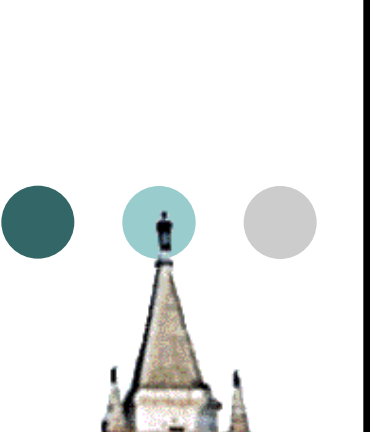




Scalable Models Using Model Transformation

Thomas Huining Feng

Ph.D. Student, UC Berkeley

Edward A. Lee

Robert S. Pepper Distinguished Professor, UC Berkeley

*1st International Workshop on Model Based Architecting and Construction of
Embedded Systems (ACESMB 2008)*

September 29, 2008

Toulouse, France



Context: Chess: Center for Hybrid and Embedded Software Systems

Principal Investigators

- Thomas Henzinger (EPFL)
- Edward A. Lee (Berkeley)
- Alberto Sangiovanni-Vincentelli (Berkeley)
- Shankar Sastry (Berkeley)
- Janos Sztipanovits (Vanderbilt)
- Claire Tomlin (Berkeley)



This center, founded in 2002, blends systems theorists and application domain experts with software technologists and computer scientists.

Executive Director

- Christopher Brooks

Associated Faculty

- David Auslander (Berkeley, ME)
- Ahmad Bahai (Berkeley)
- Ruzena Bajcsy (Berkeley)
- Gautam Biswas (Vanderbilt)
- Ras Bodik (Berkeley, CS)
- Bella Bollobas (Memphis)
- Karl Hedrick (Berkeley, ME)
- Gabor Karsai (Vanderbilt)
- Kurt Keutzer (Berkeley)
- George Necula (Berkeley, CS)
- Koushik Sen (Berkeley, CS)
- Sanjit Seshia (Berkeley)
- Jonathan Sprinkle (Arizona)
- Masayoshi Tomizuka (Berkeley, ME)
- Pravin Varaiya (Berkeley)

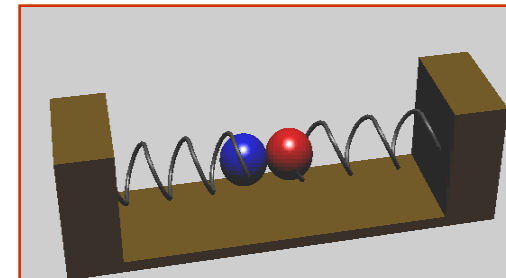
the Berkeley directors of Chess

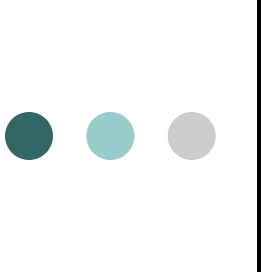
Some Research Projects

- Precision-timed (PRET) machines
- Distributed real-time computing
- Systems of systems
- Theoretical foundations of CPS
- Hybrid systems
- Design technologies
- Verification
- Intelligent control
- Modeling and simulation

Applications

- Building systems
- Automotive
- Synthetic biology
- Medical systems
- Instrumentation
- Factory automation
- Avionics

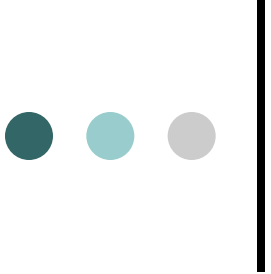




Model Transformation

Model-based design has used static structure models such as UML class diagrams to provide meta models that guide model-based design processes.

In this project, we are focusing more on actor models, which more directly express concurrency and system dynamics than what is possible with static structure models. Inspired by prior work on model-driven model transformation, we have prototyped a model-transformation mechanism that is actor-oriented.

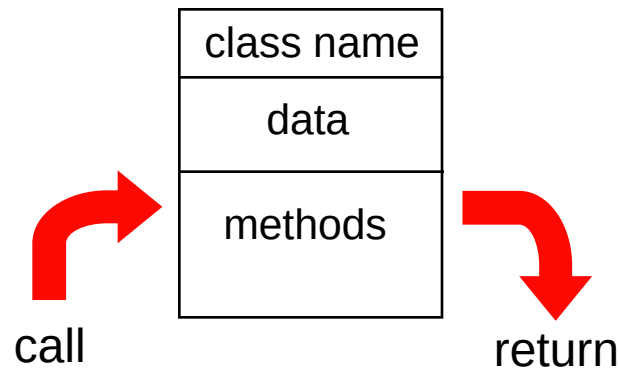


Inspirations and Influences

- AGG [Taentzer, 1999]
- AToM3 [Lara, Vangheluwe, 2002]
- FUJABA [Nickel, Niere, Zündorf, 2000],
- GReAT [Agrawal, Karsai, Shi, 2003]
- OMG MOF QVT (Query/Views/Transformations)
- PROGRES [Schürr, Winter, Zündorf, 1995]
- VIATRA2 [Balogh, Varró, 2006]

Our Premise: Components are Actors rather than Objects

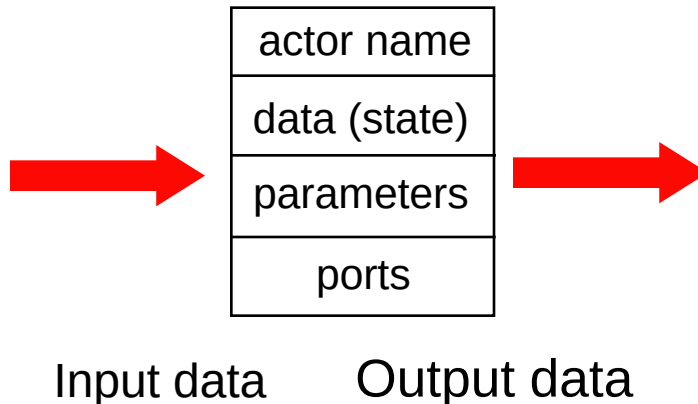
The established: Object-oriented:



What flows through
an object is
sequential control

Things happen to objects

The alternative: Actor oriented:



Actors make things happen

What flows through
an object is
evolving data

Ptolemy II: Our Open-Source Laboratory for Experiments with Actor-Oriented Design

<http://ptolemy.org>

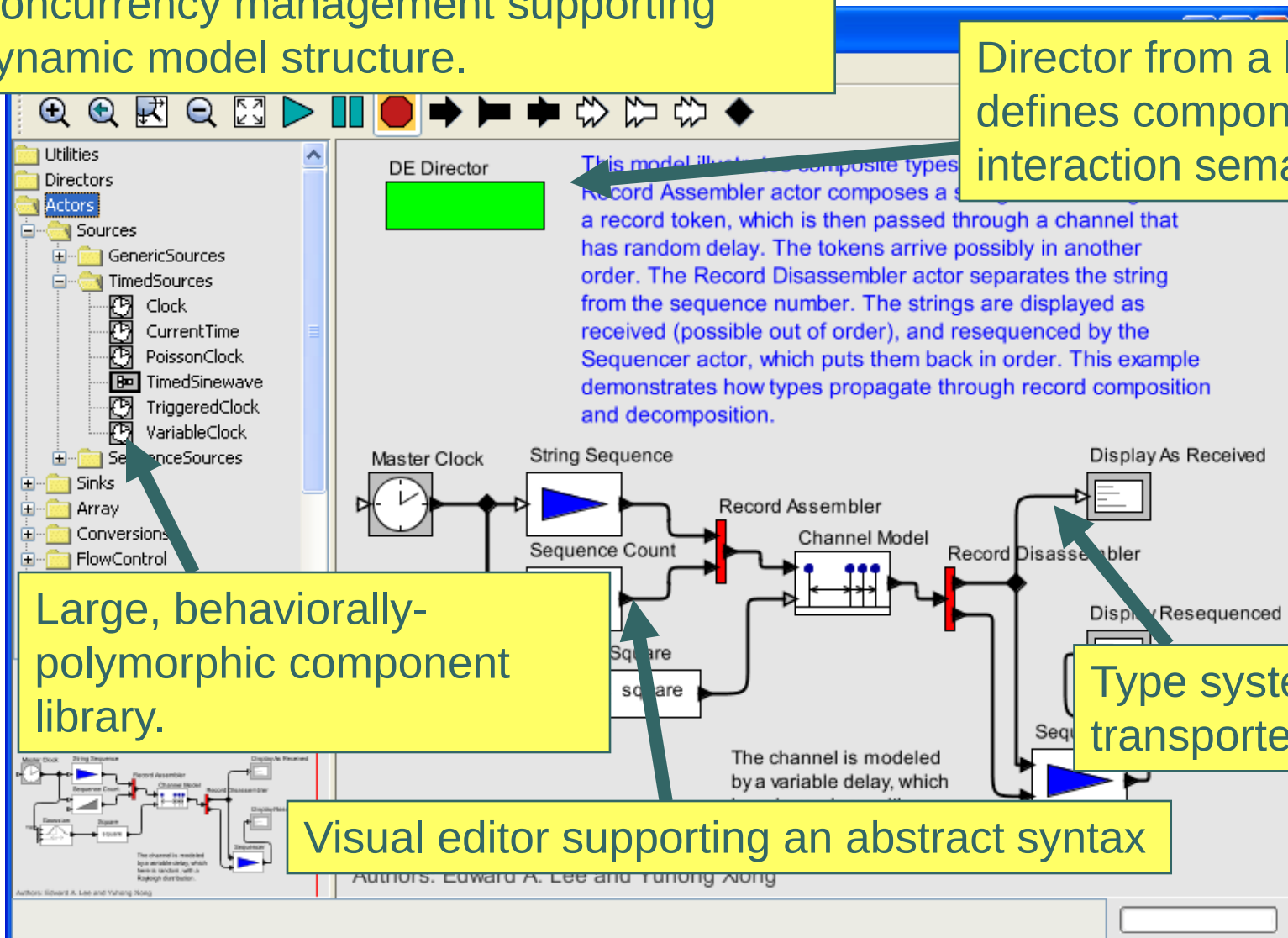
Concurrency management supporting dynamic model structure.

Director from a library defines component interaction semantics

Large, behaviorally-polymorphic component library.

Type system for transported data

Visual editor supporting an abstract syntax



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

The screenshot displays the Ptolemy II software interface. On the left, a file explorer shows the project structure: `file:/C:/ptll/ptolemy/data/type/demo/Router/Router.xml`. The main workspace contains a model diagram with components: **DE Director** (green box), **Master Clock** (clock icon), **String Sequence** (blue trapezoid), **Sequence Count** (triangle icon), and **Gaussian** (bell curve icon). A blue text box explains: "This model Record Ass a record to has random order. The from the se received (p Sequencer demonstrat and decom". A context menu is open over the Gaussian component, listing actions: **Customize**, **Documentation**, **Appearance**, **Save Actor In Library**, **Listen to Actor**, **Set Breakpoints**, **Convert to Class**, **Open Actor** (highlighted), and **Open Instance**. The **Open Actor** option has a keyboard shortcut **Ctrl+L**. Below the menu, the text "Authors: Edward A" is visible.

On the right, a code editor window shows the Java source code for the **Gaussian** actor: `file:/C:/ptll/ptolemy/actor/lib/Gaussian.java`. The code defines the **Gaussian** class, which extends **RandomSource**. It includes comments for parameters like **mean** and **standardDeviation**, and shows the constructor and **output** method. The code is as follows:

```
public class Gaussian extends RandomSource {
    /** Construct an actor with the given container and name.
     * @param container The container.
     * @param name The name of this actor.
     * @exception IllegalArgumentException If the actor cannot be contained
     *     by the proposed container.
     * @exception NameDuplicationException If the container already has an
     *     actor with this name.
     */
    public Gaussian(CompositeEntity container, String name)
        throws NameDuplicationException, IllegalArgumentException {
        super(container, name);

        output.setTypeEquals(BaseType.DOUBLE);

        mean = new PortParameter(this, "mean", new DoubleToken(0.0));
        mean.setTypeEquals(BaseType.DOUBLE);

        standardDeviation = new PortParameter(this, "standardDeviation");
        standardDeviation.setExpression("1.0");
        standardDeviation.setTypeEquals(BaseType.DOUBLE);
    }

    ///////////////////////////////////////////////////
    /////////////////////////////////////////////////// ports and parameters ///////////////////////////////////////////////////
    ///////////////////////////////////////////////////

    /** The mean of the random number.
     * @param mean has type double, initially with value 0.
     */
    PortParameter mean;

    /** The standard deviation of the random number.
     * @param standardDeviation has type double, initially with value 1.
     */
    PortParameter standardDeviation;

    ///////////////////////////////////////////////////
    /////////////////////////////////////////////////// public methods ///////////////////////////////////////////////////
    ///////////////////////////////////////////////////

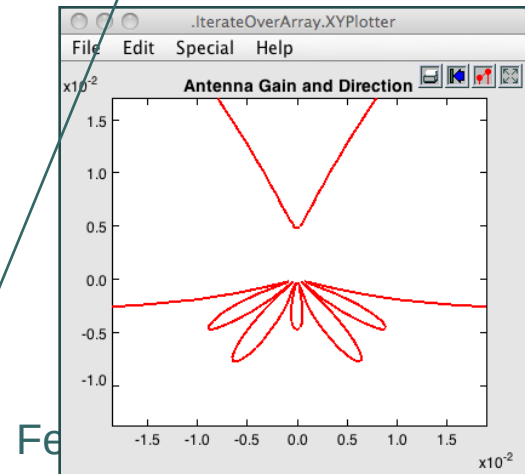
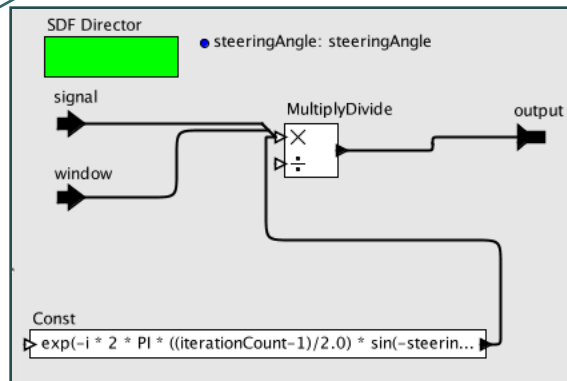
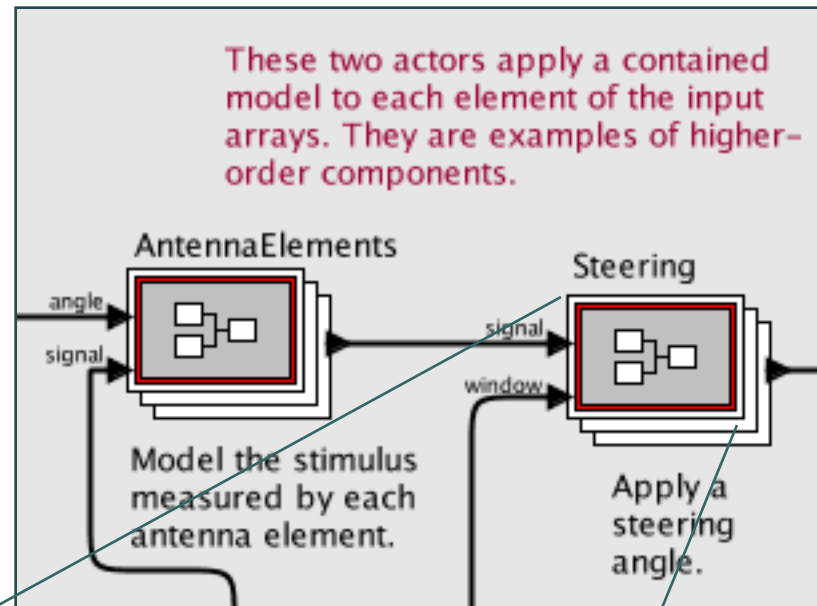
    // random number with a Gaussian distribution to the output
}
```

Key Prior Work from the Ptolemy Project:

1. Higher-Order Components

Examples of HoCs:

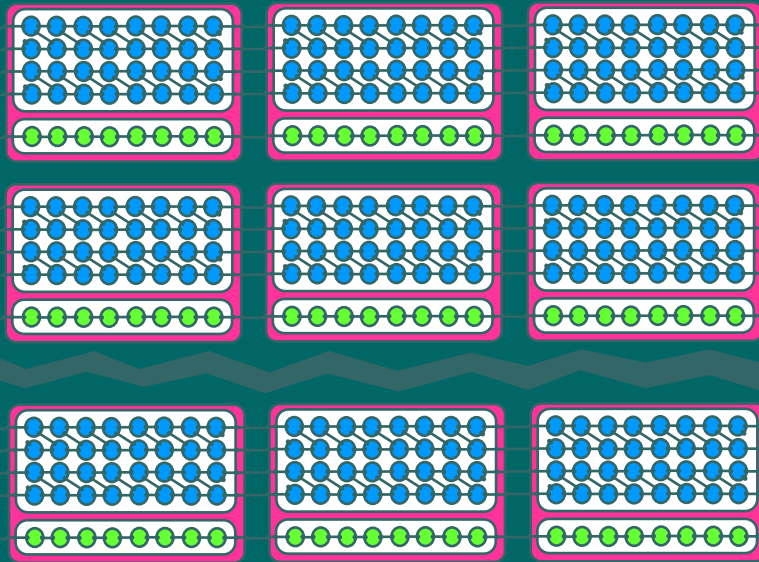
- Replicate a submodel over an array of inputs
- Structured dataflow components (case, iterate, recursion)
- Mobile models
- Parameterizing models with models
- Lifecycle models



Key Prior Work from the Ptolemy Project:

2. Composition Languages

Big Systems with Small Descriptions



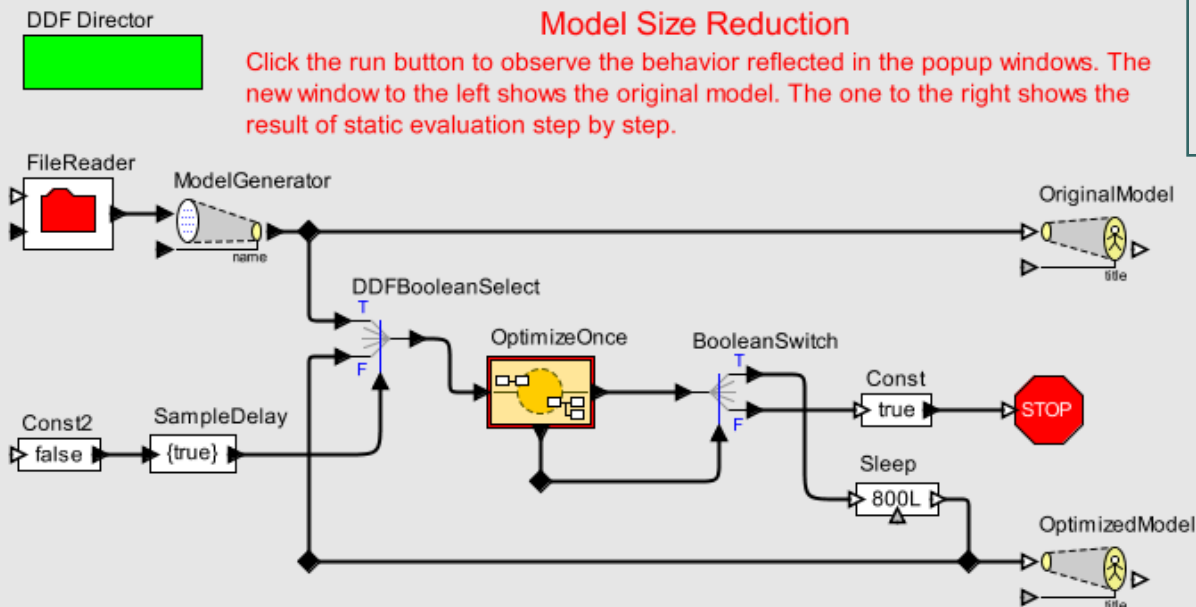
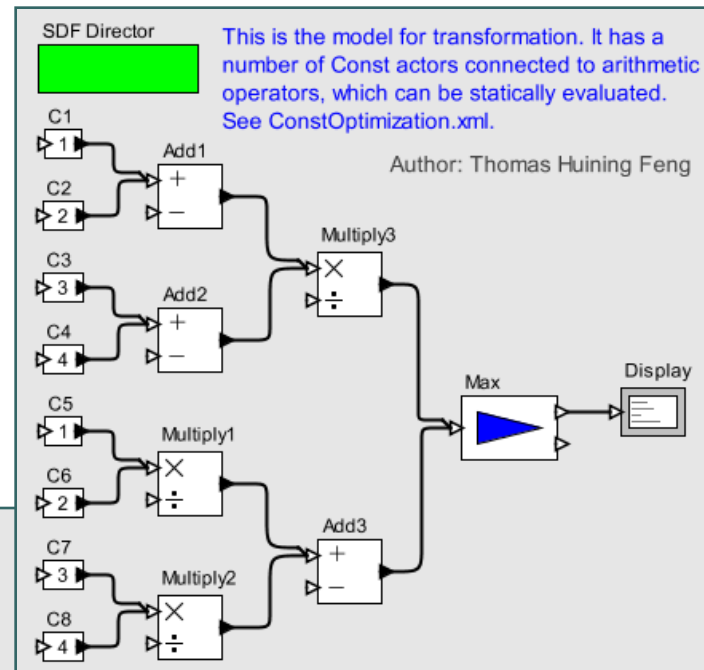
We have released a specification language that we call “Ptalon” for such systems, integrated into Ptolemy II [Cataldo 2006]

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

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

Demo: Pattern Matching and Graph Transformation

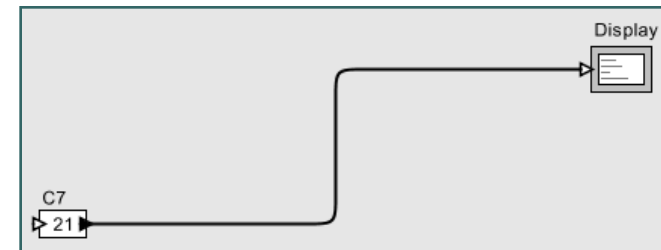
Model transformation workflow specifies iterative graph rewriting to transform the top-right model into the bottom-left model.

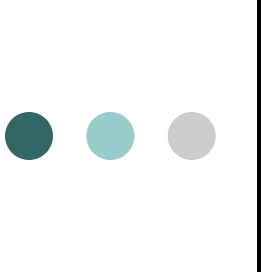


This model demonstrates how one can possibly optimize a model. The original input is the model in BaseModel.xml, which the FileReader actor reads in. The contents of this model are then converted into an ActorToken by the ModelGenerator. OptimizeOnce is a transformation rule that gets repeatedly applied to this model until no further optimization is possible (i.e., a fixpoint is reached). In each application, two Consts that are wired to an AddSubtract actor, a MultiplyDivide actor, or a Maximum actor are replaced by a single Const with the statically computed value.

Author: Thomas Huining Feng (Inspired by Thomas Mandl)

Executing the model at the left transforms the top model into the bottom model.





Applications

- *Model optimization*

 - Support programming idioms*

shown

- *Scalable model construction*

 - Adapt to problem size or parallelism*

next

- *Product families*

 - A single model transforms to multiple products*

- *Design refactoring*

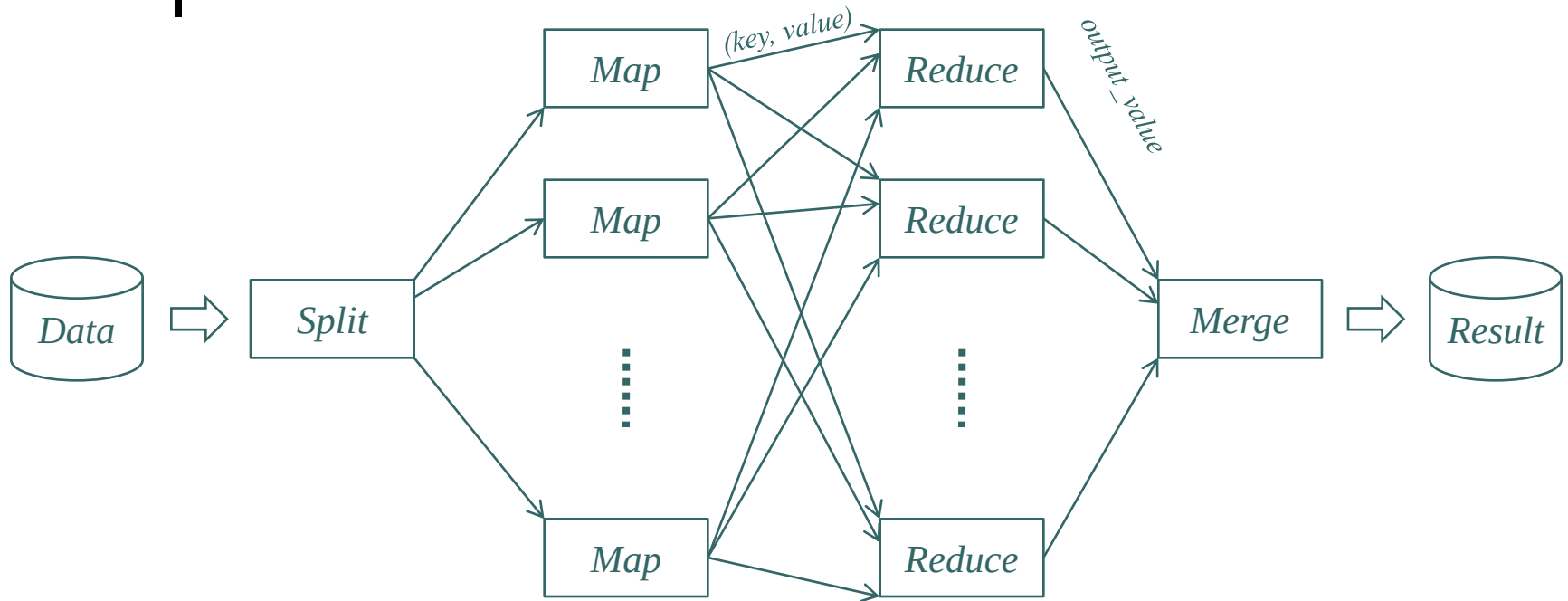
 - Common model transformations*

- *Workflow automation*

 - Configuration, composition, testing, versioning*

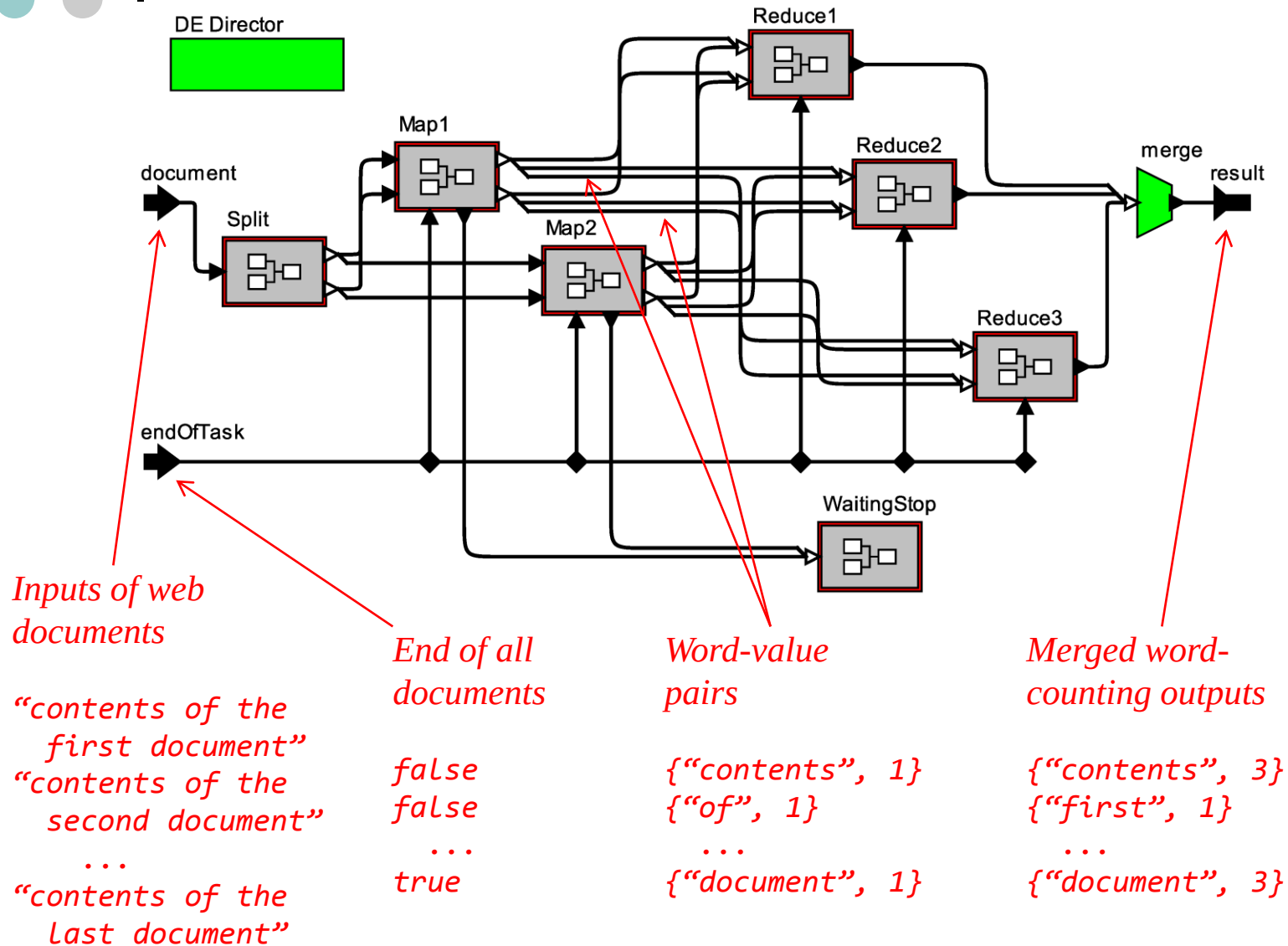
Scalable Model Construction:

MapReduce Pattern [Dean, Ghemawat, 2004]

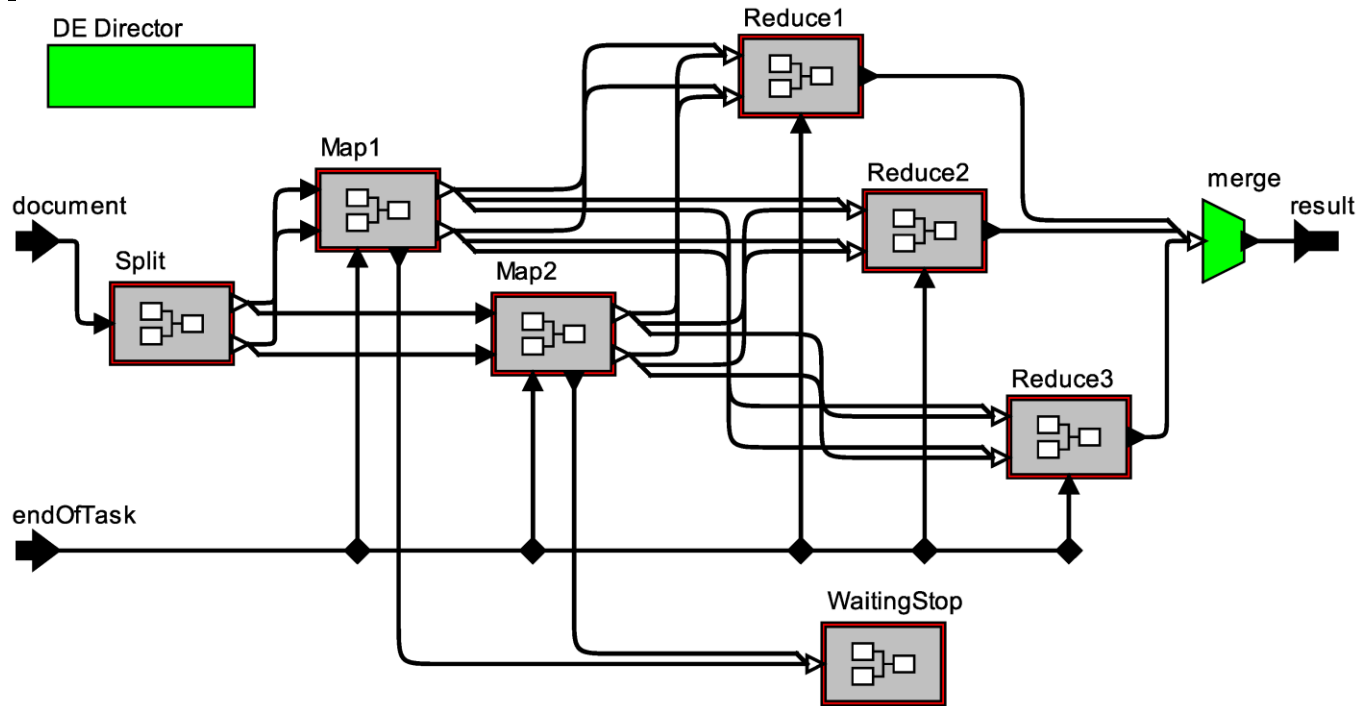


This pattern is intended to exploit parallel computing by distributing computations that fit the structure. The canonical example constructs an index of words found in a set of documents.

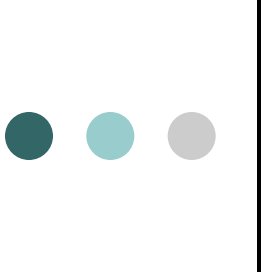
A MapReduce Model in Ptolemy II



Complexity of the Model Structure



A configurable application with m Map actors and n Reduce actors has $O(m \times n)$ connections. Inside each actor, parameters need to be configured as well.

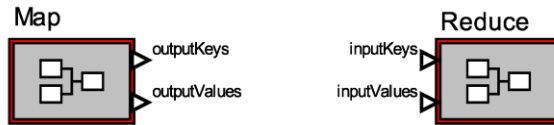


Observations

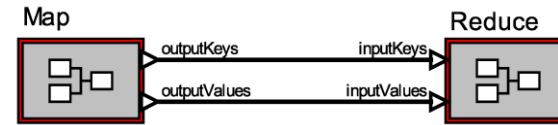
- The visual representation is only helpful for small models.
- The construction process by visual editing is tedious and error prone.
- Adapting the size of the model to varying numbers of compute resources by visual editing is unreasonable.
- Replacing the Map or Reduce actors with alternative functions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

Transformation Rule

Pattern



Replacement



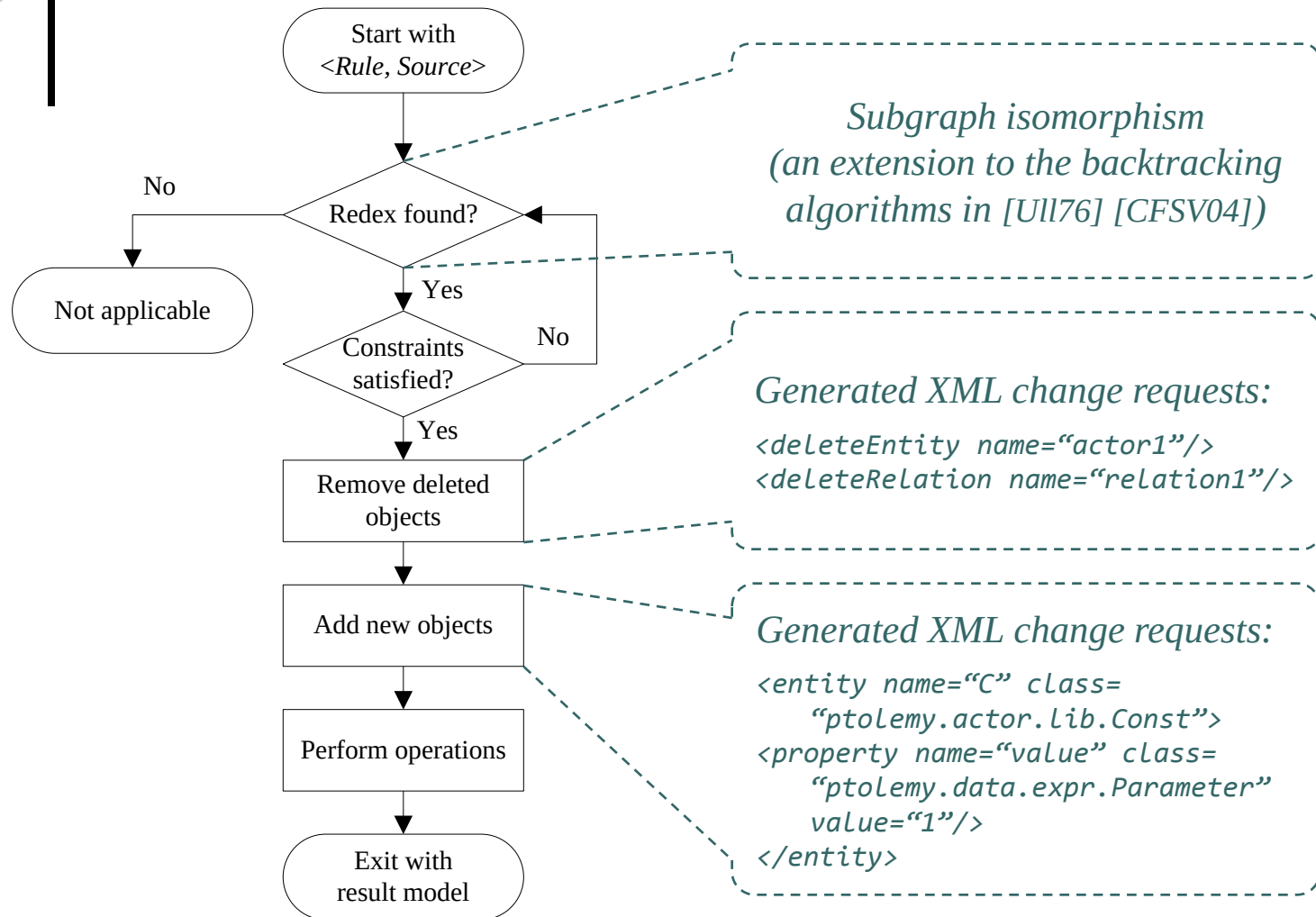
*Constraint: !Map.outputKeys.connectedPortList()
.contains(Reduce.inputKeys)*

Pattern	Replacement
Map	Map
Reduce	Reduce

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

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

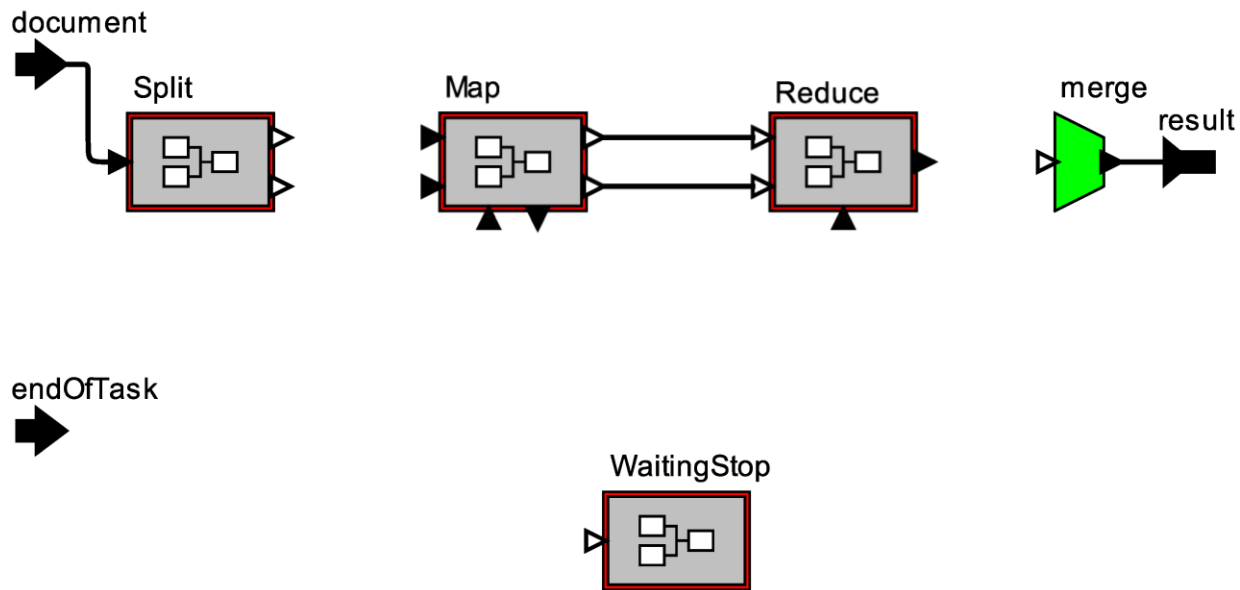
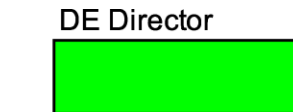
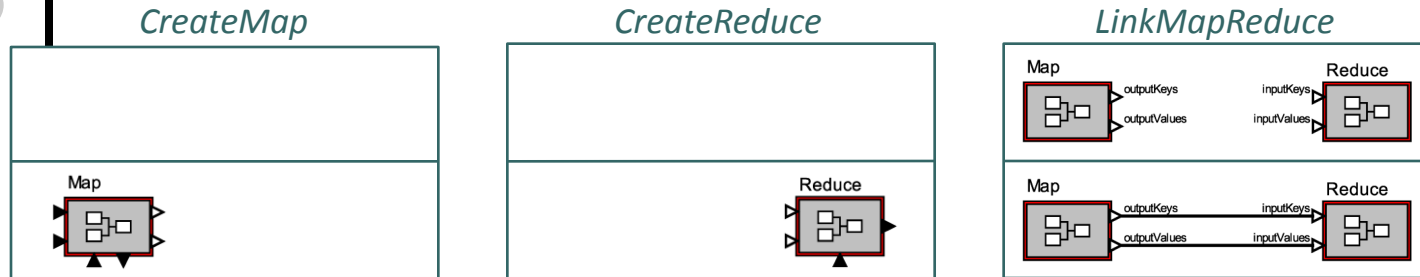
Rule Application



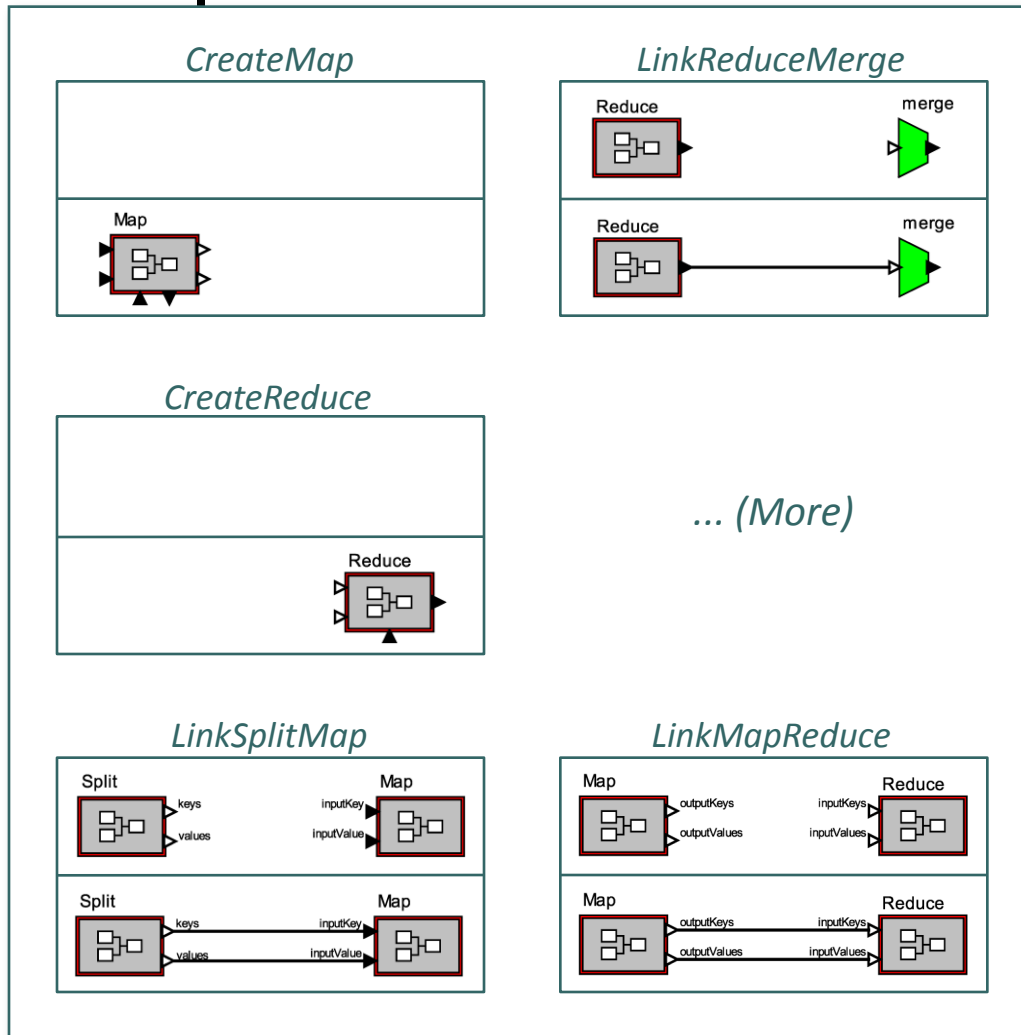
[Ull76] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1), 1976.

[CFSV04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), 2004.

A Set of Transformation Rules



Model is built by repeatedly applying transformation rules drawn from a set



Workflow mechanisms:

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

The TransformationRule Actor

Encapsulates a transformation rule

Input:

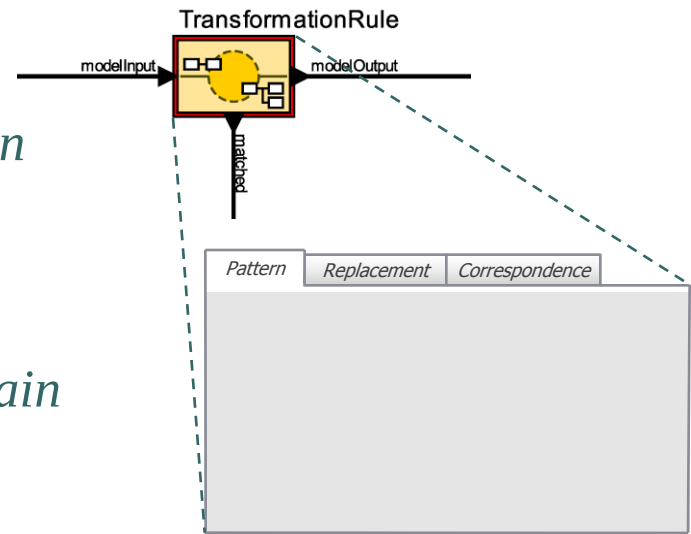
modelInput – actor tokens that contain model fragments

Output:

modelOutput – actor tokens that contain transformation results

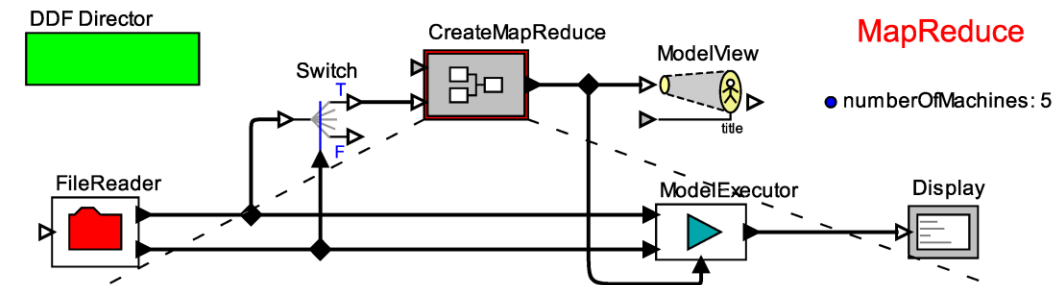
matched – whether the last transformation was successful

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

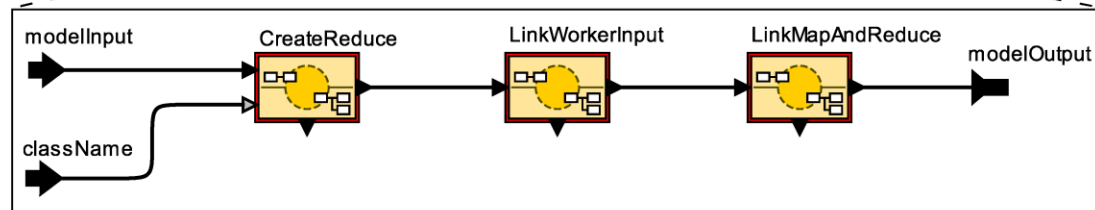
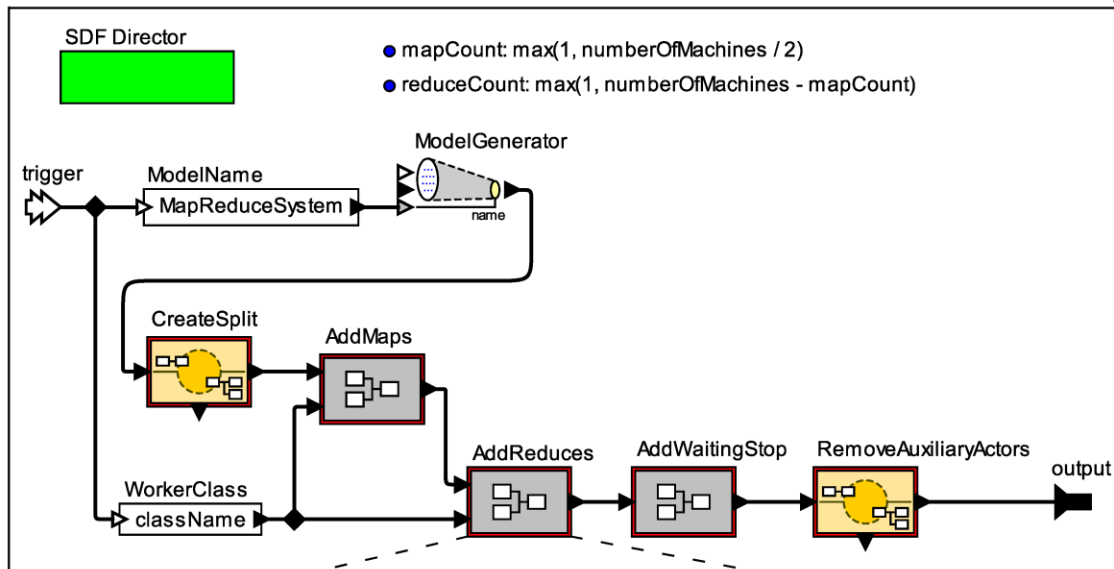


Actor provides a custom visual interface for specifying model transformations

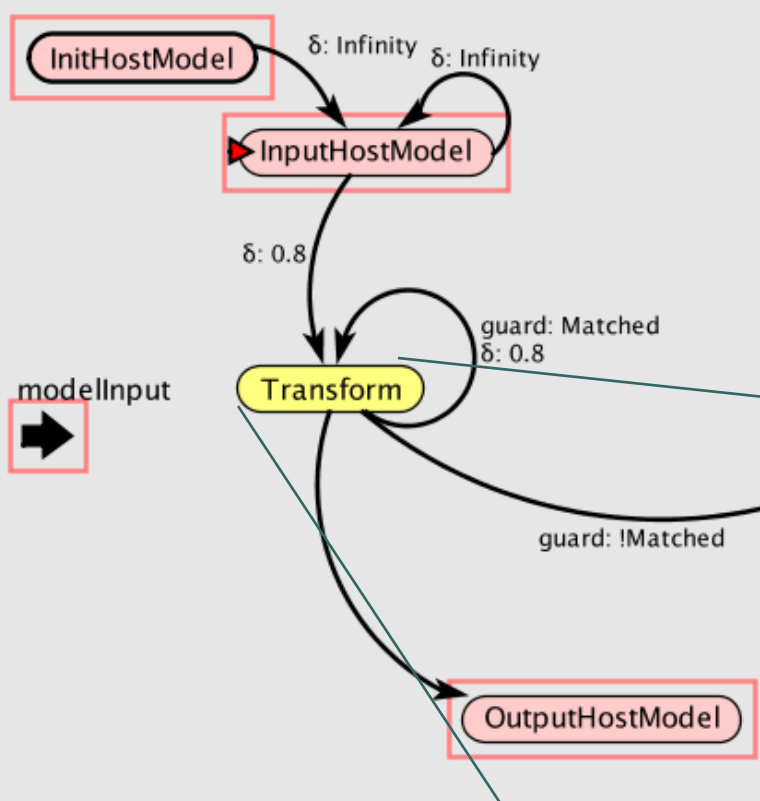
Ptolemy II Model Defines a Workflow



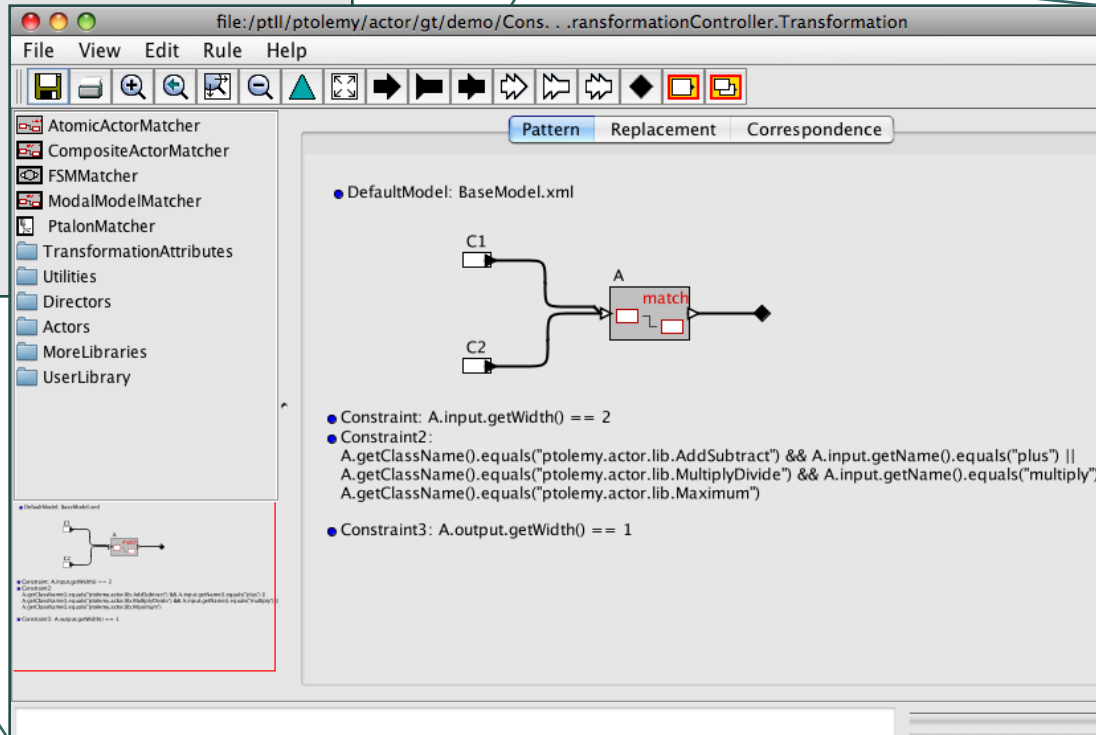
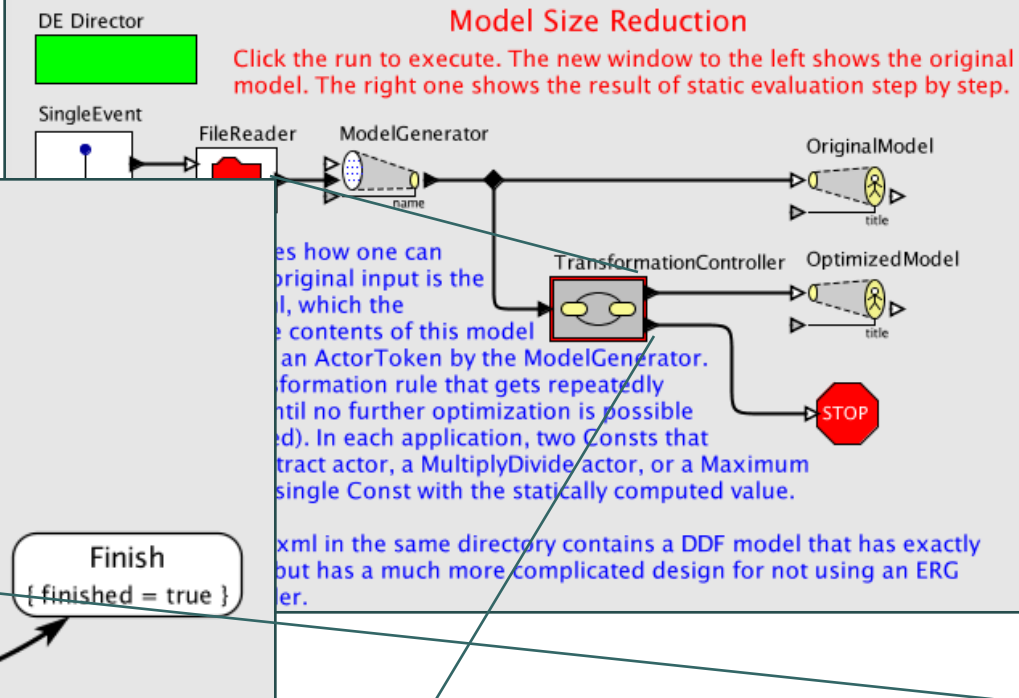
This Ptolemy II model creates and executes a MapReduce application for a parameterized number of machines.

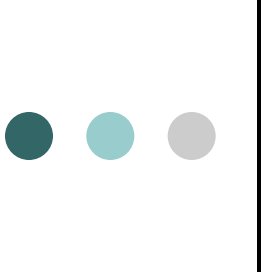


Using Other MoCs



Here we have used Event-Relationship graphs [Schruben 83] to specify the transformation logic.





Summary

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