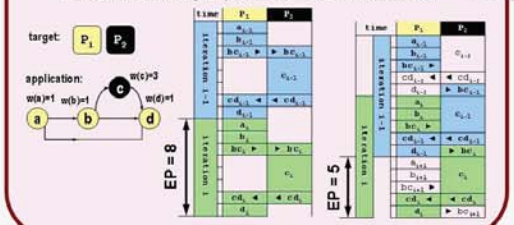
Theorem: Assuming large inter-processor buffers, for any task assignment of dataflow graph G to P processors, there exists an ordering of tasks on processors such that every precedence constraints is met (possibly by overlapping iterations), and the steady state execution period (inverse of throughput) of the application is $EP = \max_{1 \le p \le P} \text{workload}(p)$.

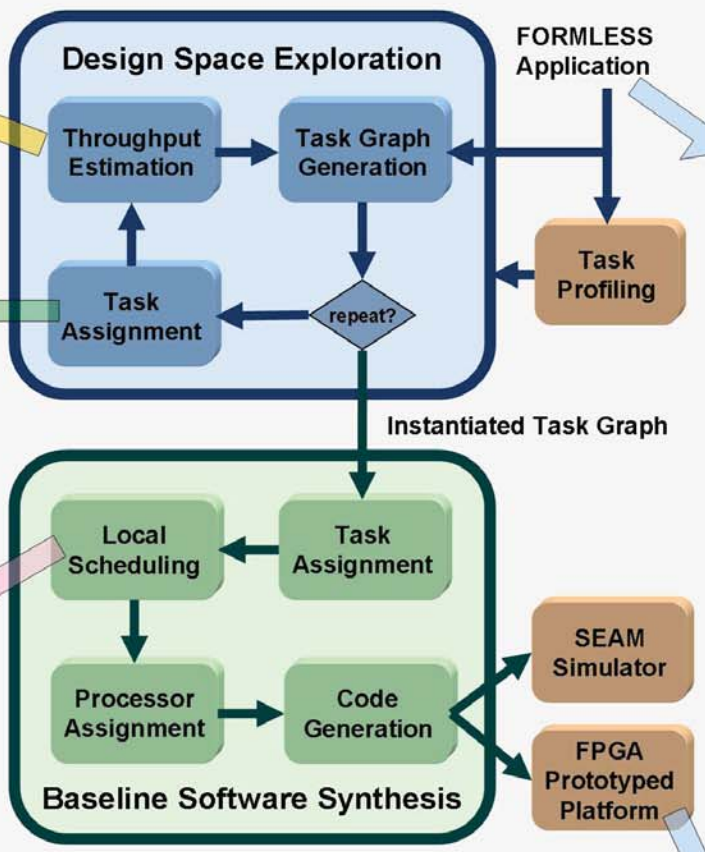
Task Assignment

- Objective: minimize execution period EP
- Constraint: each processor should have at most 4 connections from and 4 connections to other processors
- Implementation:
  - METIS graph partitioning software
  - Post partitioning adjustment to meet the above constraint

Local Scheduling

- Overlap execution of different iterations of the application tasks
  - Mush respect the dependencies
  - Increases throughput
  - Increases memory requirement for inter-task buffers






```
//We would like to optimize a merge sort task graph
//which sorts an array of N elements. The rest of the
//parameters should be decided by the DSE tool.
 set K = {2,+1,4}; //2..4
 set D = {0,+1,log(N,K)}; //0..log(N,K)
 optimize MergeSort ( N, K, D );

//sort an array using a ScatterNetwork to distribute the input
//array among a number of QuickSort actors, and a MergeNetwork
//to merge the sorted sub-arrays into the final output array:
composite_actor MergeSort ( int N, //length of input and output
                            int K, //in-degree of merge actors
                            int D  //depth of the tree
                          ){
  interface {
    input input_array ( N );
    output output_array ( N );
  }
  composition {
    if (D==0) {
      instantiate QuickSort quicksort ( N );
      connect ( input_array,  quicksort.unsorted_array );
      connect ( quicksort.sorted_array,   output_array );
    } else {
      //instantiate a scatter network which distributes
      //the input array among K^D sort actors
      instantiate ScatterNetwork scatternet( N, K, D, explore);

      //instantiate the sort actors
      for (i=0; i < K^D; i++)
        instantiate QuickSort quicksort[i] ( N / K^D );

      //instantiate a merge network which merges
      //the sorted sub-arrays into the output array
      instantiate MergeNetwork mergenet ( N, K, D, explore );

      //connections:
      connect ( input_array, scatternet.input_array );
      for (i=0; i < K^D; i++) {
        connect ( scatternet.sub_array[i],
                  quicksort[i].unsorted_array );
        connect ( quicksort[i].sorted_array,
                  mergenet.sub_array[i] );
      }
      connect ( mergenet.merged_array, output_array );
    }
  }
```

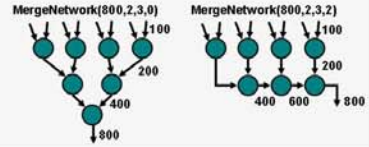
```
//sort an array:
actor QuickSort ( int N //length of input and output arrays
                ){
  interface {
    input  unsorted_array ( N );
    output sorted_array ( N );
  }
  function {
    //the quicksort algorithm
  }
}
```


```
//merge input sorted arrays into one larger output array:
actor Merge ( int   N, //length of the output array
              int   K, //number of the input arrays (in-degree)
              int[] M  //length of the input arrays
            ){
  interface {
    output merged_array ( N );
    for ( i=0; i < K; i++ ) {
      if (M != null)  m = M[i];
      else            m = N/K;
      input sub_array[i] ( m );
    }
  }
  function {
    //merge based on the method in mergesort algorithm
  }
}
```

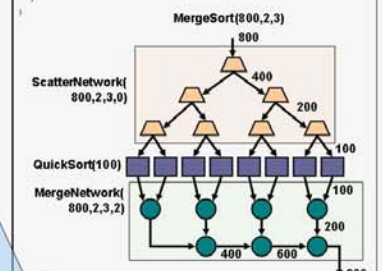

```
//merge input sorted arrays into one larger output array
//using a tree network constructed out of merge actors:
composite_actor MergeNetwork ( int N, //length of the output array
                               int K, //in-degree of merge actors in the tree
                               int D, //depth of the tree
                               int L  //the first L levels of the tree are
                                      //constructed in a linear structure.
                             ){
  explore(
    set L = {0,+1,D};
  )
  interface {
    output merged_array ( N );
    for (i=0; i < K^D; i++)
      input sub_array[i] ( N / K^D );
  }
  composition {
    //instantiate merge actors in tree structure (levels L to D-1)
    for (d=D-1; d >= L; d--)
    for (i=0; i < K^d; i++)
      instantiate Merge merge[d][i] ( N / K^d, K, null );

    //instantiate merge actors in linear structure
    int S = N / K^L;
    for (i=1; i < K^L; i++)
      instantiate Merge merge[0][i] ( S*(i+1), 2, (S*i, S) );

    //connections:
    //input
    for (i=0; i < K^D; i++)
      connect ( sub_array[i], merge[D-1][i/K].sub_array[i%K] );
    //tree structure
    for (d=D-1; d > L; d--)
    for (i=0; i < K^d; i++)
      connect ( merge[d][i].merged_array, merge[d-1][i/K].sub_array[i%K] );
    //linear structure
    for (i=1; i < K^L; i++) {
      if (i==1) connect ( merge[L][0].merged_array,   merge[0][i].sub_array[0] );
      else      connect ( merge[0][i-1].merged_array, merge[0][i].sub_array[0] );
      connect ( merge[L][i].merged_array, merge[0][i].sub_array[1] );
    }
    //output
    connect ( merge[0][K^L-1].merged_array, merged_array );
  }
}
```

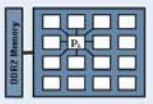


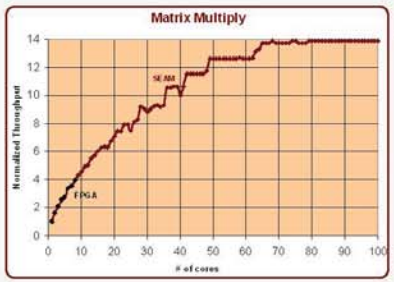

```
composite_actor ScatterNetwork //details removed for brevity
```



FPGA Prototyped Platform

- Altera DE4 FPGA board
- 16 NiosII/f cores
- instruction cache: 8KB
- data cache: 32KB
- FIFO depth: 1024





CHESS

CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

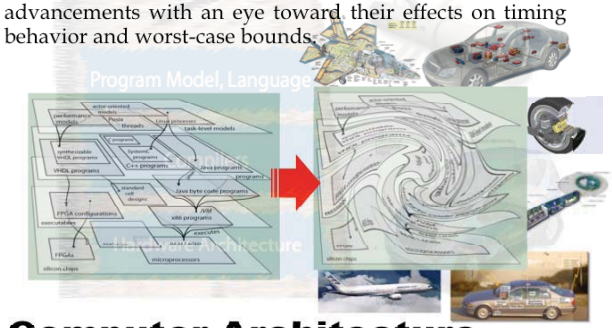
Precision Timed Machines (PRET)

Isaac Liu
Jan Reineke
Sungjun Kim
Hiren Patel
Edward A. Lee

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

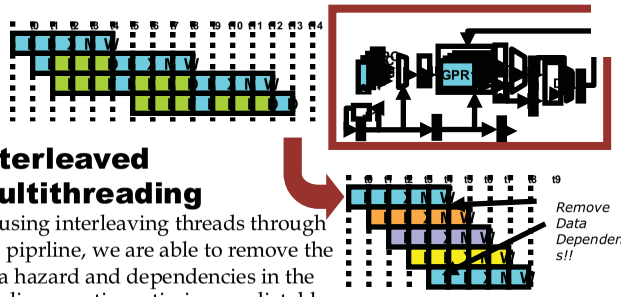
Mission

The traditional computing abstractions only concern themselves with the "functional" aspects of a program and not its timing properties. This allows the use of techniques like speculative execution, caches, interrupts, and dynamic compilation that offer improved average-case performance at the expense of predictable execution times. The PRET project aims to improve the timing predictability at all layers of abstraction by carefully reexamining and reworking various architectural and compiler advancements with an eye toward their effects on timing behavior and worst-case bounds.



Computer Architecture

In our research, we have pursued a bottom-up approach. Starting with the underlying pipeline, we have modified and added constructs to improve the timing predictability at the architectural and instruction-set-architecture level



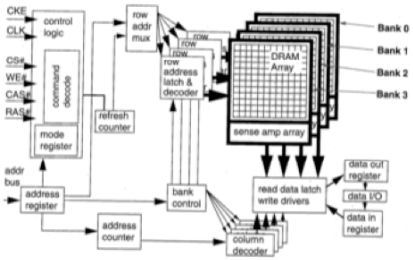
Interleaved multithreading

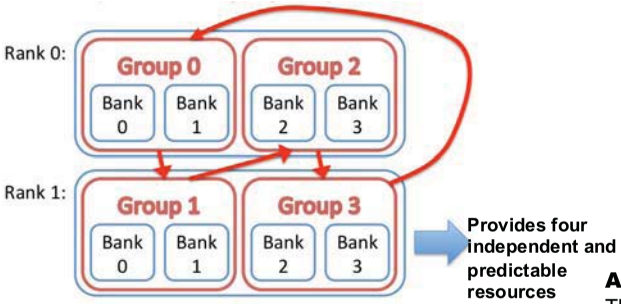
By using interleaving threads through our piprline, we are able to remove the data hazard and dependencies in the pipeline, creating a timing predictable pipeline.

Memory Hierarchy

Conventional memory systems uses a hierarchy of memory units to bridge the latency. CPUs use registers for fast data processing operations. However, the memory hierarchy is designed as "best effort" latency and bandwidth requirements. Our goal is to look at modern memory hiearchies and design a predictable memory system.

A DRAM device consists of several banks along with controller logic to decode addresses. Concurrent accesses to banks are possible, but I/O pins are shared.





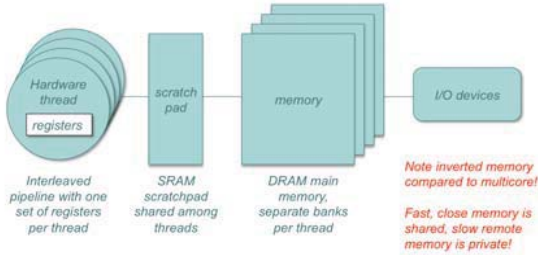
Provides four independent and predictable resources

Predictable DRAM Access

DDR2 DRAM devices utilizes bank parallelism to achieve better performance. It's difficult to predict DRAM access time because accesses to the same bank need to wait for the previous access to finish, while accessing a different bank can be done concurrently. Our DRAM controller exposes groups of banks as independent resources that are accessed sequentially which hides the latency of accessing a single bank. A TDMA scheduler provides predictable access latencies.

Scratchpad Memories

Caches can use different hardware replacement policies in attempt to prefetch the data needed from main memory. However, when the software issues a load or store instruction, it does not know the state of the cache, or what data has been prefetched in it. Scratchpad memories are another form of fast access memory which is managed in software, much like registers that use explicit load/store instructions to manipulate contents for fast data processing. With software management, better real time guarantees can be provided.



Above shows one possible memory hierarchy for PRET. The memory system is a major source of headache for analyzing or predicting execution, as it causes a wide range of execution time for even the same programming running on the same system. Our goal is to design a predictable memory system that provide systems with predictable memory access times.

Acknowledgement

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota.

CHESS
CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

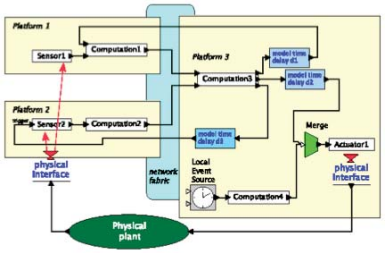
The Distributed Power System Test Case for Distributed Real-Time Systems

John Eidson
Slobodan Matic
Casidhe Lee
Ilge Akkaya

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

Hardware for this research is provided by National Instruments and Renesas

Distributed Real-Time System Implementation

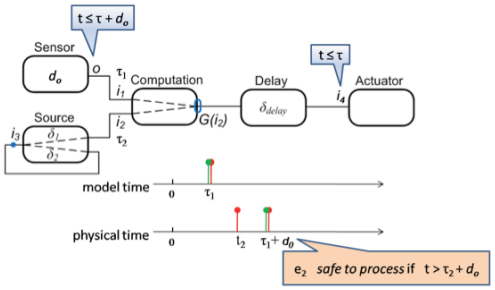


Programming Model

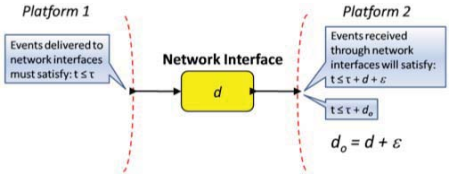
Programming **T**emporally **I**ntegrated **D**istributed **E**mbedded **S**ystems

Based on Discrete-Event semantics Event processing in time-stamp order

Relates *model time* to *physical time* at environment interfaces
d_o – sensor latency

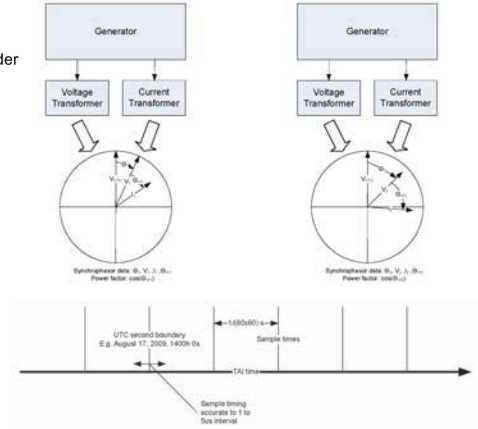


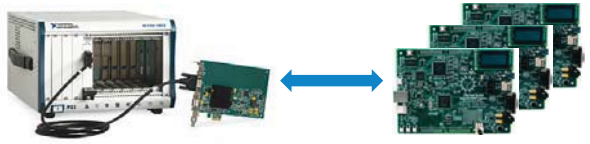
Leverages time synchronization across distributed platforms (IEEE 1588 protocol over Ethernet)
d – comm. latency, ε – sync. error

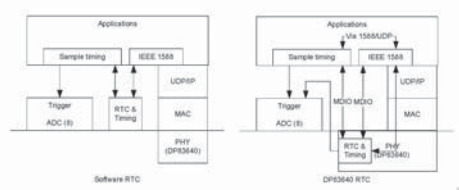


Distributed Testbed Objectives

- Safe event processing
  Bounded latency between hardware components
  Distribution: bounded-delay networking protocol
- Expressiveness of interaction
  Complexity of real sensor, actuator and network interfaces
- Timing support
  External to microprocessor
  National Semiconductor DP83640: IEEE 1588 enabled PHY chip
- Scheduling
  Deadlines defined by sensor – actuator model delays
  Distribution: local from end-to-end deadlines
- Network addressing
- Visibility during model-based design







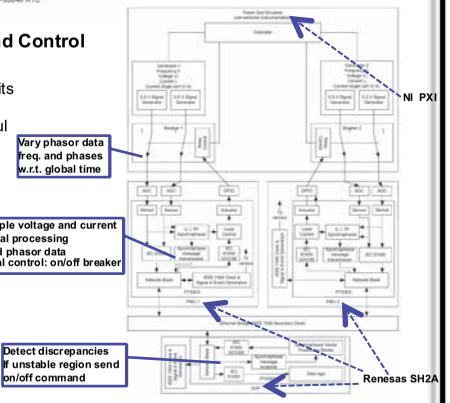
DP83640: PHY chip with PTP

Clock (read/write/adjust/scale)
TX Pkt Parser and TS unit
RX Pkt Parser and TS Unit
8 Triggers
8 Event TS
6 GPIOs4

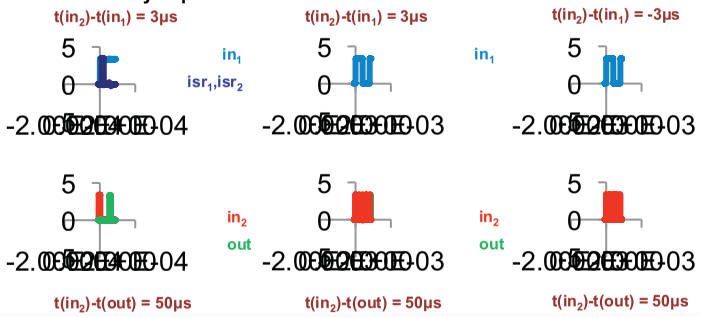
Synchrophasor-based Measurement and Control

- Frequency and phase of power generation units must remain synchronous
- Synchrophasor technology provides a powerful tool to directly measure the state
- PMU = Primary Measurement Unit
  SVP = Synchrophasor Vector Processing unit

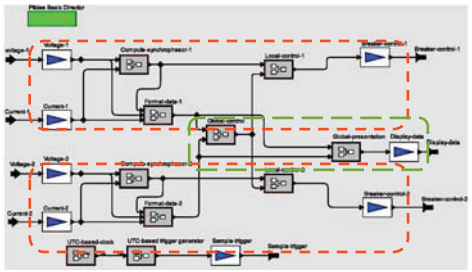
Real PTIDES problem
- Tight (1-5us) timing accuracy
- Reasonable sample rate (4.8 KSPS)
- Timing based on UTC
- Multiple functions run concurrently (UDP, PTP)
- Naturally distributed due to physical size of grid



Preliminary Experiments



Ptolemy Approximation Model



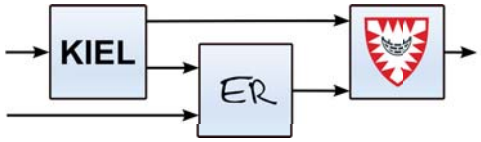
tfrrr Institut für Informatik

C | A | U

Christian-Albrechts-Universität zu Kiel

Faculty of Engineering

Department of Computer Science



Kiel Integrated Environment for Layout

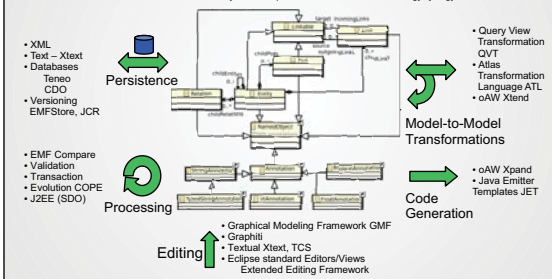
Eclipse Rich Client

KIELER Actor Oriented Modeling

Meta Modeling

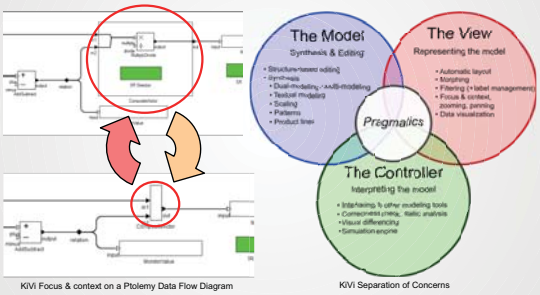
KAOM – KIELER Actor Oriented Modeling

Metamodel-based abstract syntax in Eclipse EMF enables a broad technology synergy



View Management

KiVi – KIELER View Management



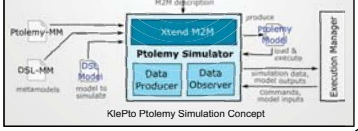
KiVi Focus & context on a Ptolemy Data Flow Diagram

KiVi Separation of Concerns

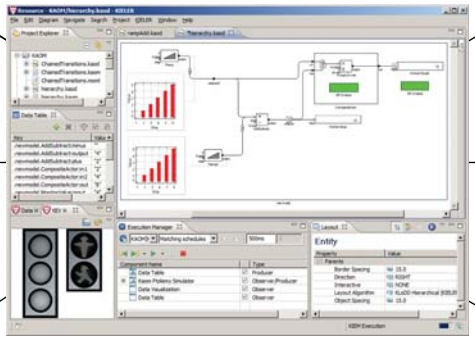
Model Execution

KIEM – KIELER Execution Manager [3]

Overview of the Execution Manger Infrastructure

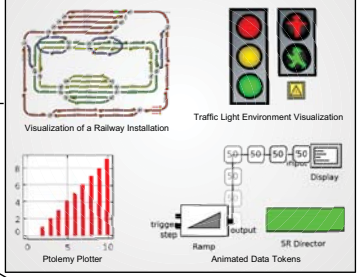


KlePto Ptolemy Simulation Concept



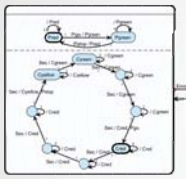
KAOM with Automatic Layout, Ptolemy rendering and Simulation in KIELER

Data Visualization

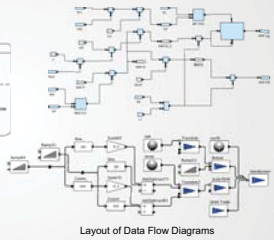


Automatic Layout

KIML – KIELER Infrastructure for Meta Layout [2, 5]



Layout of a State Chart Diagram

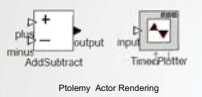


Layout of Data Flow Diagrams

- Automatic layout of GMF and Graphiti diagrams
- Generic interface for layout algorithms
- Flexible configuration of layout options by the user
- Automatic layout integrated into Ptolemy's graphical editor Vergil

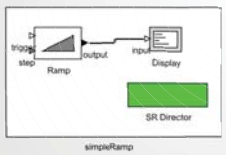
Model Rendering

KARMA – KIELER Advanced Rendering for Model Appearance



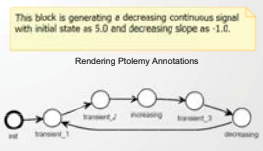
Ptolemy Actor Rendering

- Possible customized model rendering options:
  E.g., Ptolemy, LabView, Simulink, ASCET, SCADE
- GMF editor support, Graphiti (planned)



simpleRamp

Rendering Ptolemy Data Flow



Rendering Ptolemy Annotations

Rendering Ptolemy State Machines

Contact Persons:
Miro Spönemann / Christian Motika
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40, 24098 Kiel, Germany
Phone: +49 (0) 431 880-7282 /-7526
Fax: +49 (0) 431 880-7615
msp@ / cmot@informatik.uni-kiel.de
http://www.informatik.uni-kiel.de/rtsys

Contact Person:
Prof. Dr. Reinhard von Hanxleden
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40, 24098 Kiel, Germany
Phone: +49 (0) 431 880-7281
Fax: +49 (0) 431 880-7615
rvh@informatik.uni-kiel.de
http://www.informatik.uni-kiel.de/rtsys

Further Information:
http://www.informatik.uni-kiel.de/rtsys/kieler

[1] Hauke Fuhrmann and Reinhard von Hanxleden. Taming Graphical Modeling. In Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10), LNCS, Oslo, Norway, 2010. Springer.

[2] Miro Spönemann, Hauke Fuhrmann, Reinhard von Hanxleden, and Petra Mutzel. Port constraints in hierarchical layout of data flow diagrams. In *Proceedings of the 17th International Symposium on Graph Drawing (GD'09)*, LNCS, Chicago, September 2009.

[3] Christian Motika, Hauke Fuhrmann and Reinhard von Hanxleden. Semantics and Execution of Domain Specific Models. In 2nd Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe 2010) at conference INFORMATIK 2010, GI-Edition – Lecture Notes in Informatics (LNI), Leipzig, Germany, 2010.

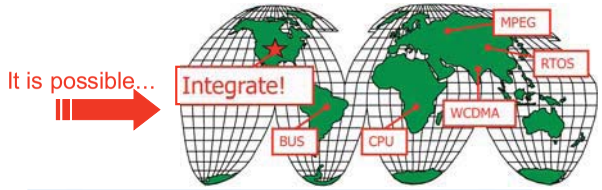
[4] Reinhard von Hanxleden. SyncCharts in C—a proposal for light-weight, deterministic concurrency. In *Proceedings of the International Conference on Embedded Software* (EMSOFT'09), Grenoble, France, October 2009.

[5] Miro Spönemann, Hauke Fuhrmann and Reinhard von Hanxleden. Automatic Layout of Data Flow Diagrams in KIELER and Ptolemy II. Technical Report 0914, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, July 2009.

Poster presented at 9th Biennial Ptolemy Miniconference (PTCONF'11), Berkeley, CA, February 2011,

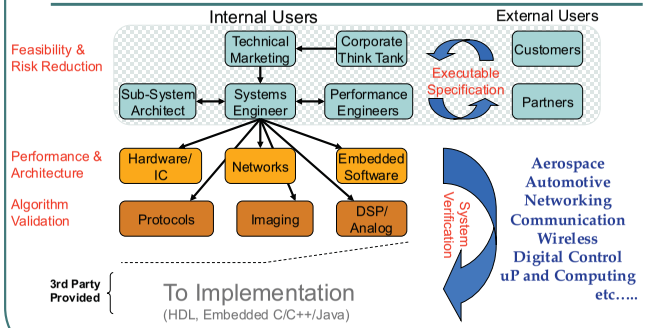
Challenges - IP Re-Use

- Internet reduced IP distribution cost, But …
  - No standard modeling interface and format
  - Several abstraction-levels and domains
  - No unified design and simulation environment



Non-standard formats prevent IP reuse over the Internet

VisualSim Solution Model

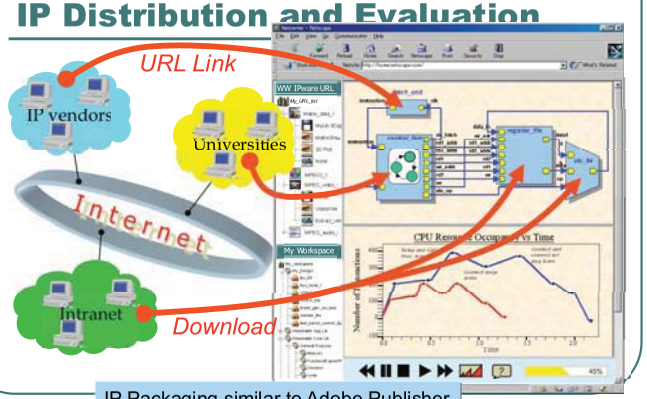


Integrate requirements, design, verification & management

Using Ptolemy/VisualSim for Internet-based Model Sharing & Communication

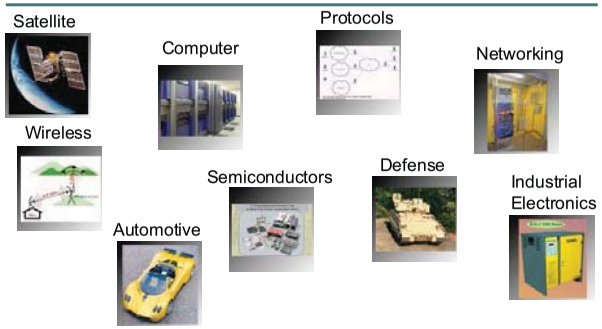
Darryl Koivisto
Chief Technology Officer
dkoivisto@mirabilisdesign.com

IP Distribution and Evaluation



IP Packaging similar to Adobe Publisher

What is a System?



System is what you are building

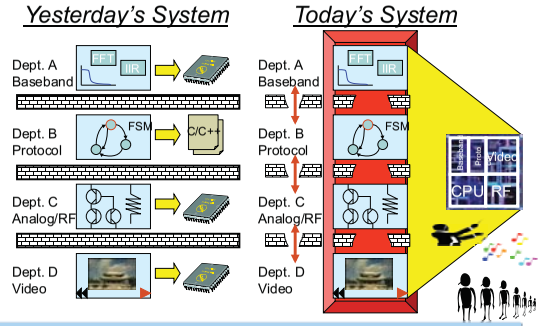
Mission Statement

System Simulation Software
with
Application-specific Modeling Libraries
for
Performance and Architecture Exploration
of
Large Complex Electronic Systems

Innovative methodology for addressing today's Ad-Hoc Approaches

Creating the Right Design

Challenges - Convergence



Convergence of Multimedia, Computing and Communications

About Mirabilis Design

- System design and exploration for Systems, SoC and Software
- VisualSim- Modeling and simulation environment
- Based in Silicon Valley with offices in Southern California, Czech and India
  - Experienced application support in US, India, Japan and China
- Largest source of system modeling IP with embedded timing and power
- Experts in system modeling and architectures

Select the "Right" configuration to match customer request

Multidimensional Dataflow in Ptolemy

Chris Shaver Dai Bui Stavros Tripakis
University of California, Berkeley

Motivation

- Dataflow models processing arrays of data.
- ArrayOL formalism for multidimensional dataflow.
- Coarse and fine grain parallelization.
- SpearDE, developed by Thales.

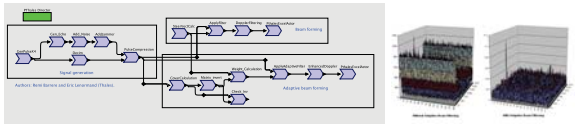


Figure: A Pthales model for adaptive beam-forming

Goals

- Develop the Pthales domain for multidimensional dataflow.
  - Use the Ptolemy platform as a basis for research in multidimensional dataflow models in the context of heterogenous system semantics, code generation, scheduling techniques, and the semantics of time.
- Implement the semantics of SpearDE and ArrayOL in Pthales.
- Formulate semantics for static and dynamic models.
- Integrate the Pthales domain with other models of computation.
- Develop code generation and hardware mapping.
- Experiment with temporal semantics in Pthales.

A Pthales Model

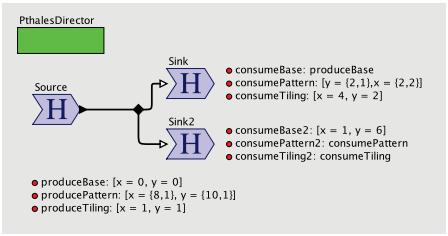


Figure: A simple Pthales model.

Pthales Port Semantics

At each port, Pthales actors produce and consume patterns of data tiled within arrays. Ports in the model are given a set of array data parameters of the following form:

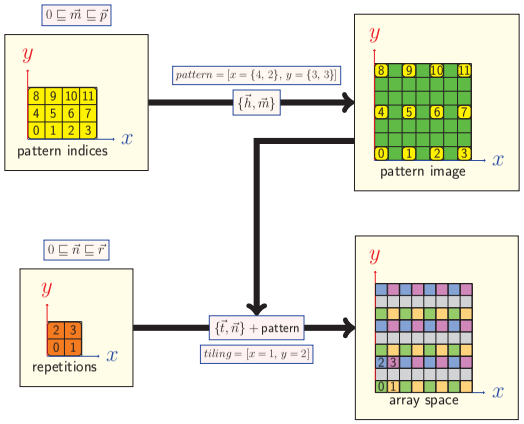
$$\textbf{parameter} = [a = x_a,\ b = x_b, \ldots]$$

where $(a, b, \ldots)$ are names and $(x_a, x_b, \ldots)$ are associated data.

Parameter	Form x_n	Meaning
base	b_n	b_n - base coordinate
pattern	$\{p_n,\ h_n\}$	p_n - pattern dimension, h_n - pattern step size
tiling	t_n	t_n - tiling step size
size	s_n	s_n - array dimension
repetitions	r_n	r_n - pattern repetitions

[if p_n is substituted for $\{p_n,\ h_n\}$, h_n defaults to 1]

Illustration



Computing Parameters

A parameter set $(\vec{b}, \vec{p}, \vec{h}, \vec{t}, \vec{s}, \vec{r})$ is overdetermined with the constraint that the repetitions must fill the array as much as possible. $\vec{s}$ can be calculated from $\vec{r}$:

$$\vec{s} \sqsupseteq \vec{b} + \{\vec{t},\ \vec{r} - \vec{1}\} + \{\vec{p},\ \vec{h} - \vec{1}\} + \vec{1}$$

where $\{\vec{a}, \vec{b}\}$ denotes point-wise multiplication.
If $\vec{s}$ is given, $\vec{r}$ can be calculated:

$$\vec{r} \sqsubseteq \{\vec{s} - \vec{b} - \vec{1} - \{\vec{p},\ \vec{h} - \vec{1}\} - \vec{t},\ \vec{t}^{-1}\}$$

the greatest $\vec{r}$ satisfying this condition being:

$$\vec{r} = \lfloor \{\vec{s} - \vec{b} - \vec{1} - \{\vec{p},\ \vec{h} - \vec{1}\} - \vec{t},\ \vec{t}^{-1}\} \rfloor$$

Dynamic Pthales

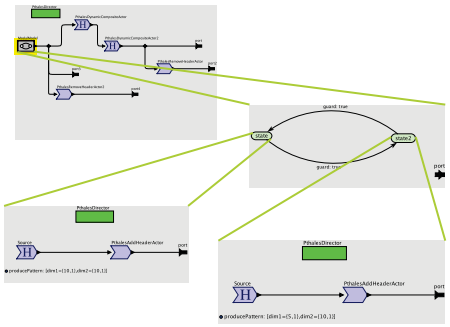


Figure: A modal Pthales model.

Modal Pthales

- Static Pthales extended with modal models.
- Parameter Dynamic Pthales allowing $\vec{s}$ to vary with data and $\vec{r}$ to be recomputed.

Parameter-Dynamic Pthales

- Parameters such as data sizes, tiling, pattern, repetitions, are dynamically changed.
- Parameter changes are propagated throughout models using header information included in data packets.

Future Work

- Embedding Pthales semantics into Process Networks.
- Expanding specification language to include expressions as parameters.
- Automatic generation of parameters from other specifications.
- Code generation for multicore platforms.
- Pipelined scheduling techniques.
- Hierarchical characterization of dynamic semantics:
  - How can different forms of dynamic multidimensional semantics be interrelated or included in one another?
  - What is the relationship between scenario-based and modal model approaches to dynamic semantics?

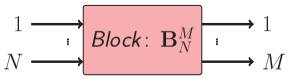
Alternative Syntaxes for Ptolemy Models

Chris Shaver
University of California, Berkeley

Block Combinator Syntaxes

- Ptolemy models can be represented in a combinator syntax rather than a point-to-point syntax with named ports.
- Models can be composed, edited, or reasoned about in such a syntax.

Blocks

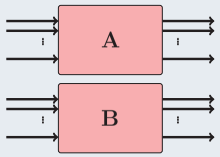


B_M^N signifies the type of a block in the combinator language with N *inputs* and M *outputs*.

Operators

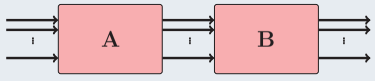
Parallel Composition: $A \otimes B$

$\otimes : \mathrm{B}_M^N \times \mathrm{B}_P^K \to \mathrm{B}_{M+P}^{N+K}$



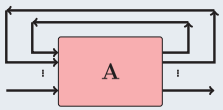
Serial Composition: $A \gg B$

$\gg : \mathrm{B}_M^N \times \mathrm{B}_N^K \to \mathrm{B}_M^K$



Feedback Contraction: $\langle A \rangle_n$

$\langle\rangle_n : \mathrm{B}_M^N \to \mathrm{B}_{M-n}^{N-n}$



Influences

- The *Faust* signal processing language.
- Diagrammatic Linear Algebra.
- Milner's Calculus of Communicating Systems.

Abstract Syntax

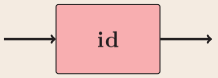
Syntax of block expression (both visual and textual):

- $E =_{syn} Name \mid E \otimes E \mid E \gg E \mid \langle E \rangle_{\mathrm{n}}$
- $Bind =_{syn} Name \leftarrow E$

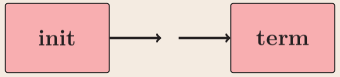
where a $Name$ can be a primitive block or bound to an E.

Primitives

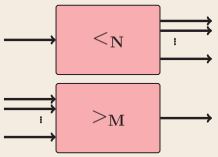
Identity: id



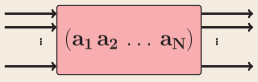
Initiator/Terminator: $init, term$



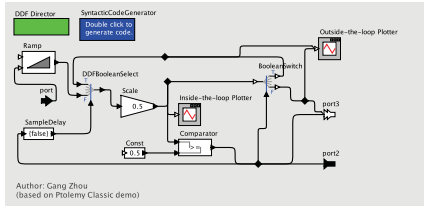
Split/Merge: $<_N, >_M$



Permutation: $(a_1\, a_2\, a_N)$



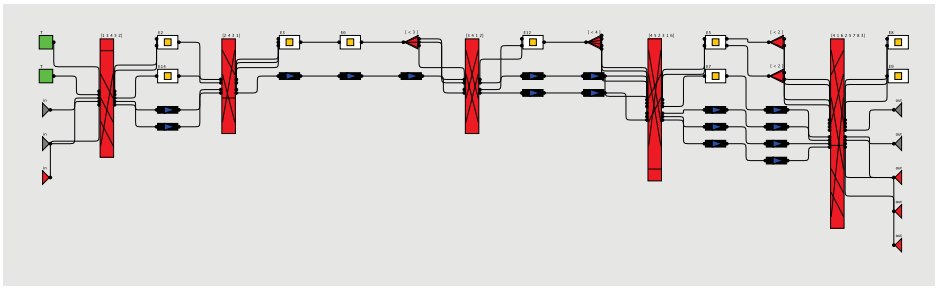
Typical Ptolemy Model



Textual Combinator Form

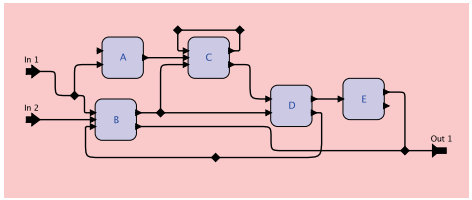
$$Expr_1 = \langle Expr_2 \rangle_2$$
$$Expr_2 = init \otimes init$$
$$\gg (13452) \gg E2 \otimes E14$$
$$\gg (2431) \gg E3 \gg E6 \gg <_3$$
$$\gg (3412) \gg E12 \gg <_4$$
$$\gg (452316) \gg E5 \otimes E7 \gg <_2 \otimes <_2$$
$$\gg (41625783) \gg E8 \otimes E9$$

Visual Combinator Form

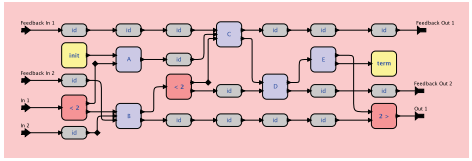


Conversion from Ptolemy to Combinator Form

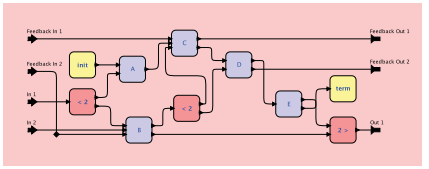
Starting with a Ptolemy model, the point-to-point syntax can be converted into a combinatorial visual syntax.



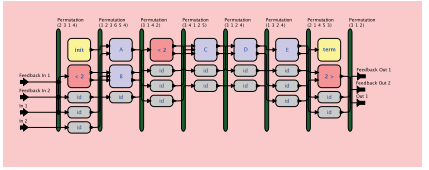
4. Organize blocks into columns dependent on predecessor columns.
5. Insert identity operators to make columns opaque.



1. Split/Merge primitives replace multiply connected relations.
2. Initiator/Terminator primitives are added to unconnected ports.
3. Feedback edges are cut and drawn out to input/output pairs.



6. Order identities to the bottom of columns.
7. Insert permutation primitives between columns.



CHESS
CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

Lattice Composition for Ontology Analysis

Elizabeth Latronico, Charles Shelton (Bosch)

Ben Lickly, Edward Lee (Berkeley)

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

125 Bosch 1886–2011

Learn more!

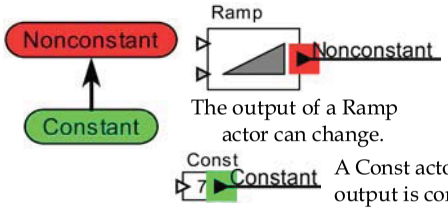
Try the ptolemy.data.ontologies package.

Publication: "Scalable Semantic Annotation using Lattice-based Ontologies", MODELS 2009

Motivation

Ontologies can be used to analyze models. A developer defines lattices plus actor constraint rules. Here are some examples:

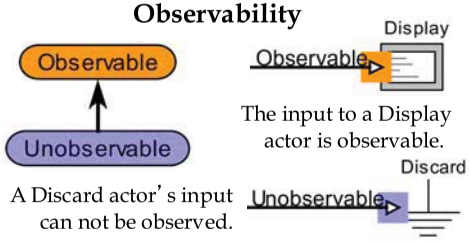
Constant / Non-constant



The output of a Ramp actor can change.

A Const actor's output is constant.

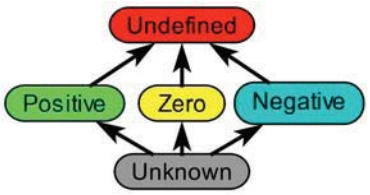
Observability

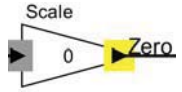


The input to a Display actor is observable.

A Discard actor's input can not be observed.

Abstract Interpretation (for scalar values)





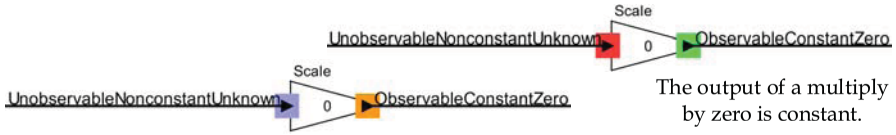
The output of a multiplication by zero is zero.

If an ontology is a lattice, the Rehof and Mogensen algorithm performs model analysis in linear time ("Tractable Constraints in Finite Semilattices", 1996). (Time is linear in the number of constraints, for finite height lattices.)

Why compose lattices?

A product lattice can be more powerful than a set of orthogonal individual lattices (Click and Cooper, "Combining Analyses, Combining Optimizations", 1995).

- The original rules from all sub-lattices are inherited.
- The developer writes a few new rules taking advantage of all sub-lattices.

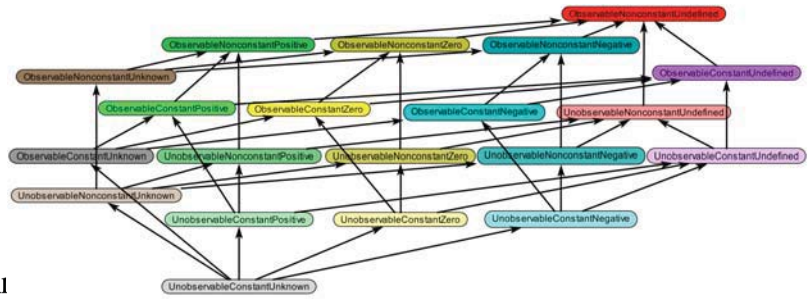


The output of a multiply by zero is constant.

The input to a multiply be zero operation is unobservable.

This approach **Supports multiple developers**, **Minimizes dependencies**, and **Gives maximum anal**

Product Lattice



Current status and objectives

The product lattice is:

- Automatically created from sub-lattices.
- Checked to ensure that the result is also a lattice.

Remaining challenges include:

- Allowing the user to edit the product lattice, especially Top and Bottom concepts.
- Some concepts in the product lattice don't make sense. Allow users to note these with acceptance criteria.
- Combining finite and infinite width lattices.
- Combining inherited and new rules and ensuring monotonicity.

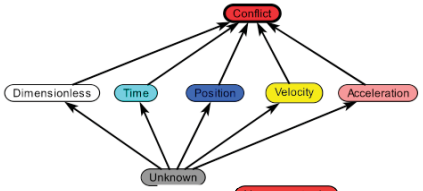
Infinite Lattice Patterns for Ontology Analysis

Ben Lickly, Charles Shelton, Elizabeth Latronico, Edward A. Lee

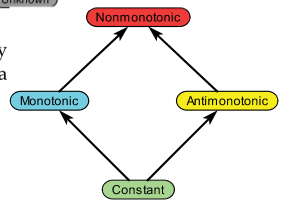
Backgroud

Using ontologies allows us to statically analyze Ptolemy models in a principled way. The first step to doing so is to represent the domain of interest as a lattice-based ontology.

Example Ontologies.



By formally expressing a body of knowledge as a lattice-based ontology, we can encode domain knowledge to inform analysis.



Infinite Ontologies

While many types of knowledge can be encoded in finite ontologies, infinite ontologies can express a larger class of properties. We have found two patterns of infinite ontoligies to be broadly useful:

•**Infinite Flat Lattices**: Useful for including value information into the ontologies.

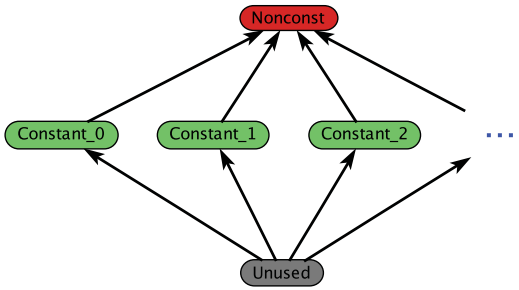
•**Recursive Lattices**: Useful for including structured information corresponding to structured data types.

Acknowledgements

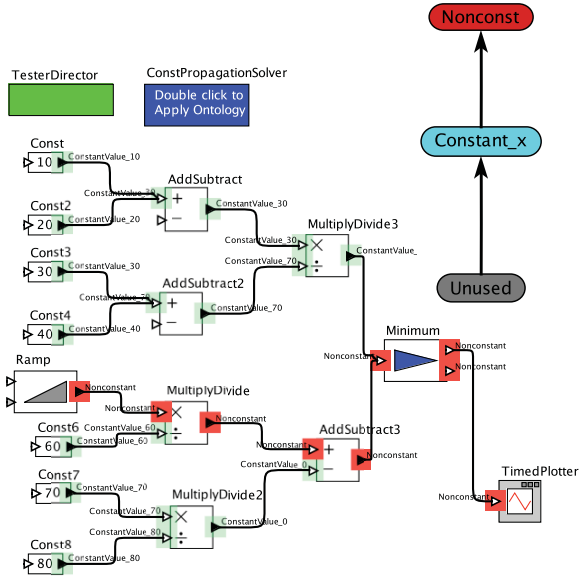
This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES), and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

Infinite Flat Lattice Pattern

In the infinite flat lattice pattern, we allow lattices that are infinite in one dimension at specific points in the lattice. This is most often useful as a way of parameterizing a concept by a value in order to make something resembling a dependent type system in which values are incorporated into types.



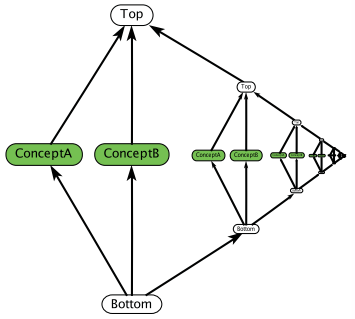
Example: Constant Propagation

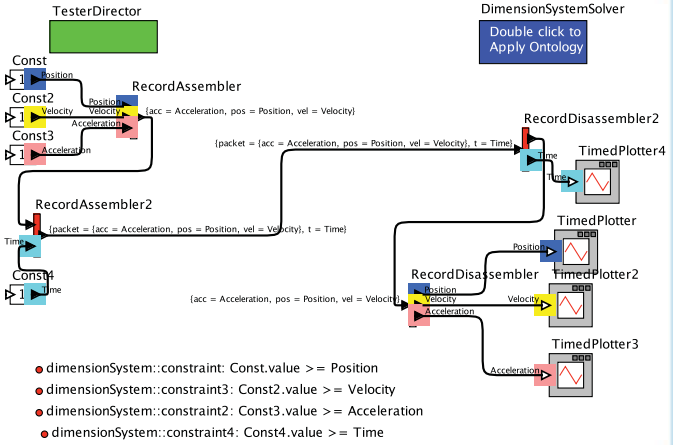


Infinite Recursive Lattice Pattern

Recursive lattices may be contained within a concept in the lattice. In the lattice to the right, for example, the rightmost concept is isomorphic to the entire lattice.

A classic example of a recursive lattice is a type lattice that contains array or record types. Since arrays (or records) may recursively contain arrays or records within them, this creates an infinite lattice of potential types.

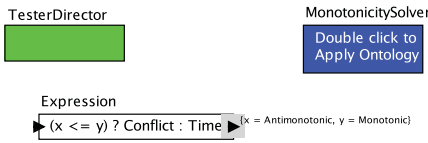




Example: Monotonicity Analysis

In addition to dataflow-based graphs, Ptolemy expressions can also be analyzed with our solver. This example shows an analysis that determines whether expressions are monotonic, antimonotonic, constant, or not.

Monotonic:
$x \le y \Rightarrow f(x) \le f(y)$
Antimonotonic:
$x \le y \Rightarrow f(x) \ge f(y)$
Constant:
$f(x) = f(y)$



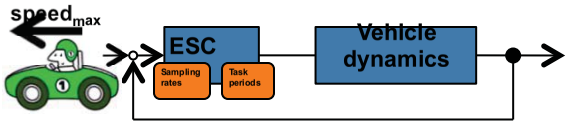
February 16, 2011
Berkeley, CA

Ptolemy Miniconference

Towards flexible and robust cyber-physical systems through self-organization

Systems run in worst case mode determined at design time

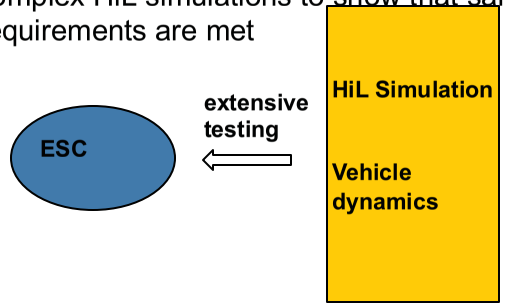
Timing constraints are given by the worst case scenario



Sensor sampling rates and ESC control task periods run at a speed to avoid oversteering of the car at max. speed

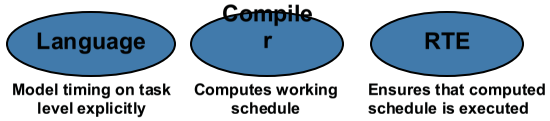
At best, there are different modes of operation available

System is given no information to react in an aware manner => Timing is tested in complex HiL simulations to show that safety requirements are met



Reasoning about timing as the key

Giotto language allows to reason about timing explicitly



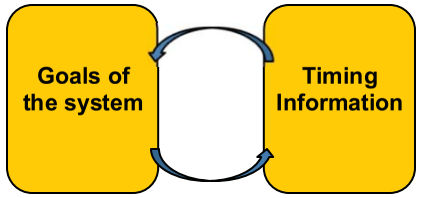
However, once the schedule is computed, it is static and not ought to be changed during run-time

From the timing properties, such as task periods and deadlines, used for describing the timing behavior of the system it is not possible to derive the reasons for the timing constraints, the information is missing

Recent results from Prof. Tabuada have shown that control tasks show stable behavior while **self-triggering**

This is especially interesting because timing is derived from the physics of the system => there is a meaning, it is not just a property of the worst case analysis

To be really able to adapt timing to changing conditions, timing information must be coupled to the goals of the system



In the ESC example, sensor sampling rates and task periods could be adjusted to the current speed of the car

Goal of this ongoing work: Equip run-time models with needed information to adapt their timing behavior at run-time

In the case of control applications, stability must be guaranteed when changing timing settings.

Andreas Thuy
University of Paderborn/C-LAB
athuy@c-lab.de

c-lab
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

The Earlier the Better: A Theory of Timed Actor Interfaces

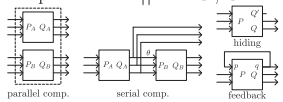
Marc Geilen — TU Eindhoven
Stavros Tripakis — UC Berkeley
Maarten Wiggers — University of Twente

1. Component-based design

Questions:
What are the right building blocks?
Which ones to use? How to connect them?
What is a component? How to reason about components?

2. Composition

- Components: A, B, ...
- Composition operators: $A||B = C$, $C \circ D = E$, ...



- **Substitutability**: can I replace C by F in E?

3. Abstraction-Refinement

Two faces of the same golden coin!
$B \sqsubseteq A$:

- A is an abstraction of B
- B is a refinement of A

4. Compositionality

- If $B \sqsubseteq A$ and $B' \sqsubseteq A'$ then $(B||B') \sqsubseteq (A||A')$, and $(B \circ B') \sqsubseteq (A \circ A')$, and ...
- If $B \sqsubseteq A$ then B preserves interesting properties of A

Many existing frameworks, e.g., interface theories [dAH01]. But:

- focus on functional/correctness models and properties;
- not suitable for cyber-physical systems: timing, performance.

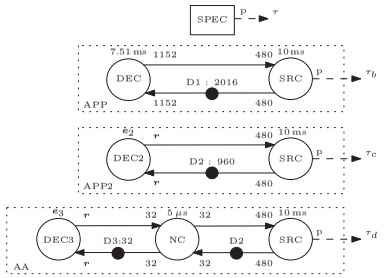
5. This work

A compositional theory for timing and performance properties.
Deterministic abstractions for non-deterministic systems!

E.g., MP3 player:

- Deterministic specification: throughput = 44.1kHz, latency = 50 ms.
- Performance preserved by stepwise refinement.

6. Example: MP3 player



7. Actors

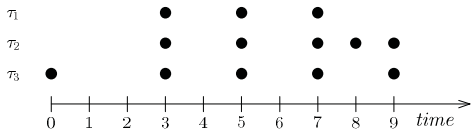
Formal descriptions of components:

$$A = (P, Q, R_A)$$

P: set of input ports.
Q: set of output ports.
$R_A \subseteq Tr(P) \times Tr(Q)$: relation between input and output event traces.

8. Event traces

Event trace: port-wise vector of event sequences.
Event sequence: sequence of "pure" (non-valued) events in time.

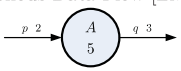


9. Examples of actors

Variable-delay actor $\Delta_{[d_1,d_2]}$:

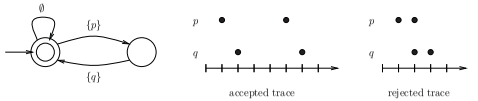
$$x\Delta_{[d_1,d_2]}y \text{ iff } |x(p)| = |y(q)| \wedge \forall n < |x(p)| : x(p)(n) + d_1 \leq y(q)(n) \leq x(p)(n) + d_2 \wedge (n > 0 \implies y(q)(n) \geq y(q)(n-1)).$$

SDF (Synchronous Data Flow [LM87]) actor:



$$xAy \text{ iff } |y(q)| = 3 \cdot \min_{p \in P} (|x(p)| \div 2) \wedge \forall n < |y(q)| : y(q)(n) = 5 + x(p)((n \div 3 + 1) \cdot 2 - 1)$$

Finite-automaton actor:

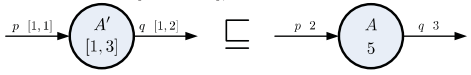


10. The earlier-the-better refinement

$B = (P, Q, R_B)$ refines $A = (P, Q, R_A)$, denoted $B \sqsubseteq A$, iff
(1) $\text{in}_A \subseteq \text{in}_B$ (legal inputs of A are also legal in B);
(2) $\forall x \in \text{in}_A, \forall y : xBy \implies \exists y' : y \sqsubseteq y' \wedge xAy'$ (outputs of B come no later than those of A).

Where $y \sqsubseteq y'$ if events in y happen no later than those in y'. E.g., $\tau_3 \sqsubseteq \tau_2$, $\tau_2 \sqsubseteq \tau_1$, but $\tau_1 \not\sqsubseteq \tau_2$.

Example: CSDF (cyclo-static dataflow [BELP96]) actor refining SDF actor:



11. Results

- Refinement is compositional w.r.t. parallel, serial and feedback composition (under some conditions).
- Refinement preserves worst-case throughput and latency.
- Algorithms to check refinement and compute compositions for various finite representations (SDF, automata, ...).
- Semantical unification of existing frameworks (dataflow, automata, service curves, ...).

12. References

[BELP96] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstmete. Cyclo-static dataflow. *IEEE Tran. on Signal Processing*, 44(2), 1996.

[dAH01] L. de Alfaro and T. Henzinger. Interface theories for component-based design. In *EMSOFT'01*. Springer, LNCS 2211, 2001.

[GTW10] Marc Geilen, Stavros Tripakis, and Maarten Wiggers. The earlier the better: A theory of timed actor interfaces. Technical Report UCB/EECS-2010-130, EECS Department, University of California, Berkeley, Oct 2010. Shorter version to appear in 14th Intl. Conf. Hybrid Systems: Computation and Control (HSCC'11).

[LM87] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

February 16, 2011
Berkeley, CA

Ptolemy Miniconference

Automated Fixed-Point Analysis in Ptolemy

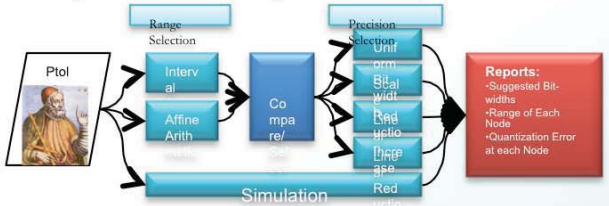
Derrick Gibelyou
Michael Wirthlin

Motivation

- Using a uniform bit-width for FPGAs is inefficient
  - Uniform bit-width selection is useful for DSP and other fixed-width systems
  - Doesn't take advantage of the flexibility of FPGAs
  - Finding the optimal bit-width reduces area and increases clock rate while maintaining the quality of the answer
- Finding the optimal bit-width is difficult
- Many techniques exist for finding near-optimal bit-widths:
  - Statistical Simulation
  - Feed-forward Heuristics
  - SAT / ILP solver
- Ptolemy provides a good infrastructure on which to implement these algorithms

New Ptolemy Director

- Based on SDF director
- Ends when all strategies are finished
  - Strategies are like sub-directors
- Director runs strategies to find range, then runs strategies to find precision
- Prints a report when all strategies have completed

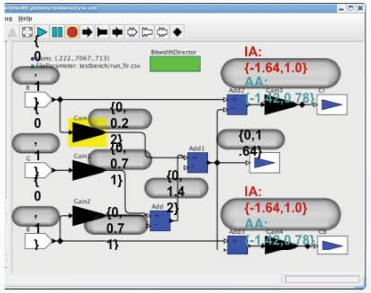


New Tokens

- Implemented new Range Tokens and Simulation Tokens
- Range Tokens inherit from ScalarToken
  - Allows math with ranges and constants
  - Although represented by more than one number, a range token should be treated as a scalar
- Error Tokens
  - Holds two range tokens:
    - Dynamic range
    - Range of Quantization Error
  - Still represents a scalar entity

Range Algorithms

- Interval Arithmetic
  - Simple method for calculating range:
  - X=[-1,1], Y=[-2,2]; X + Y= [-3,3]
  - Cannot be used in feedback systems
- Affine Arithmetic:
  - Accounts for correlations between the inputs, e.g.
    - $X \in [0..1]$
    - Interval Arithmetic $X - X \in [-1..1]$
    - Affine Arithmetic $X - X \in [0]$
  - Can be used to solve for the range of IIR filters (feedback systems)



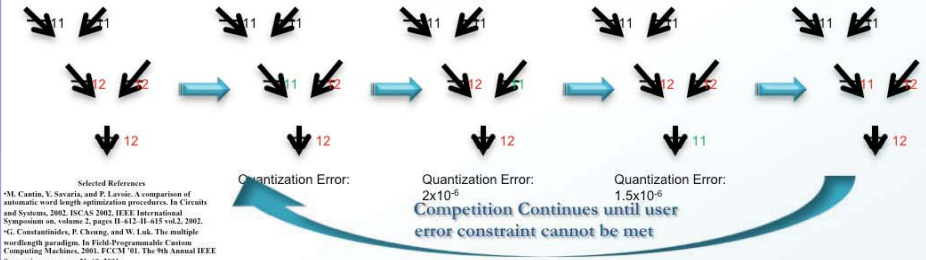
Selected References
•R. Moore. Interval analysis. Englewood Cliffs, 1966.
•G. A. Constantinides, Word-length optimization for differentiable nonlinear systems. ACM Trans. Des. Autom. Electron. Syst., 11(1):26–43, 2006
•C. F. Fang, R. A. Rutenbar, and T. Chen. Fast, accurate static analysis for Fixed-Point Finite-Precision effects in DSP designs. In Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design, page 275. IEEE Computer Society, 2003.

Precision Algorithms

- Most published techniques involve heuristic competitions to find a near optimal bit-width
- Competition:
  - Once the range is found, the system error can be calculated
  - The winning operator in each iteration is the one that that both:
  - increases the error the least
  - decreases the area the most
  - Competition continues until user constraint can no longer be met

Competition Pseudo Code

```
While(system_error_constraint is met)
   foreach (operator)
      reduce operator width by 1 bit
      measure system error
      score = error * cost_function(operator)
      Save score
      restore operator width
   (Operator with min score).width = width-1
```



Selected References
•M. Cantin, Y. Savaria, and P. Lavoie. A comparison of automatic word length optimization procedures. In Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, volume 2, pages II–612–II–615 vol.2, 2002.
•G. Constantinides, P. Cheung, and W. Luk. The multiple wordlength paradigm. In Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on, pages 51–60, 2001.
•W. Osborne, R. Cheung, J. Coutinho, W. Luk, and O. Mencer. Automatic accuracy-guaranteed bit-width optimization for fixed and floating-point systems. In Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on, pages 617–620, 2007.

Results

- Several simple test benches have been created:
  - FIR and IIR filters
  - DCT
  - RGB to YUV converter
  - BPSK timing loop
- Results for BPSK timing loop closely match those of human selected values:

Circuit Portion	Operator	Hand Optimized	Program Results	Difference
farrow	Product1	10	6	4
farrow	Product2	10	5	5
farrow	Sum	10	6	4
farrow	Sum10	10	13	-3
farrow	Gain1	10	13	-3
farrow	Sum7	10	4	6
farrow	Sum5	10	6	4
farrow	Sum6	10	8	-4
farrow	Sum3	10	8	4
farrow	Sum4	10	8	4
farrow	Sum1	10	7	3
farrow	Sum2	10	6	4
Loop Constant	Loop filter constant	11	1	10
Loop Filter	First Order Loop Filter	11	11	0
NCO	mod	11	12	-1
NCO	Gain	9	14	-5
NCO	Sum9	11	12	-1
NCO	Sum8	11	11	0
ZCTED	Product3	10	14	-4
ZCTED	Subtract1	10	14	-4

Work funded by CHREC
NSF Center for High-Performance Reconfigurable Computing

February 16, 2011
Berkeley, CA

Ptolemy Miniconference

IEEE 1588 Time Synchronization for Real-Time Distributed Systems

Michael Zimmer, Edward A. Lee, University of California, Berkeley

Introduction

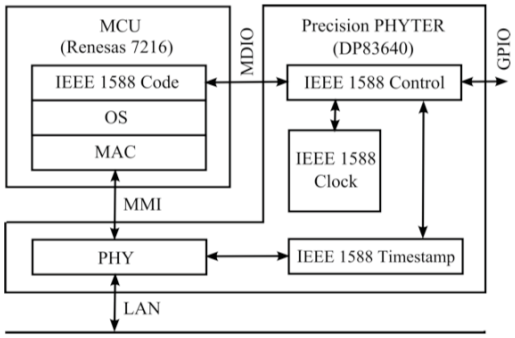
Many real-time distributed systems require some shared notion of time. When measuring or acting on the physical world, the relative times of these actions between devices in a distributed system can be of significant importance. The accuracy required for this time synchronization is application specific, and can range anywhere from seconds to nanoseconds. Some areas which can utilize accurate time synchronization are test and measurement, industrial automation and control, and power generation.

The IEEE 1588-2008 standard defines a Precision Time Protocol (PTP) to synchronize time in a distributed system over a network to the sub-microsecond range. The protocol was developed to bridge the gap between the existing common methods of time synchronization, where Network Time Protocol (NTP) may not be accurate enough and Global Positioning System (GPS) may be too expensive.

IEEE 1588 Basics

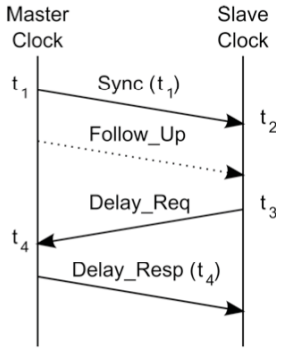
Architecture

The protocol operates on a hierarchical master-slave distribution where slave clocks synchronize their time to master clocks. A best master clock (BMC) algorithm is used to select master clocks. Basically, all clocks in a network exchange information about their stability and accuracy, and the best clock is elected as the master.



Event Messages

Timestamped event messages are used to synchronize the slave clock to the master clock. The event message sequence as shown below typically occurs every second, with the messages being send as User Datagram Protocol (UDP) packets over Ethernet (although other configurations are possible).



Mean Path Delay:

$$t_d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2}$$

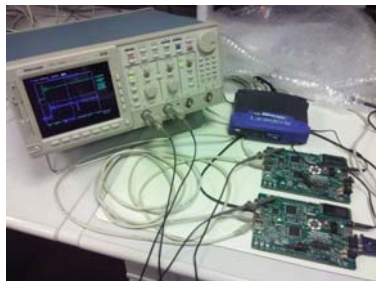
Offset From Master:

$$t_o = (t_2 - t_1) - t_d$$

Implementation

Hardware

The implementation is being developed on a Renesas 7216 Demonstration Kit with a 32-bit RISC SuperH CPU. The DP83640 Precision PHYTER from National Semiconductor is used to provide hardware support for IEEE 1588 in the form of an integrated 125MHz IEEE 1588 clock, packet detection and timestamping with 8ns precision, and synchronized event timestamping and triggering through GPIO ports.

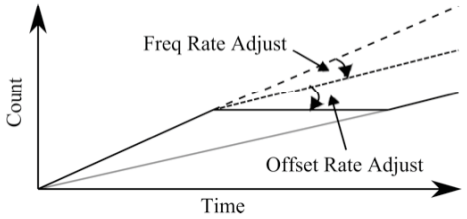


Discrete Event Semantics

The protocol itself is implemented as an actor with discrete event semantics, and is designed for use within the PTIDES (Programming Temporally Integrated Distributed Embedded Systems) programming model. The code is only executed when provided with an input event, and executes to completion without blocking. In order to provide timer functionality, the actor can produce an event to be provided to itself as input at a time in the future.

PTP Servo Algorithm

A clock servo algorithm is used to adjust the frequency of the slave clock. The goal is to match the frequency of the slave clock with that of the master clock to maintain a zero offset between the two clocks. It is important to point out that although the clock value can also be set or stepped, this is to be avoided since it can make past events appear as having occurred in the future.



Results

The basics of the protocol are implemented, and two directly connected boards are able to achieve synchronization within 25ns. Once connected through a router which doesn't support IEEE 1588, time synchronization is only accurate to around 100ns. The implementation was also successfully tested in a network with another implementation of the protocol. Future work involves fully supporting the protocol and tuning the servo algorithm.

Acknowledgments

The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. The authors would also like to thank Slobodan Matic and John Eidson for their guidance.

CHESS

CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

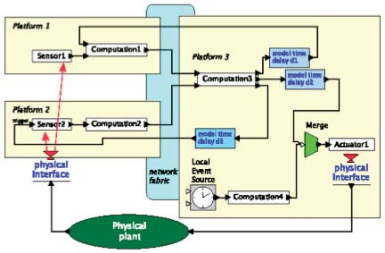
From PTIDES to PtidyOS: Programming Distributed Real-Time Embedded Systems

John Eidson
Jeff C. Jensen
Edward A. Lee
Slobodan Matic
Jia Zou

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

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES), and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

Distributed Real-Time System Implementation



Programming Model

Programming **T**emporally **I**ntegrated **D**istributed **E**mbedded **S**ystems

Based on Discrete-Event semantics

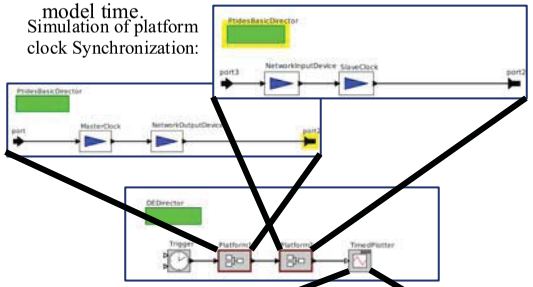
Event processing in time-stamp order

Clock Synchronization Simulation

Notions of Time:

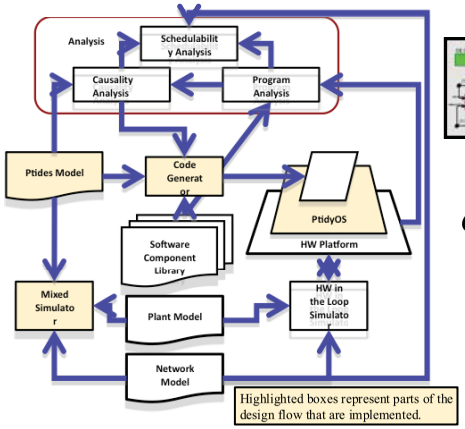
- Top level simulate real world (oracle) physical time.
- Each platform contains a clock to keep track its notion of platform physical time.
- Each Ptides director has a notion of model time.

Simulation of platform clock Synchronization:

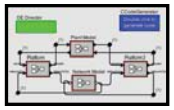


Slave's platform clock tracks master's platform clock, as shown on the output graph:

Ptides Design Flow



Highlighted boxes represent parts of the design flow that are implemented.



Code Generation

Ptides Simulator

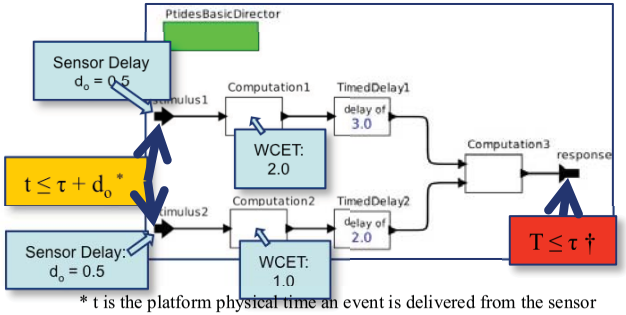
Simulate functional and real-time properties.

Function simulation :

- Same denotational semantics as DE simulation, i.e., Same functional outputs whether Ptides or DE Director is used.
- Performs safe-to-process analysis instead of DE simulation
- Currently support 2 Ptides execution strategies: Basic, PreemptiveEDF (combines Ptides with Earliest-Deadline-First).

Real-time simulation:

- Users annotate WCET and sensor delays.
- Check for deadline misses at actuators.



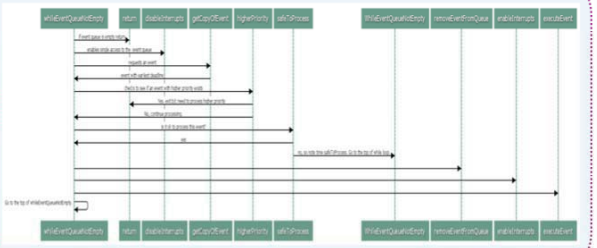
* t is the platform physical time an event is delivered from the sensor to the rest of the platform. τ is the timestamp of the sensor event.

† t is the platform physical time an actuation event is delivered to the actuator. τ is the timestamp of the actuation event.

PtidyOS

- Is a library linked against application C code.
- Implements Ptides semantics
- Local execution strategy includes a safe to process analysis and resource scheduling layer.
- Uses a Single Stack
- Steps away from threading model
- To ensure event of highest priority is always processed first, interrupts play an important role:
  - Event processing is done within Interrupt service routines
  - Reentrant interrupts
- No dynamic memory allocation.

Event Processing Algorithm Sequence Diagram

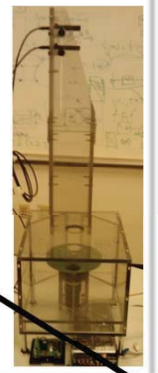


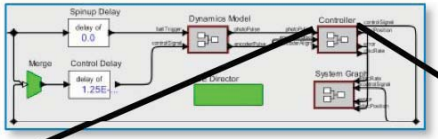
***The use of a single stack and interrupts for mutual exclusion was done in TinyOS. PtidyOS uses these concepts and introduces time semantics.**

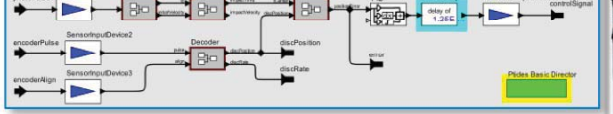
Application: The Tunneling Ball Device

- Cyber-physical system with real-time constraints
- Event-driven signals (motor encoding, object detection)
- Parallels an automotive engine control unit (ECU)
- Timed semantics ensure:
  - Deterministic control delay
  - Data associated with the same physical time are processed together

- Balls drop above a rapidly spinning disc
- Device must adjust disc to allow ball to pass through a small hole, without stopping disc







A. JAMES CLARK SCHOOL OF ENGINEERING

The Dataflow Interchange Format: Towards Co-design of DSP-oriented Dataflow Models and Transformations

Shuvra S. Bhattacharyya

Maryland DSPCAD Research Group
http://www.ece.umd.edu/DSPCAD/home/dspcad.htm

Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies
University of Maryland, College Park, 20742, USA.

Ptolemy Miniconference, University of California, Berkeley,
Feb. 16, 2011 [Version: Feb. 13, 2011]

UMIACS University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

A. JAMES CLARK SCHOOL OF ENGINEERING

Outline

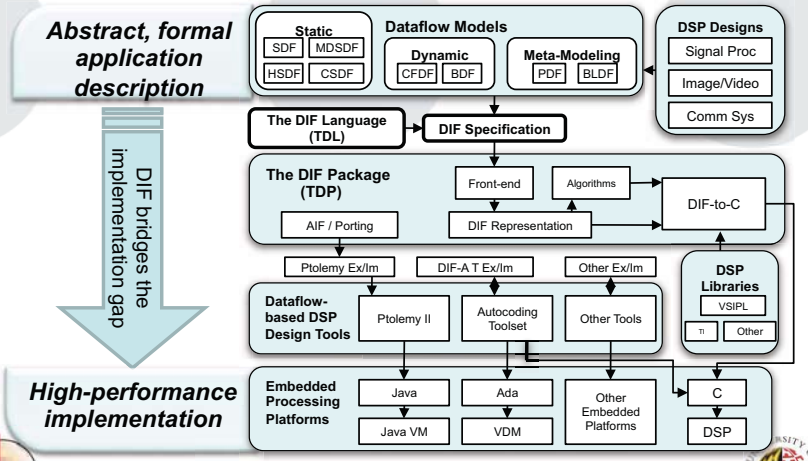
- → Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
- Application case study: high energy physics
- Wrapup

UMIACS University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

The Dataflow Interchange Format (DIF)

- DIF captures coarse grain dataflow applications formally [4]
- To formally describe applications, the DIF Language (TDL) is
  - Designed to capture a variety of dataflow models
  - Can be used in conjunction with functionally simulatable actor descriptions
- To facilitate design, the DIF Package (TDP) provides:
  - Scheduler, simulator, analyzers



Other benefits to beginning with a formal description:

- Bounded memory and deadlock detection
- Buffer and communication minimization:
- Parallel, Multirate loop, or Quasi-static scheduling
- Heterogeneous task mapping and co-synthesis
- Probabilistic design, Data partitioning, Vectorization,

The DIF Language: Sketch

```
[dataflowModel] graphID {
    basedon {
        graphID;
    }

    [topology] {
        nodes = ndID, ...;
        edges = edgeID(srcNdID, snkNdID), ...;
    }

    [builtInAttr] {
        elementID = value;
        elementID = id;
        elementID = id1, id2, ...;
    }

    [attribute] usrDefAttr {
        elementID = value;
        elementID = id;
        elementID = id1, id2, ...;
    }
    [refinement] {
        ...
    }
}
```

Evolution of Dataflow Models of Computation for DSP: Examples

- Computation Graphs and Marked Graphs [Karp 1966, Reiter 1968]
- Kahn process networks [Kahn 1974]
- Synchronous dataflow, [Lee 1987]
  - Static multirate behavior
  - SPW (Cadence) , National Instruments LabVIEW, and others.
- Well behaved stream flow graphs [1992]
  - Schemas for bounded dynamics
- Boolean/integer dataflow [Buck 1994]
  - Turing complete models
- Multidimensional synchronous dataflow [Lee 1992]
  - Image and video processing
- Scalable synchronous dataflow [Ritz 1993]
  - Block processing
  - COSSAP (Synopsys)
- CAL [Eker 2003]
  - Actor-based dataflow language
- Cyclo-static dataflow [Bilsen 1996]
  - Phased behavior
  - Eonic Virtuoso Synchro, Synopsys El Greco and Cocentric,
    Angeles System Canvas
- Bounded dynamic dataflow
  - Bounded dynamic data transfer [Pankert 1994]
- The processing graph method [Stevens, 1997]
  - Reconfigurable dynamic dataflow
  - U. S. Naval Research Lab, MCCI Autocoding Toolset
- Stream-based functions [Kienhuis 2001]
- Parameterized dataflow [Bhattacharya 2001]
  - Reconfigurable static dataflow
  - Meta-modeling for more general dataflow graph reconfiguration
- Reactive process networks [Geilen 2004]
- Blocked dataflow [Ko 2005]
  - Image and video through parameterized processing
- Windowed synchronous dataflow [Keinert 2006]
- Parameterized stream-based functions [Nikolov 2008]
- Enable-invoke dataflow [Plishker 2008]
- Variable rate dataflow [Wiggers 2008]

A. JAMES CLARK SCHOOL OF ENGINEERING

DIF Project Components

- Core components
  - The DIF language (TDL)
  - The DIF package (TDP)
  - Enable-invoke dataflow (EIDF) and functional DIF
  - DIFML: XML dialect
- Plug-ins
  - DIF-to-C: Software synthesis for SDF
  - TDIF and TDIFSyn
  - The dataflow schedule graph (DSG)
- Interfaces to ADS, OpenDF, LabVIEW, Ptolemy II, …

UMIACS University of Maryland Institute for Advanced Computer Studies
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

High Level Dataflow Transformations

- A well designed dataflow representation exposes opportunities for high level algorithm and architecture transformations.
- High level of abstraction → high implementation impact
- Dataflow representation is suitable both for behavior-level modeling, structural modeling, and mixed behavior-structure modeling
  - Transformations can be applied to all three types of representations to focus subsequent steps of the design flow on more favorable solutions
- Complementary to advances in
  - C compiler technology (intra-actor functionality)
  - Object oriented methods (library management, application service management)
  - HDL synthesis (intra-actor functionality)

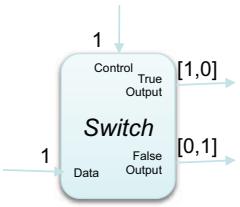
Representative Dataflow Analyses and Optimizations

- Bounded memory and deadlock detection: consistency
- Buffer minimization: minimize communication cost
- Multirate loop scheduling: optimize code/data trade-off
- Parallel scheduling and pipeline configuration
- Heterogeneous task mapping and co-synthesis
- Quasi-static scheduling: minimize run-time overhead
- Probabilistic design: adapt system resources and exploit slack
- Data partitioning: exploit parallel data memories
- Vectorization: improve context switching, pipelining
- Synchronization optimization: self-timed implementation
- Clustering of actors into atomic scheduling units

Formal Model Detection (Core Functional Dataflow [3])

- Divide actors into a set of modes
  - Each mode has a fixed consumption and production behavior
- Write the enabling conditions for each mode
- Write the computation associated with each mode
  - Including next mode to enable and then invoke
- For example, consider a standard Switch:

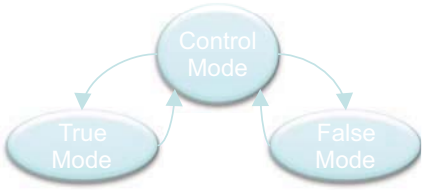
Switch Actor



Production & consumption behavior of switch modes

Mode	Consumes		Produces	
	Control	Data	True	False
Control	1	0	0	0
True	0	1	1	0
False	0	1	0	1

Mode transition diagram between switch modes

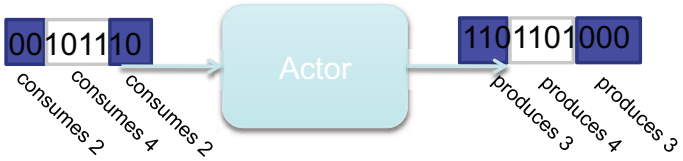


Practical Model Detection on Units

- Deterministic – Does the output repeat?

- Statefulness – Does the output just reorder?

- Dataflow model – Does input & output behavior repeat?



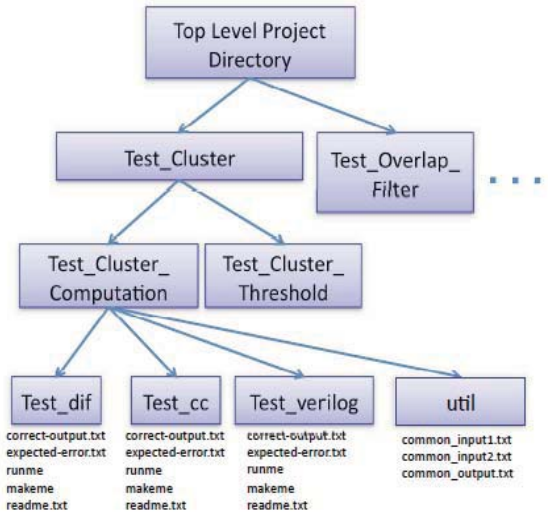
DICE: DSPCAD Integrative Command-Line Environment [2]

What it is…

- a framework for managing cross-platform testing
- language independent
- an open source resource

*What it does **not** do*

- provide code synthesis or debugging tools
- provide simulation capabilities
- transcode between platforms or languages

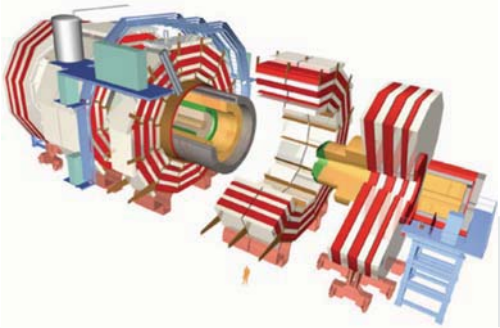


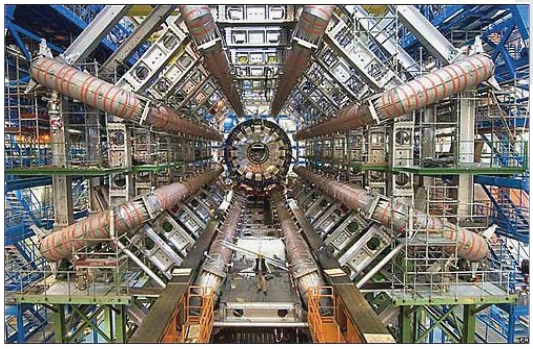
Outline

- Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
- → Application case study: high energy physics
- Wrapup

Case Study: Compact Muon Solenoid Trigger

- Complex:
  - 9300 magnets
  - Protons travel at 99.99% times the speed of light
  - 7 TeV beam collisions
- Performance Oriented:
  - 6 collision detectors
  - 600 million proton collisions per second
- International Collaboration:
  - 2000 Scientists
  - 155 Institutes
  - 37 Countries





A. JAMES CLARK
SCHOOL OF ENGINEERING

CMS Trigger Background

- Large Hadron Collider (LHC)
  - CERN: Switzlerland/France
  - Event rate of 1GHz
  - Trigger Selectivity: ratio of trigger rate to event rate (e.g., 10^{-11})
- Compact Muon Solenoid
  - General purpose particle physics detector for the LHC
  - CMS Trigger: Multi-Level Filtering: Level 1 (FPGA) → High Level Trigger (software) → Tape storage

Source: http://en.wikipedia.org/wiki/Trigger_%28particle_physics%29, **and** http://en.wikipedia.org/wiki/Compact_Muon_Solenoid#Layer_2_.E2.80.93_The_Electromagnetic_Calorimeter, **Dec. 1, 2010.**

UMIACS
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

A. JAMES CLARK SCHOOL OF ENGINEERING

Goals: Efficient, Agile Design

- The upgraded Calorimeter Trigger will require new algorithms
- Modern field programmable gate arrays (FPGAs) provide efficient platforms
- Implement Calorimeter Trigger using
  - A unified design platform
  - Unified design and test methodologies
  - Techniques that facilitate future upgrades
- Start by implementing a baseline design for the new algorithms

UMIACS University of Maryland Institute for Advanced Computer Studies

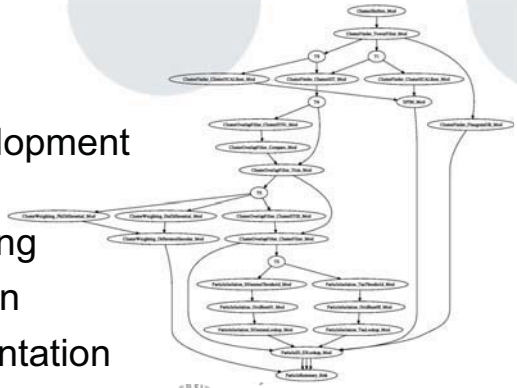
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

A. JAMES CLARK SCHOOL OF ENGINEERING

Solution: Novel Implementations and a Unified Cross-Platform Management System

- Collaboration with University of Wisconsin [1]
- Novel FPGA designs
  - Reexamination of physics algorithms for FPGAs
  - Structured analysis of resource usage
- Cross-platform design management
  - Novel, light weight development framework
  - Cross-platform unit testing
  - Dataflow model detection
  - Enhanced auto-documentation

Automatically generated application graph



UMIACS University of Maryland Institute for Advanced Computer Studies

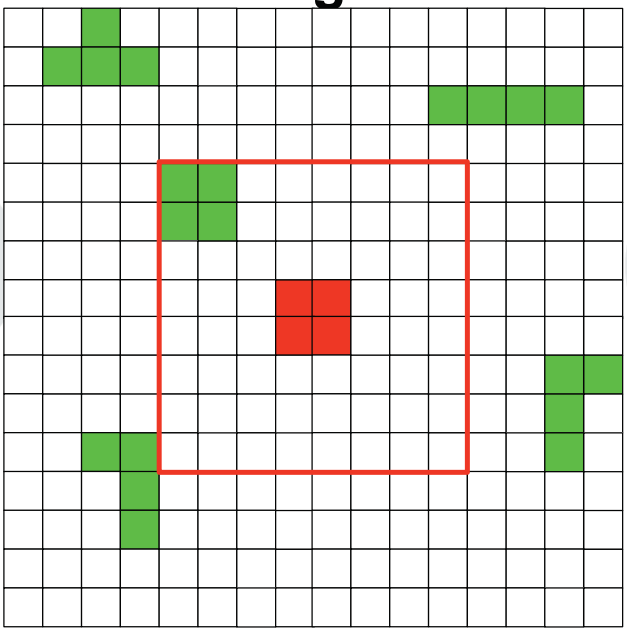
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Impact: Performance and Cost

- Novel FPGA implementations for over a dozen modules in the CMS detector
  - Improve performance
  - Cut implementation costs by reducing the number of FPGAs required for the upgrade
- New design process
  - Bugs found earlier in design process saves time and money
  - Automated documentation facilitates fast collaborative design process

A. JAMES CLARK
SCHOOL OF ENGINEERING

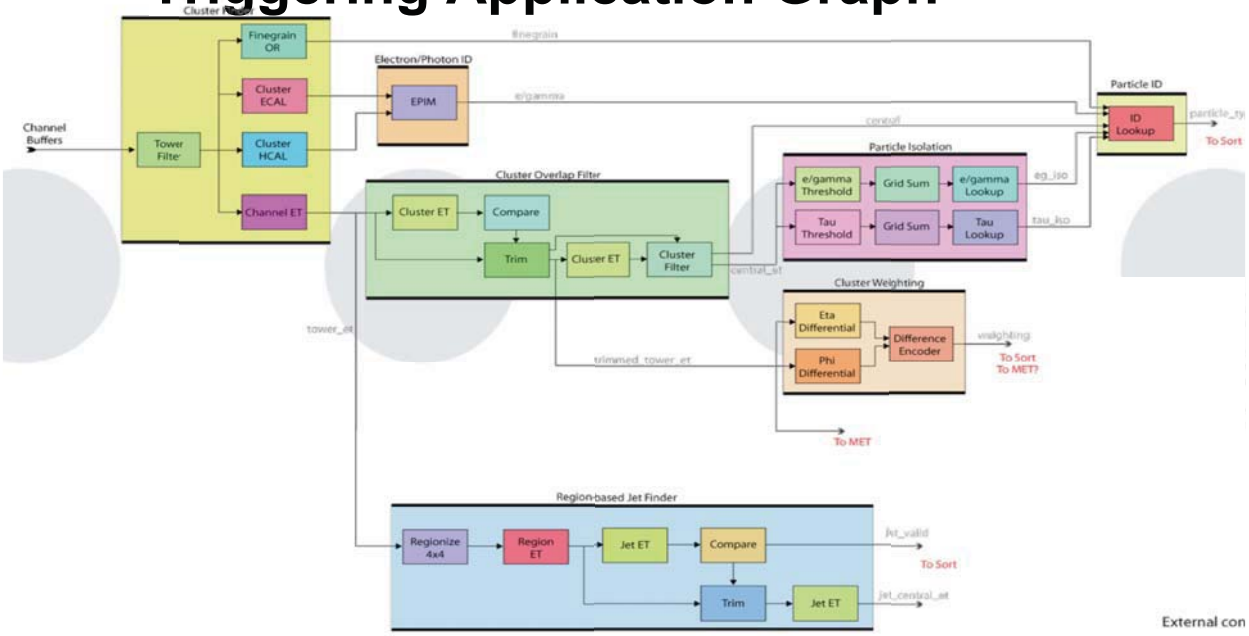
Processing Detectors



- 56x72 sized grids
- With millions of events a second, storing all of the data would result in GigaBytes per second
- Instead, store **only** events that trigger certain conditions
- L1 trigger finds image features that represent certain particles from a series of:
  - Thresholding
  - Filtering
  - Sorting
- Must complete in nanoseconds to process every sample period

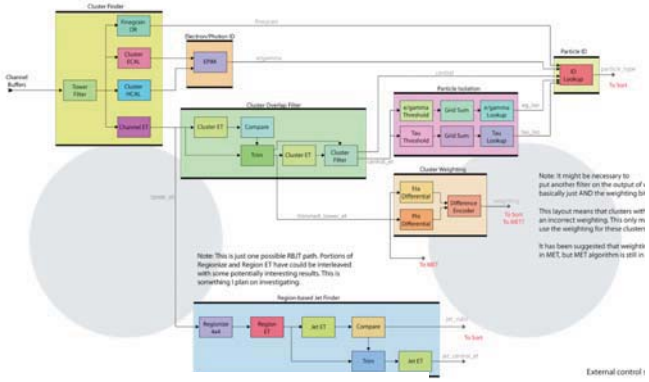
UMIACS
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Triggering Application Graph



A. JAMES CLARK
SCHOOL OF ENGINEERING

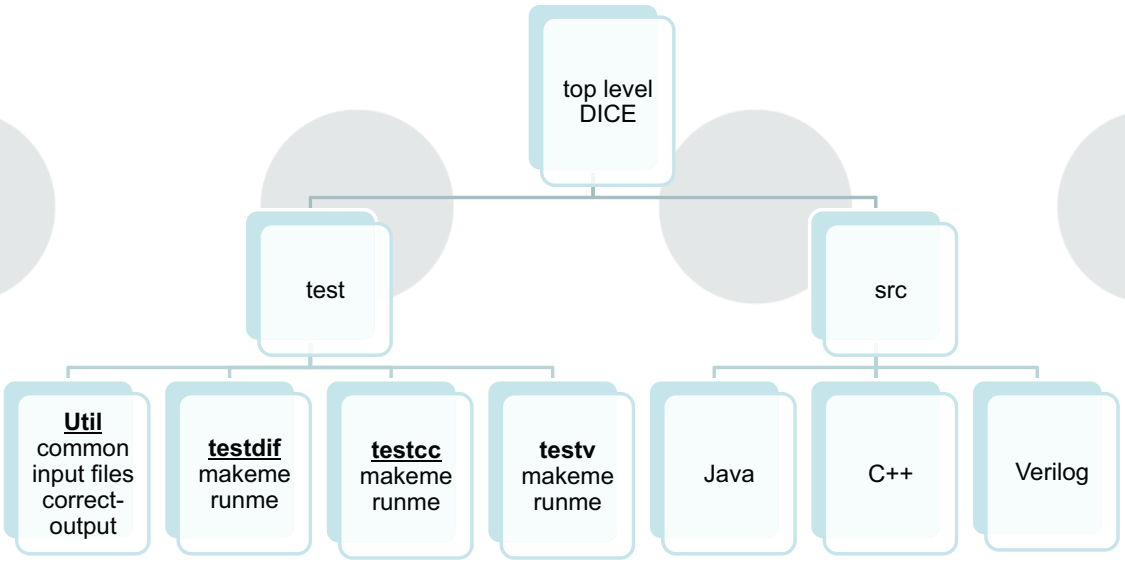
Triggering Application Graph



Written by application designers and then re-implemented by hardware engineers → Cross-platform verification is a problem

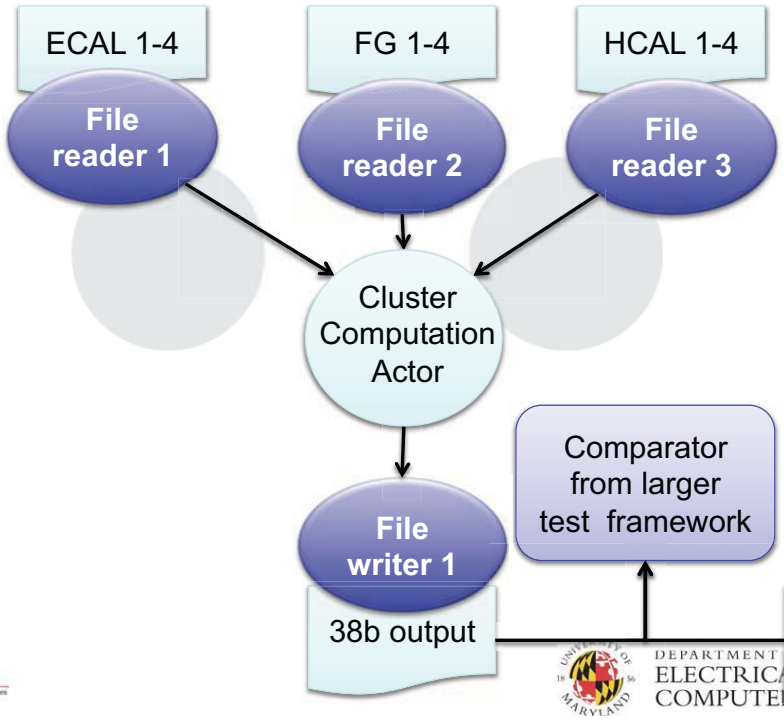
UMIACS
University of Maryland Institute for Advanced Computer Studies
DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

Test directory structure



Model based testbench creation

Text files (sample input provided by user)



Results

Actor	Inputs	Outputs	Determ-inistic	Model Detected	State
Cluster Thresh	12	12	Yes	HSDF	No
Cluster Compute	12	6	Yes	HSDF	No
Overlap Filter	8	4	Yes	SDF	No
Jet Reconstruction	1	2	Yes	SDF	No

(H)SDF = (homogeneous) synchronous dataflow

A. JAMES CLARK SCHOOL OF ENGINEERING

Outline

- Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
- Application case study: high energy physics
- → Wrapup

UMIACS University of Maryland Institute for Advanced Computer Studies
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Summary

- The dataflow interchange format (DIF) project
  - The DIF Language (TDL)
  - The DIF Package (TDP)
  - Plug-ins for simulation and synthesis
- The DSPCAD Integrative Command Line Environment (DICE)
- Application case study: high-energy physics
- Other ongoing application thrusts in the DIF project include: embedded speech processing, software-defined radio, wireless sensor networks, image registration, radio astronomy instrumentation
- Co-design of dataflow-based representations and transformations

A. JAMES CLARK SCHOOL OF ENGINEERING

Acknowledgements

- Portions of the work presented here have been sponsored by DARPA (through MCCI), and NSF (ECCS0823989 and CNS0720596).

- For more details on these projects, and associated publications: **http://www.ece.umd.edu/DSPCAD/home/dspcad.htm.**

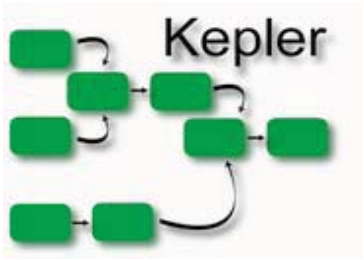
UMIACS University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

To Probe Further …

(Available from: http://www.ece.umd.edu/DSPCAD/papers/contents.html)

- [1] W. Plishker, C. Shen, S. S. Bhattacharyya, G. Zaki, S. Kedilaya, N. Sane, K. Sudusinghe, T. Gregerson, J. Liu, and M. Schulte. Model-based DSP implementation on FPGAs. In *Proceedings of the International Symposium on Rapid System Prototyping*, Fairfax, Virginia, June 2010. Invited paper.
- [2] S. S. Bhattacharyya, S. Kedilaya, W. Plishker, N. Sane, C. Shen, and G. Zaki. The DSPCAD integrative command line environment: Introduction to DICE version 1. Technical Report UMIACS-TR-2009-13, Institute for Advanced Computer Studies, University of Maryland at College Park, August 2009.
- [3] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya. Functional DIF for rapid prototyping. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 17-23, Monterey, California, June 2008.
- [4] C. Hsu, F. Keceli, M. Ko, S. Shahparnia, and S. S. Bhattacharyya. DIF: An interchange format for dataflow-based design tools. In *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, pages 423-432, Samos, Greece, July 2004.

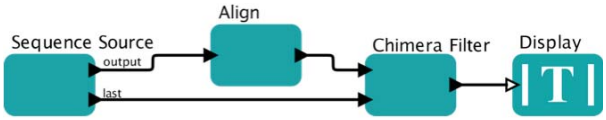


Workflow Fault Tolerance for Kepler

Sven Köhler, Timothy McPhillips, Sean Riddle, Daniel Zinn, Bertram Ludäscher

Introduction

- Scientific Workflows
  - Automate scientific pipelines
  - Have long running computations
  - Often contain stateful actors
- Workflow execution can crash because of …
  - Hardware failures
  - Power outages
  - Buggy / malicious actors, …
- Current approach: Start workflow from the beginning



Current Fault Tolerance Solutions …

- Manage actor failures or sub-workflow failures AND their effects
  - Atomicity and provenance support for pipelined scientific workflows [Wang et al.]
  - Ptolemy's "Backtrack" [Feng et al.]
- Use caching strategies for faster re-execution
  - W.A.T.E.R.S. memoization [Hartman et al.]
  - "Skip over" strategy [Podhorszki et al.] (CPES)`

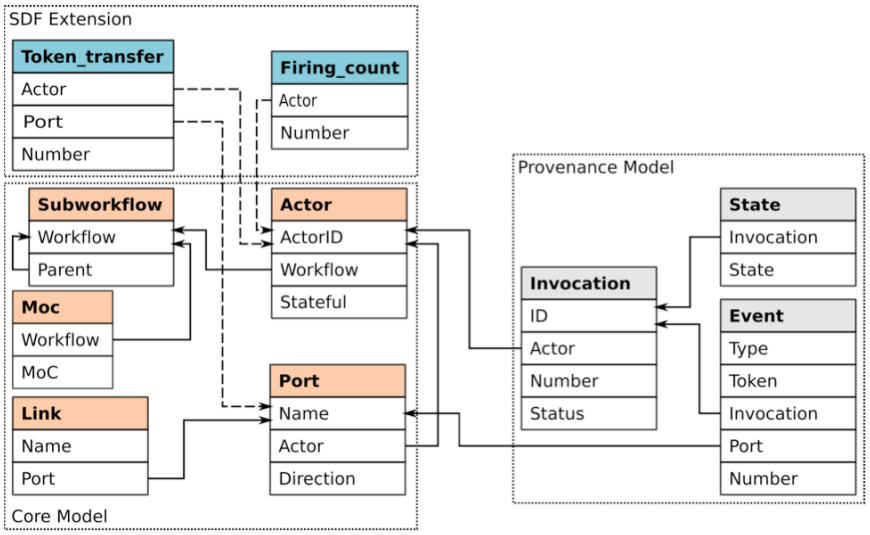
Our Fault Tolerance Approach

- Recovery based on readily available Provenance

1. Create a uniform model for workflow descriptions and provenance
2. Record actor state in provenance in relation to invocations
3. After a workflow crash: Use provenance data in our uniform model and start recovery

- Different strategies for recovery

Model for Workflows and Provenance



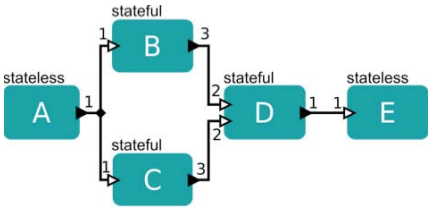
Our Recovery Strategies

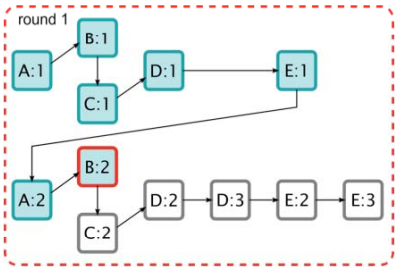
Strategy	Description
Naive	- Restart the workflow without using provenance - Re-executes everything
Replay	- Use **basic provenance** to speed up recovery - Re-execute stateful actor with input from provenance (*replay*) - Restore all queues - Resume the workflow according to the model of computation
Checkpoint	- **extension** of **replay** strategy - Use **checkpoints** (state of actors stored in provenance) - Reset stateful actors to recorded state - Replay successful invocations after the checkpoint - Restore queue content - Resume the workflow

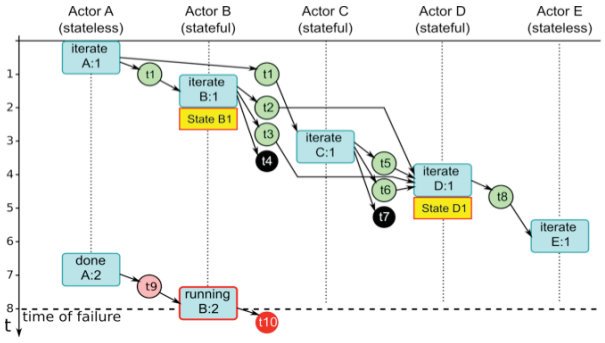
Example: Checkpoint in SDF

Workflow with a mix of stateful and stateless actors

Corresponding schedule of the workflow with a fault during invocation B:2







Execution with a Failure

Execution of the previous workflow

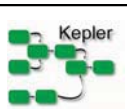
Checkpoints for actor B and D but not for C

At invocation B:2 - Crash

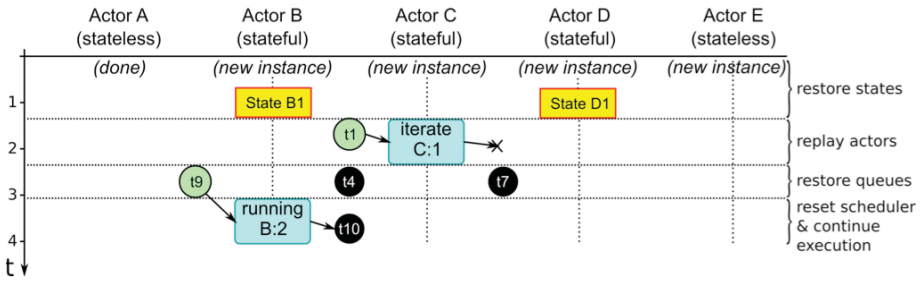
Tokens **t4** and **t7** - in queue

Token **t9** - to be restored

Token **t10** - to be deleted



Stages of Checkpoint Recovery



Prototype Implementation in Kepler

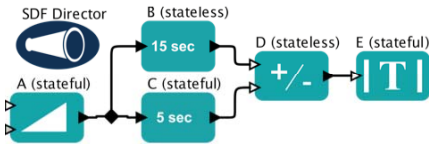
- Using Kepler with the Provenance Recorder
- Extensions to the Provenance Recorder:
  - Record serialized tokens
  - Extend the provenance schema
  - Add queries
- Recovery Extension in the SDF Director:
  - Serialize states after one iteration of the SDF schedule
  - Black-list to prevent capturing transient actor information
  - White-list if actors are annotated with state-information

Prototype Implementation in Kepler

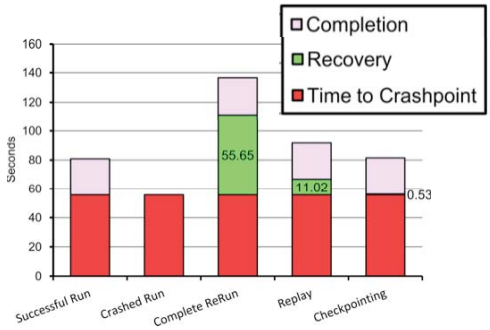
- Upon restart:
  - SDF director checks provenance information
  - SDF director calls the recovery engine
- Recovery:
  - Restore the internal state of actors
  - Replay successful invocations using input tokens from provenance
  - Restore content of all queues
  - Return to SDF director with information about where to resume

Evaluation

Synthetic Workflow



Results



Conclusion

- Advantages of our strategy:
  - Efficient workflow recovery using readily available information
  - Quick constant time recovery (checkpoint strategy)
  - Generalized approach, saving labor
  - Robustness
- Disadvantages of previous strategies:
  - Required labor-intensive customized systems
  - Failure required restarting long-running workflows from the beginning
  - Caching only works for stateless actors
  - Caching only provides a partial recovery

Design, Analysis, and Implementation of Static Dataflow Models for Hardware Targets,

Kaushik Ravindran, Murali Parthasarathy, et. al, (*National Instruments*)

Abstract: We present the DSP Designer framework to implement applications specified in the Static Dataflow (SDF) model of computation on hardware targets, such as FPGAs. Prior studies have shown the effectiveness of SDF as a natural model to specify multi-rate streaming applications. However, the focus of these works has primarily been on SDF implementations for processor targets. DSP Designer specializes the SDF model to make it suitable for hardware targets. It facilitates hardware actor definition and intellectual property (IP) integration. The back end additionally provides analysis methods tuned for synthesis of efficient hardware designs, such as resource allocation, memory optimization, and scheduler generation. The objective is to deliver an exploration framework that empowers application domain experts to become hardware designers. In this talk, we highlight key concepts underlying DSP Designer, demonstrate preliminary capabilities for exploration and implementation using practical applications, and discuss open challenges related to the specification of control and timing along with dataflow. We also summarize key features of the DSP Designer software architecture and invite partners to leverage our infrastructure and API to build tools for graphical design.

Kepler/G-Pack: A Kepler Package Using the Google Cloud for Interactive Scientific Workflows (a.k.a. Koogle-Kuration Package)

Gongjing Cao, Lei Dou, Quinn Hart, Bertram Ludaescher
UC Davis

9th Biennial Ptolemy Miniconference

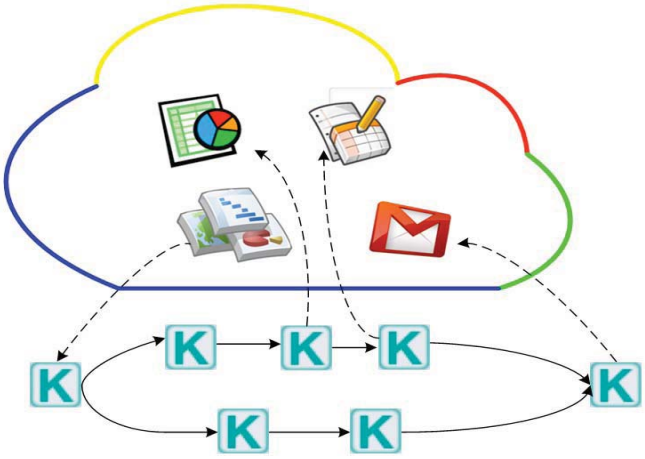
Berkeley, CA
February 16, 2011

Interactive Scientific Workflows

- Requirements for human interaction in scientific applications
  - dynamic branching based on scientists' runtime decision
  - semi-automatic data curation
- Category of human interaction

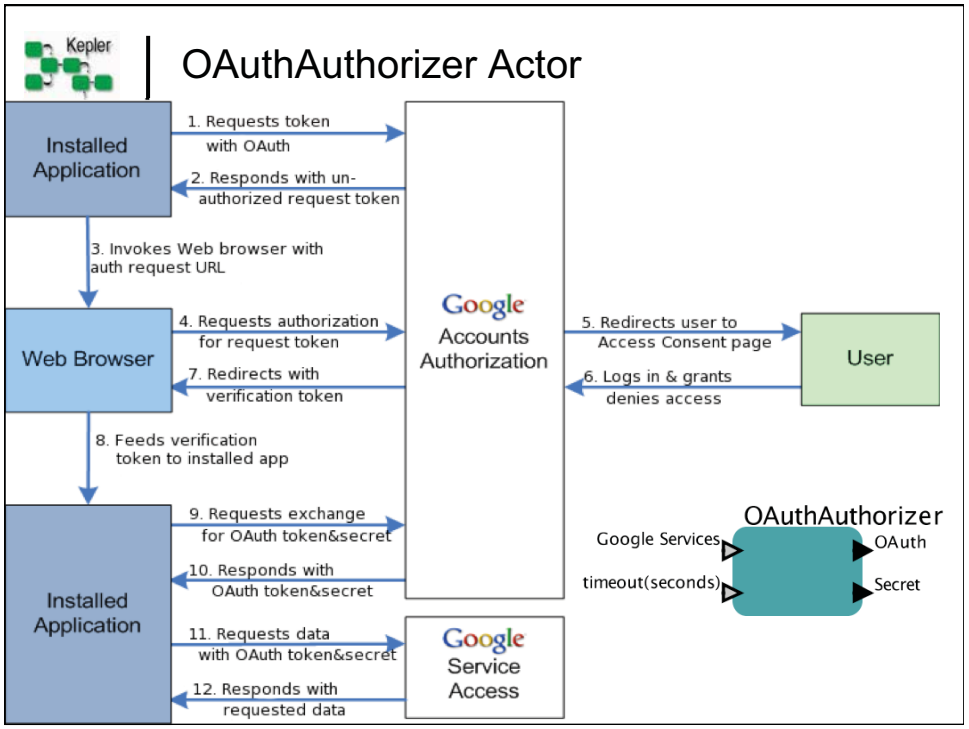
	Synchronous	**Asynchronous**
time cost	short time, instantly	unknown
people involved in interaction	workflow executor	people other than workflow executor
workflow blocking	yes	not necessarily
implementation approaches	graphical window web page/browser	dedicated server mail, polling, callback

Google Cloud Computing in Kepler



Actors in Kepler/G-Pack

- Authorization
  - 1st step to acquire access to Google services
- Spreadsheet Operations
  - Various manipulations on Google Spreadsheet, like copy, share, import, export, query, audit.
- Data Analysis
  - Various operations especially for data curation purpose, like duplicates identification and fuse, data boundary inspection.
- Data Access
  - Google visualization datasource actor allows SQL-like access to Google cloud data
- Mail Service
  - MailSender Actor supports sending email through SMTP with UserName/Password or OAuth token/secret.



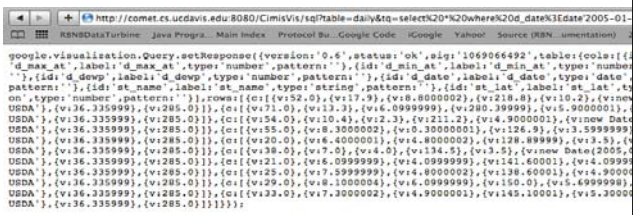
Actors	Functions
Importer	import data to a spreadsheet
Exporter	export data from a spreadsheet
Copy	copy a spreadsheet from a template
Share	share the spreadsheet with another user
Query	query data from the spreadsheet
Auditor PollingQuery	allow human interaction during the execution of the workflow

Data Access
VisualizationDataSource Actor

- Get the response from a servlet which is implemented with Google visualization datasource.
- e.g. servlet:
- http://comet.cs.ucdavis.edu:8080/CimisVis/sql

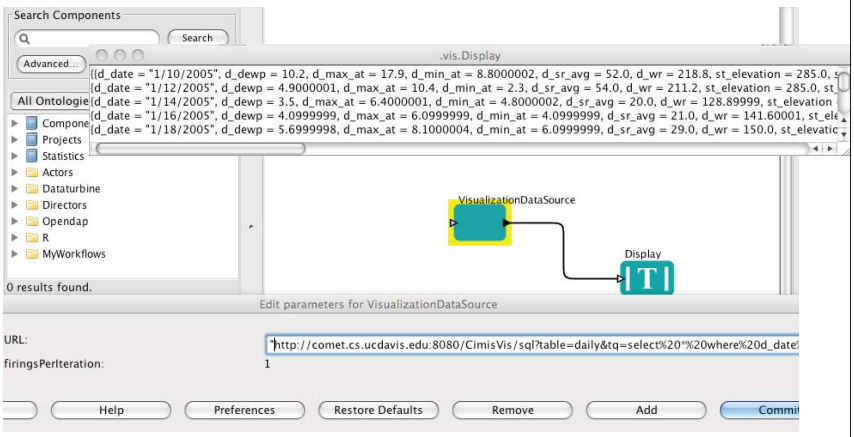
  Table: daily

query:select * where d_date>date'2005-01-09' and d_date<date'2005-01-20'



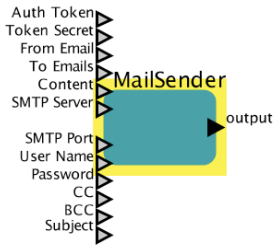
Data Access
VisualizationDataSource Actor

- Actor parses JSON string to Kepler token
- array of RecordToken

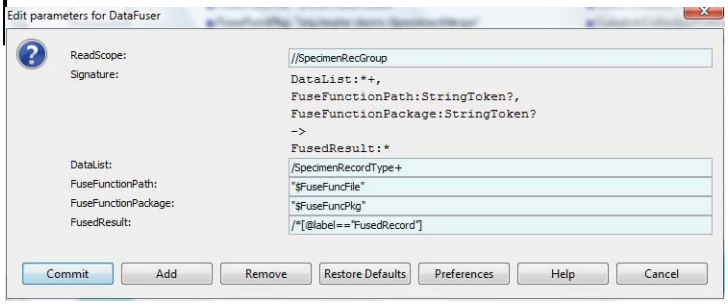


MailSender Actor

- Gmail and other mail server supporting SMTP protocol
  - It supports sending email through SMTP with UserName/ Password or OAuth token/secret.



Data Analysis Actors (COMAD actors)



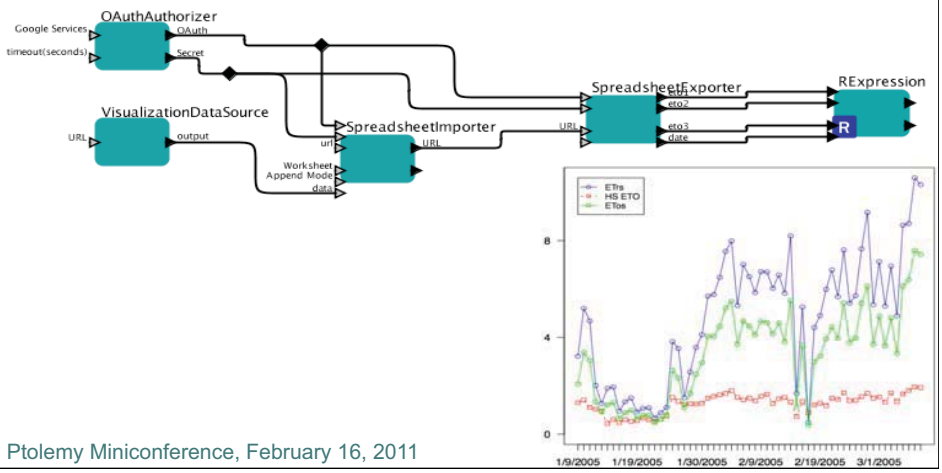
Actors	Functions
Clustering	Do clustering on a list of RecordToken by applying specified function for specified field.
DataFuser	Fuse a list of RecordToken with specified function
ConditionTester	Test whether specified condition is satisfied or not.

Evapotranspiration Workflow

SDF Director

Evapotranspotation Workflow – Koogle demo1

Spreadsheet is used as data storage and calculateation tool during this demo.
ETOs are calculated from CIMIS wheather station data according to different models

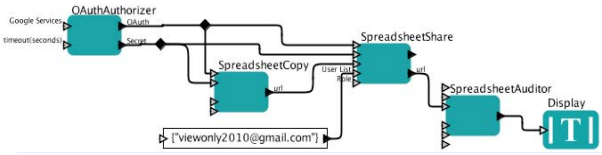


Biofuel Refinery Workflow

SDF Director

biofuel refinery workflow

This workflow allows users to copy a spreadsheet from a templete, then use the existing data or edit the spreadsheet to help them make dicision in building a biofuel refinery.



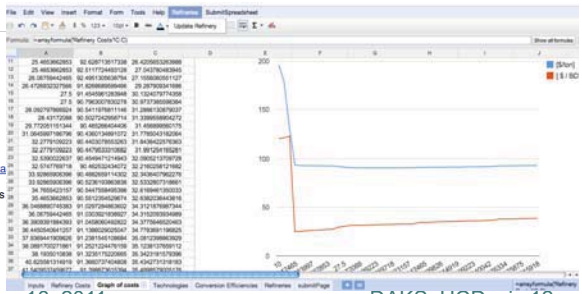
refinery-copy2 Inbox | x

from kepler.ucd@gmail.com
to viewonly2010@gmail.com
date Fri, Feb 11, 2011 at 9:06 AM
subject refinery-copy2
mailed-by doclist.bounces.google.com

I've shared a document with you:

refinery-copy2
https://spreadsheets.google.com/ccc?key=0AgT1sEGRa

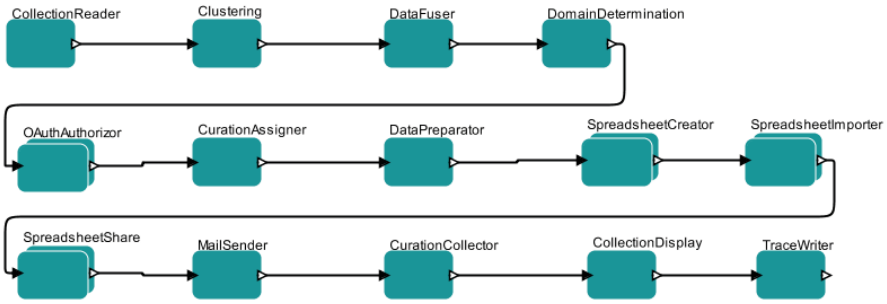
It's not an attachment -- it's stored online at Google Docs



SpecimenRecordMerge Workflow

This workflow demonstrates how specimen data with duplicates are fused and curated semi-automatically.

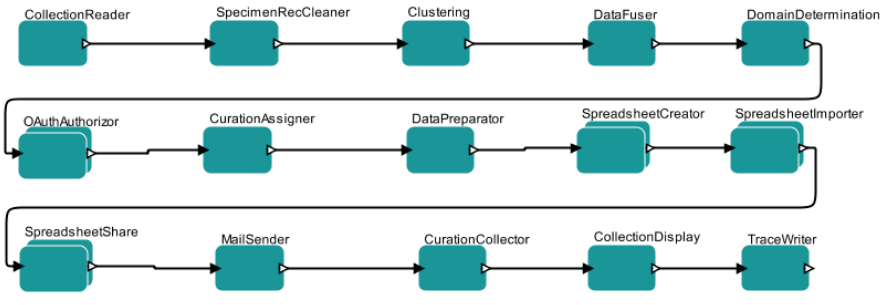
1. Firstly the specimen records are imported from a file in csv format.
2. Secondly the duplicated sets of specimen records are identified through specific clustering function and for each duplicate set a fused record is generated.
3. Thirdly the original specimen records and the fused record are imported into a google spreadsheet for human curation. Meanwhile the corresponding curators are informed of such curation request.
4. Finally, the human curation result is collected once it's finished by the curator.



AdvancedSpecimenRecordMerge Workflow

This workflow makes the following improvement of SpecimenRecordMerge workflow.
It's demonstrated how easy it is to reconfigure COMAD workflow to adapt to new functionalities.

1. Insert SpecimenRecCleaner actor to clean source data by removing "bad" specimen records with unclear collector of "et al."
2. Reconfigure Clustering actor to cluster specimen records against collector field with fuzzy match method. Therefore the specimen records collected at the same time and the same location but by different collector of "E. L. Morris" and "E. Morris" could be identified as duplicates.



Thanks & Question

Context-Aware Actors

Anne H. H. Ngu
Department of Computer Science
Texas State University-San Marcos

Outline

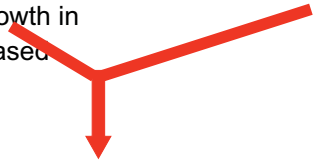
- **Why Context-Aware Actor?**
- **Context-Aware Scientific Workflow System**
  - **Architecture**
  - **Context Modelling and Provisioning**
  - **Context Providers and agents**
  - **Context Annotation**
  - **Demo of Context-Aware actors**
  - **Conclusion and future work**

Why Context-Aware Actors?

- Actor-oriented scientific workflow model - ideal for modeling data intensive, stream-based and concurrent execution nature of scientific workflow.
- Problems with using Actor-oriented model for modeling dynamic workflows.
  - too many low-level control-flow actors and wiring lead to complex workflow that is hard to comprehend and re-use.
- Proposed solution in the past: Static Frame and Template, Dynamic frame actor, and Generic actor.
  - Static frame and template cannot adapt to runtime conditions (SciFlow 2006).
  - Dynamic frame encodes control-flow implementation of runtime conditions in the frame actor, thus cannot adapt to new situation without re-implementation (SSDBMS 2009).
  - Generic actor encodes all variations in control-flow logic a-priori in the actor.
  - All involve knowing low-level actor programming skills.
- Context-aware actor is proposed to enable actor-oriented scientific workflow to be more personalized, adaptive, intuitive, and intelligent by modeling the logic related to quality runtime adaptations as contexts and provisioned by a separate computing unit to the actor.

Context-Aware Actors

- **Context awareness**: the capability of being aware of its physical environment or situation (context) and responding proactively and intelligently based on such awareness
- One of the most important trends in computing that is becoming more important with relentless growth in mobile services (location-based services)



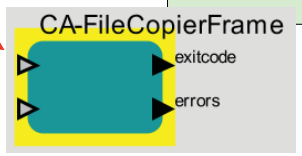
- **Actor**: a reusable component that may encapsulate external computation programs, grid services, scripts or local applications.
- **Context-aware Actors (CAA):** *make actors more personalized*, adaptive, and *intelligent*

Context-Aware FileCopier Actor
An Example CAA

User enters source file, destination file, source machine, destination machine, and a high-level file transfer option (e.g., fast protocol)

Recommend fastest type of protocol to transfer file taking user's contextual information such as size of the source file, availability of protocols and speed of the network connection between machines.

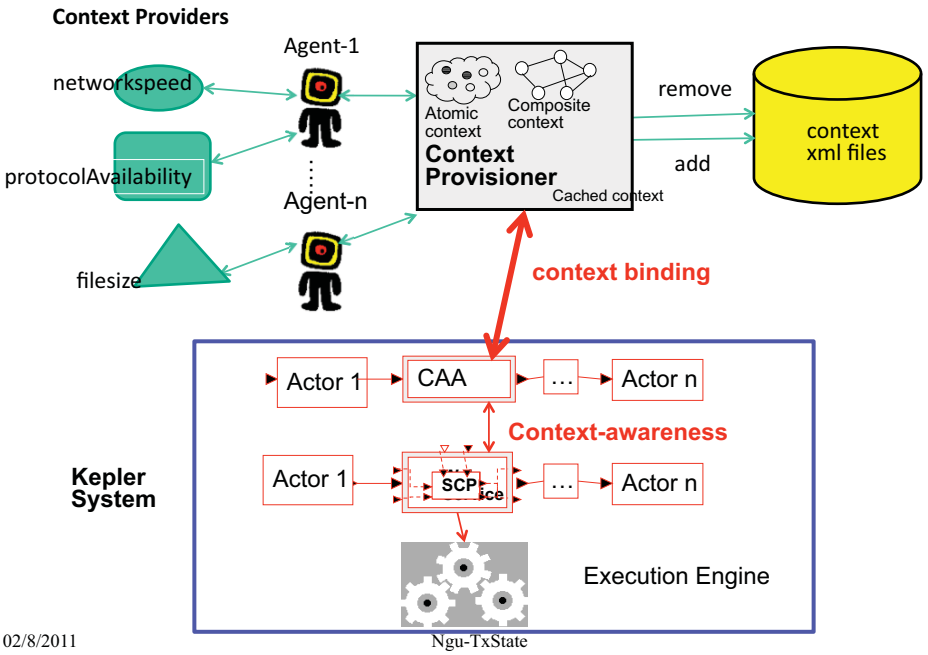
Example of contexts used for deciding on a protocol are:
If filesize is > 6GB and network connection speed is < 6ms, use bbcp protocol.
If bbcp protocol is not available and recursive option is set, use scp protocol



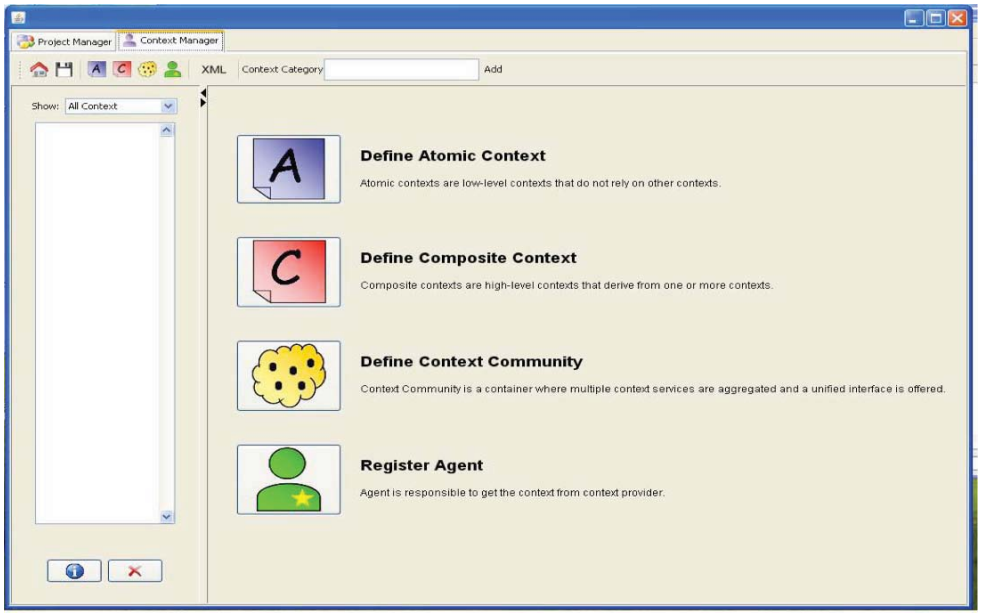
Context and Context-Awareness

- Context:
  - All data that can be gathered automatically at runtime which can affect the actor's behavior :
    - System parameters (OS, CPU usage, job queue status, network speed, type of machines, availability of resources and their versions, occurrences of certain events)
  - All data supplied by the user (gathered manually) especially those data related to his/her preferences. This can include strategies that can be applied to obtain specific level of guaranteed quality
- Context-Awareness
  - The mechanism for adapting the execution of an actor based on the sensed /gathered contextual information.

Kepler Context-Aware Scientific Workflow Architecture



Context Provisioning Server



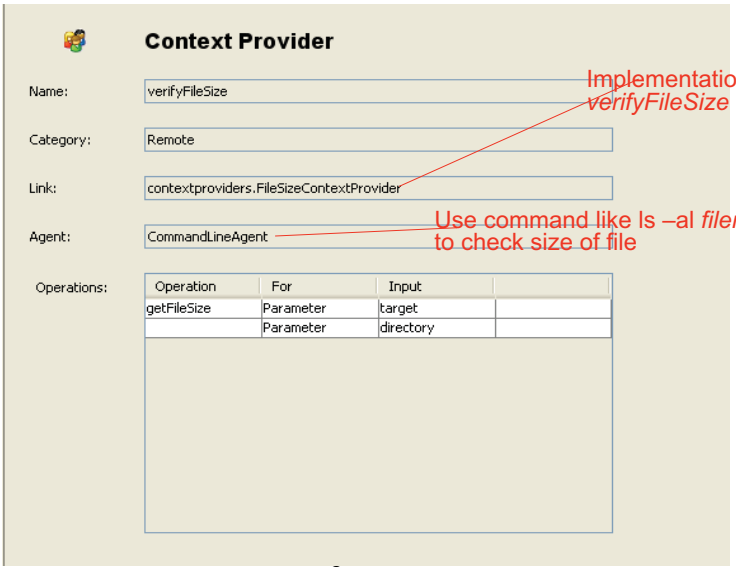
Context Provisioning Server (CP)

- Is adapted from the ContextServ project whose goal is to provide a platform for rapid development of context-aware web services. It is funded by Australian Research Council.
- CP is responsible for setting up, acquiring, and managing the contexts used by any context-aware actors.
- CP contains a set of agents and a repository of context XML files (context specification)
- The context XML files store the specification of all contexts used by a specific project. CP uses the context XML files to figure out how to acquire defined context information from the appropriate agents.
- CP is implemented as a web service using Apache CXF (a light-weight Java based web services development tool kits)

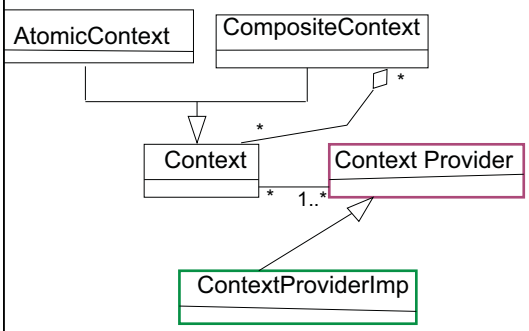
Context Providers

- Context providers are the context sources. They are independent from CP.
- There are many different kinds of context providers (hardware sensors, PDA, software systems, web services).
- Agents are used to provide a uniform abstraction for obtaining context from a particular kind of context provider (e.g. Webservice agent is used to acquire contexts from all context providers which can be queried through SOAP protocol)
- Context providers can be added and removed anytime.
- It is the responsibility of the context provider to provide the implementation of services (i.e. gathering of the contextual data).

An example of a context provider

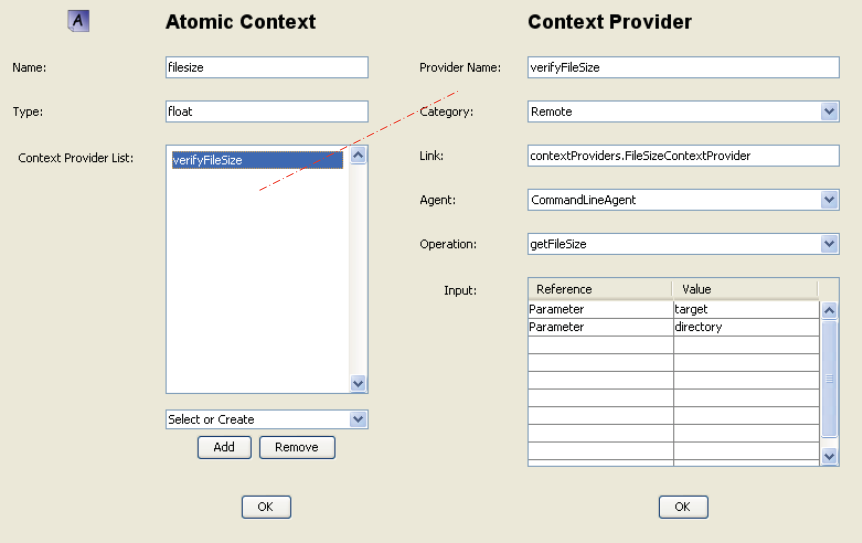


Context Modelling



- Context Provisioner distinguishes between *atomic* contexts and *composite* contexts
- **Atomic contexts**: low-level contexts, directly provided by context providers
  - e.g., filesize, temperature,
- **Composite contexts**: high-level contexts, no direct providers, aggregate multiple atomic or composite contexts
  - e.g., fastProtocol
  - Provide more powerful context modelling mechanism
- Context can be provided by one or more context providers

Example of Atomic Context



XML representation of atomic context

```
<context>
   <name>filesize</name>
   <type>float</type>
   <category>Atomic</category>
   <context_provider>
        <name>verifyFileSize</name>
         <category> Remote</category>
         <agent>commandLineAgent>
         <link>contextProviders.FileSizeContextprovider</link>
          <operation>getFileSize</operation>
          <input ref="Parameter"> target</input>
          <input ref="Parameter">directory</input>
       </context_provider>
 </context>
```

Rules for recommending a fast protocol defined as a composite context

Fast protocol:

If ((bbcp is available)

// big file, slow network speed, so better off with a fast protocol

if (filesize is >=6 GB) & (network speed > =6ms)) then choose bbcp

else if (filesize < 6 GB OR networkspeed < 6 ms)

if recursive transfer

choose scp

else

if (sftp available) choose sftp

else choose scp

else

if recursive transfer

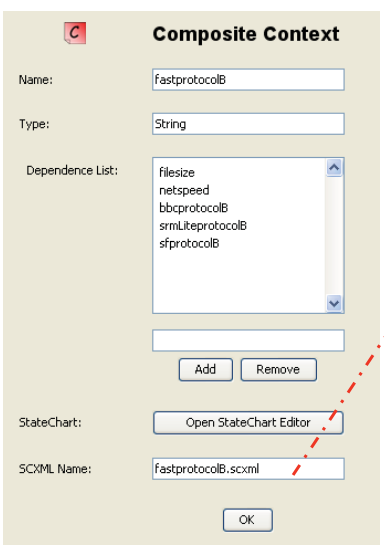
choose scp

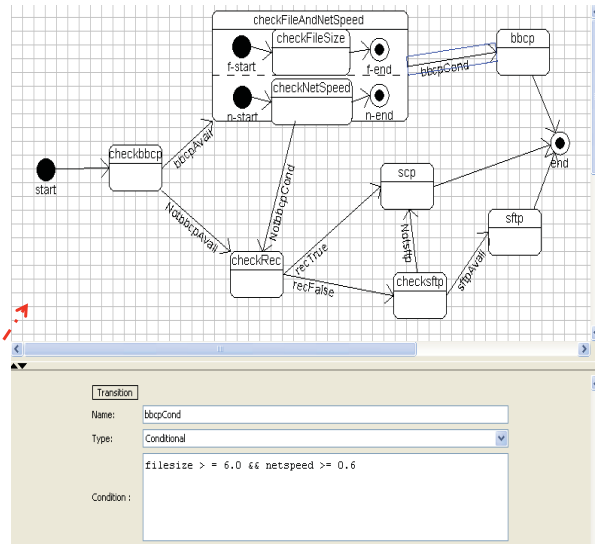
else

if (sftp available) choose sftp

else choose scp

Example of Composite Context

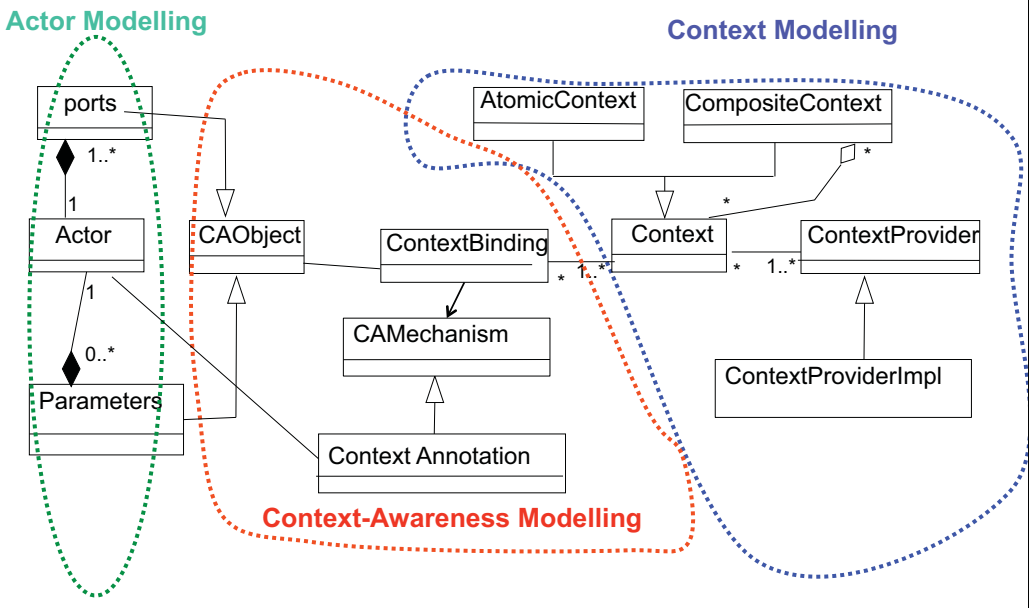




Representation of composite context

```
<context>
    <name>fastprotocolB</name>
    <type>String</type>
    <category>Composite</category>
    <dependence>filesize</dependence>
    <dependence>netspeed</dependence>
    <dependence>bbcprotocolB</dependence>
    <dependence>srmLiteprotocolB</dependence>
     <dependence>sfprotocolB</dependence>
     <scxml>fastprotocolB.scxml</scxml>
 </context>
```


Development of Context-Aware Actor



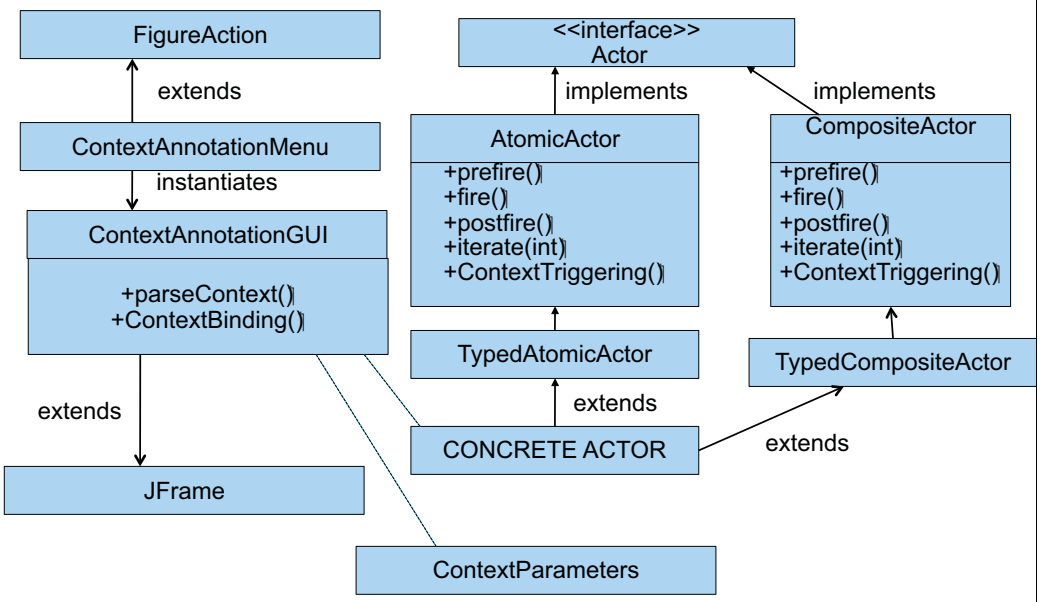
Context Awareness Modeling

- Context binding is the ability for an actor to bind to/access relevant contexts defined in the CP.
- Context awareness is the mechanism by which an actor behavior can be modified based on the gathered contextual information.
- A context-aware scientific workflow system must implement a context binding mechanism and a context-triggering mechanism.

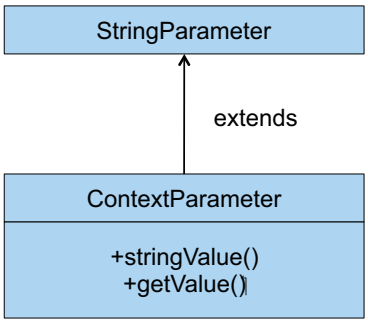
Context Annotation as the Awareness Mechanism

- The main purpose of context annotation is to facilitate an end user (scientist) to tailor an existing actor in the KEPLER repository to his/her environmental context by interacting with a visual annotation tool. Context is injected into an existing workflow on demand –basis.
- The end user must identify context sensitive parameters (CAObject) in an actor and bind them to appropriate contexts. We assume that the end user understand the semantics of the defined contexts in his/her domain.
- Two new classes, ContextAnnotationGUI and ContextParameter are introduced. Context annotation encapsulates all user interactions with the context provisioning server that will result in correct context binding.
- ContextParameter class encapsulates actions associated with contacting the context provisioning system to provide real-time evaluation of the contextual value of a context sensitive port or parameter in an actor.
- Context Annotation provides a generic mechanism for incorporating context without any code changes.

Implementation of Context Annotation



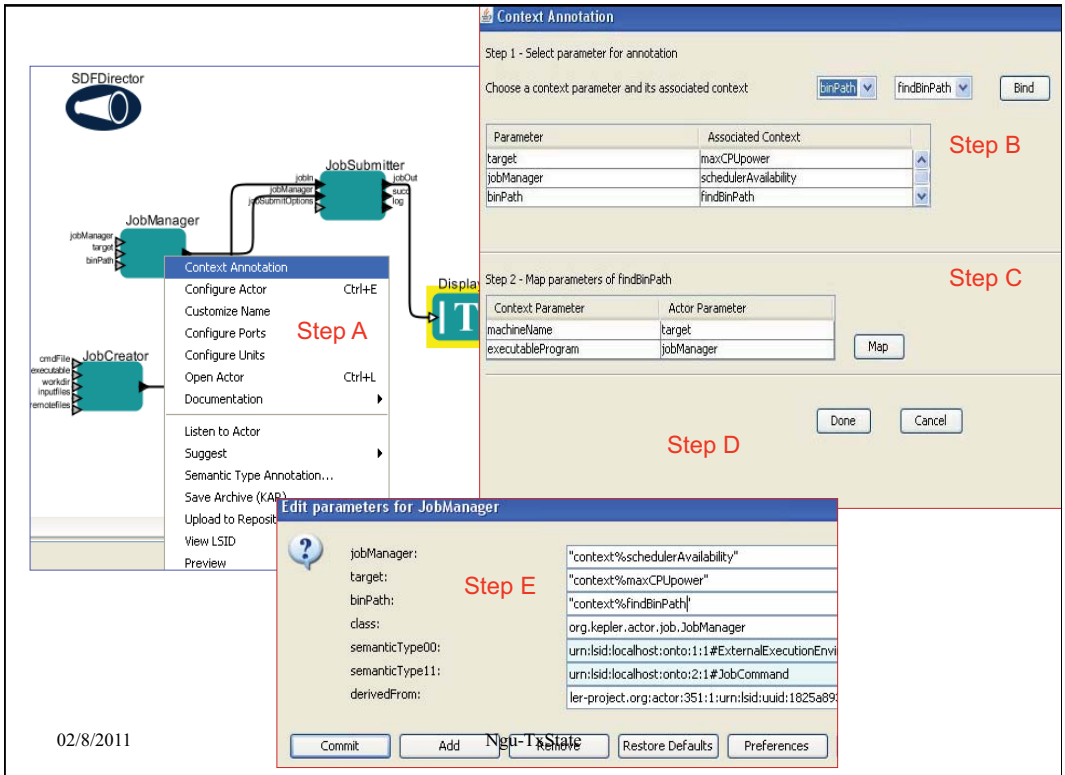
Implementation of ContextParameter




```
public String getValue(Object[][] paras) throws
IllegalActionException
{
// contacts the Context Provisioning server
//saves the list of parameters from an actor
//uses getContext() API to evaluate the context
```

Steps in Context Annotation

1. From the GUI, display all the existing parameters or ports of the actor that are candidates for context annotation.
2. Let the user selects the context parameter for annotation.
3. Contacts the CP server to find the list of contexts that is relevant to the selected context parameter.
4. Let the user picks the required context to bind to.
5. If the binding requires mapping, map the context variables with actor parameters.
6. Click “map” to finish the binding and repeat 2 to 5 for the next parameter.
7. The context triggering phase started when user pressed the done button and run the actor.
8. The context computation takes place during the firing of the actor



Summary of Context Annotation

- Context annotation presented a new framework that enabled scientific tasks to exploit dynamic environmental information during runtime without introducing complex control-flows and additional proliferate actors.
- Context annotation enables different kepler's actors to be context-aware without any low-level re-coding of the actors.
- Context annotation simplifies the construction of scientific workflows that involve intricate adaptive behaviour, especially when a myriad of environmental conditions must be checked and verified. When environment conditions must be checked in different workflows, it can be outsourced to the CP rather than coding those conditions in every wokflow.
- Intelligent defaults set up in contexts can be re-used across different workflows. This simply the actor programming.
- Context annotation provides the ability for the end user to plug in different contexts for the different situations under which a workflow can be executed. For example, the ability to choose the machine with maximum CPU, the network with shortest delay, or the protocol with highest reliability transparently for the users. It hides the complexity of running the workflow in different execution environments.

Future Work

- Apply context-aware model to Cyber Physical System
  - power grids, medical device systems, traffic control systems, environmental monitoring all invariably must deal with real time changing physical conditions. Can a more robust and adaptive CPS system be built with context-aware model?
- Integrate context-aware model with scientific workflow template design workbench.
  - Workflow templates encapsulate both control flow and data flow patterns that can be reused and adapted by scientists with minimal configuration in their pursue in designing and executing their own scientific processes. Actor and parameter bindings in template can benefit from the additional contextual information.
  - Context-driven binding of templates will result in a configured workflow that is of better quality.

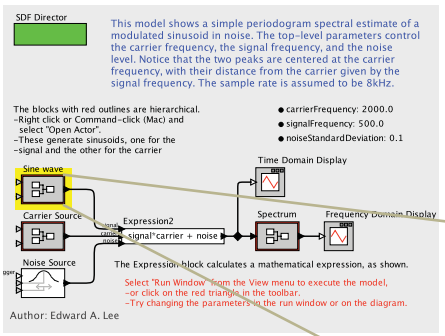
Modular Synchronous Dataflow Code Generation

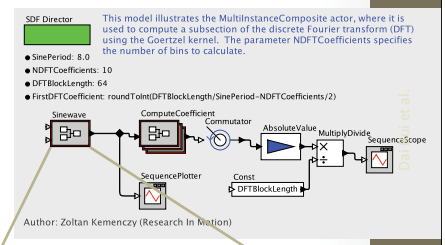
Dai Bui, Stavros Tripakis, Marc Geilen, Bert Rodiers, Edward A. Lee
UC Berkeley

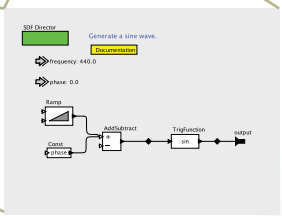
Ptolemy Mini-Conference, February 16, 2011

Modular Code Generation

- Code generated for a composite block independently from used contexts:



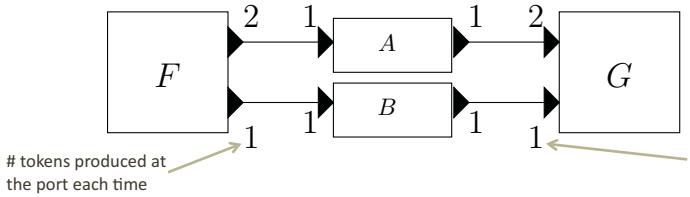




Motivations

- Reusability
  - Incremental compilation
  - IP protection
- Modularity
  - Unit verification and testing
  - Parallelization
  - Scalability
- Reduction of runtime overhead
  - Speeding up simulations

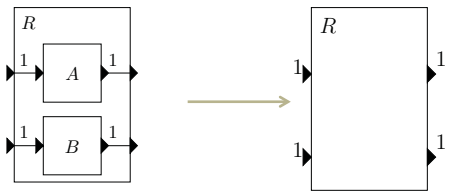
Naïve SDF Code Generation

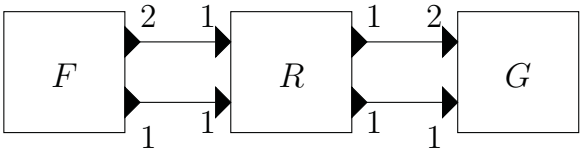


Finite-buffer schedule:FAABG

tokens produced at the port each time the actor fires

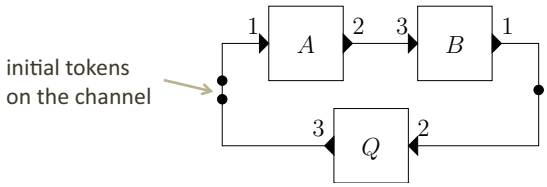
tokens consumed at the port each time the actor fires



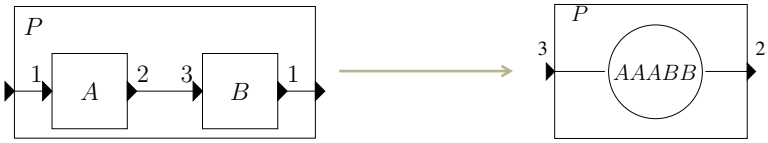


No finite-buffer schedule!

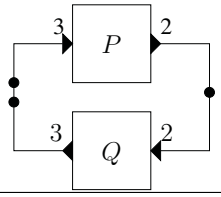
Naïve SDF Code Generation



Schedule: AABQAB



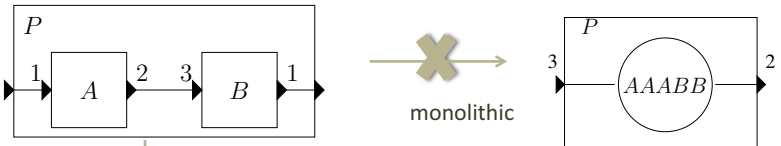
Schedule: AAABB



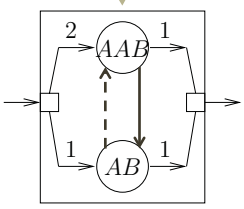
Deadlocked!

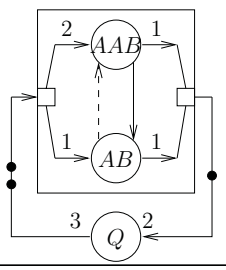
Modular SDF Code Generation

- Non-monolithic firing function
- DSSF = Deterministic SDF with Shared FIFOs



Non-monolithic How do we synthesize?



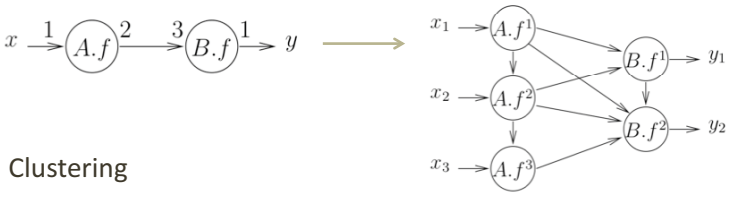


(AAB) (Q) (AB)

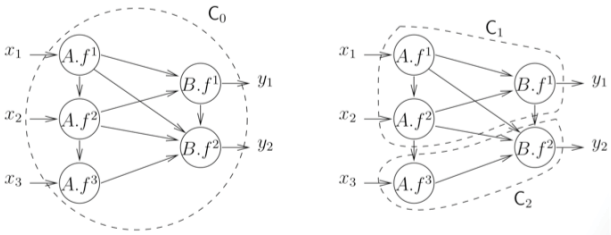
Can we do
(AAB)(Q)(AAB)
?

Synthesis

- Unfolding (non-homogeneous to homogeneous)



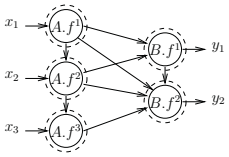
- Clustering

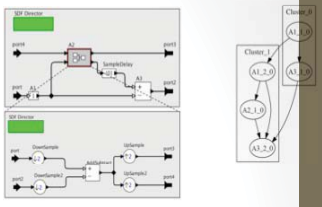


Clustering Algorithm Effectiveness

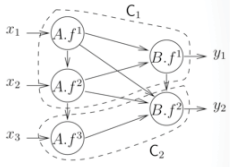
Valid clustering

yet bad!





Better clustering



Test cases	# ins/outs	# actors	# nodes in unfolding graph	# clusters
Test	2/2	3	5	2
Entropy	1/1	15	1545	1
CD to DAT	1/1	4	156	15

Conclusions

- Hierarchical SDF models are not compositional
- Introduce DSSF profiles as a compositional representation of composite actors and show how this representation can be used for modular code generation
- Propose a synthesis algorithm that can handle hierarchical models of arbitrary depth

Future Work

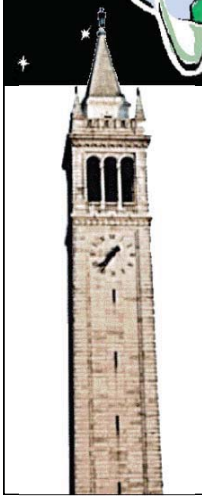
- Implement more advanced clustering algorithms to reduce the number of clusters
- Understand the relations between the number of clusters, the size of generated code and performance
- Estimate throughput and delay of a code-generated model

References

- Stavros Tripakis, Dai Bui, Marc Geilen, Bert Rodiers, Edward A. Lee, “Compositionality in Synchronous Data Flow: Modular Code Generation from Hierarchical SDF Graphs”. To appear at ACM Transactions on Embedded Computing Systems (TECS)
- Roberto Lublinerman, Christian Szegedy, Stavros Tripakis, “Modular code generation from synchronous block diagrams: modularity vs. code size”, POPL ‘09 Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languagesProceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.
- Joachim Falk, Joachim Keinert, Christian Haubelt, Jürgen Teich, Shuvra S. Bhattacharyya, “A Generalized Static Data Flow Clustering Algorithm for MPSoC Scheduling of Multimedia Applications”, EMSOFT ’08 Proceedings of the 8th ACM International Conference on Embedded Software.
- Joseph Buck, “Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model”, PhD Thesis, EECS Department, University of California, Berkeley, 1993.

Thank you

The Ptolemy Project
Advancing System Design



Edward A. Lee

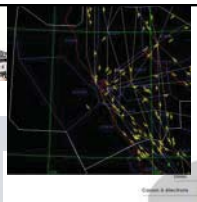
Robert S. Pepper Distinguished Professor

Ninth Biennial Ptolemy Miniconference

February 16, 2011
Berkeley, CA, USA

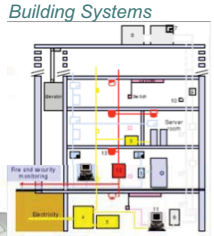
Cyber-Physical Systems (CPS):
Orchestrating networked computational resources with physical systems

Avionics



Transportation (Air traffic control at SFO)

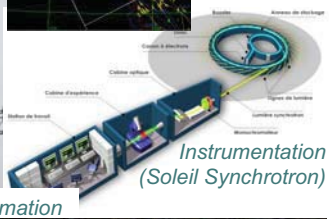
Building Systems



Telecommunications

Automotive

E-Corner, Siemens



Instrumentation (Soleil Synchrotron)

Daimler-Chrysler

Power generation and distribution

Military systems:

Courtesy of Doug Schmidt

Courtesy of General Electric

Factory automation

Courtesy of Kuka Robotics Corp.

CPS Example – Printing Press

Bosch-Rexroth

- *High-speed, high precision*
  - *Speed: 1 inch/ms*
  - *Precision: 0.01 inch*
    -> Time accuracy: 10us
- *Open standards (Ethernet)*
  - *Synchronous, Time-Triggered*
  - *IEEE 1588 time-sync protocol*
- *Application aspects*
  - *local (control)*
  - *distributed (coordination)*
  - *global (modes)*

Where CPS Differs from the traditional embedded software problem:

- *The traditional embedded software problem:*

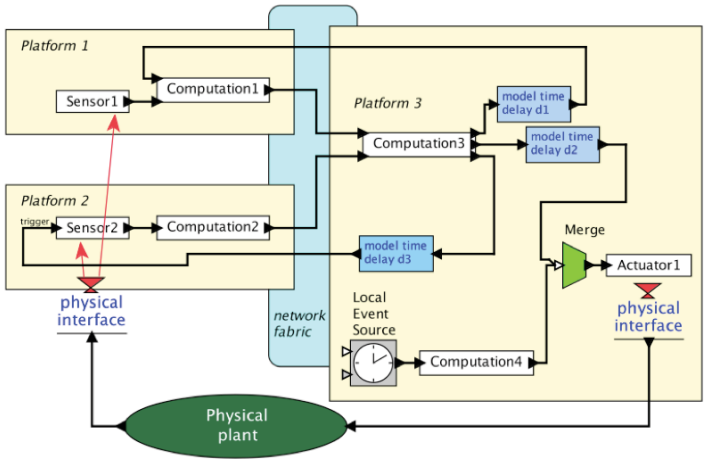
  Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources).

- *The CPS problem:*

  Computation and networking integrated with physical processes. The technical problem is managing dynamics, time, and concurrency in networked computational + physical systems.

Approaching the CPS Challenge

Physicalizing the cyber (PtC): to endow software and network components with abstractions and interfaces that represent their dynamics in time.



Cyberizing the Physical (CtP): to endow physical subsystems with cyber-like abstractions and interfaces

Projects at Berkeley focused on Physicalizing the Cyber

Time and concurrency in the core abstractions:

- *Foundations:* Timed computational semantics.
- *Bottom up*: Make timing repeatable.
- *Top down*: Timed, concurrent components.
- *Holistic*: Model engineering.

ON TO 2013! AND BEYOND!

Photo by Chamberlain Fong

Directions to the Faculty Club