Applying an Object-Oriented Petri Net Language to Heterogeneous Systems Design

Robert Esser
Jörn W. Janneck
Martin Naedele

Overview

- Objectives
- Heterogeneous Systems
- Requirements
- Basic Time Petri Net Model
- Object-Oriented Time Petri Nets
- Supporting other Formalisms
- Containment & Inheritance
- Experiences from a Case Study
- Conclusion

The Moses Project: Objectives

- modeling of complex heterogeneous systems
- simulation and validation/evaluation environment
- visualizing systems structures and behavior
- experimentation platform for
  - verification algorithms
  - simulation techniques
  - embedding of other formalisms (than Petri nets)

Heterogeneous Systems

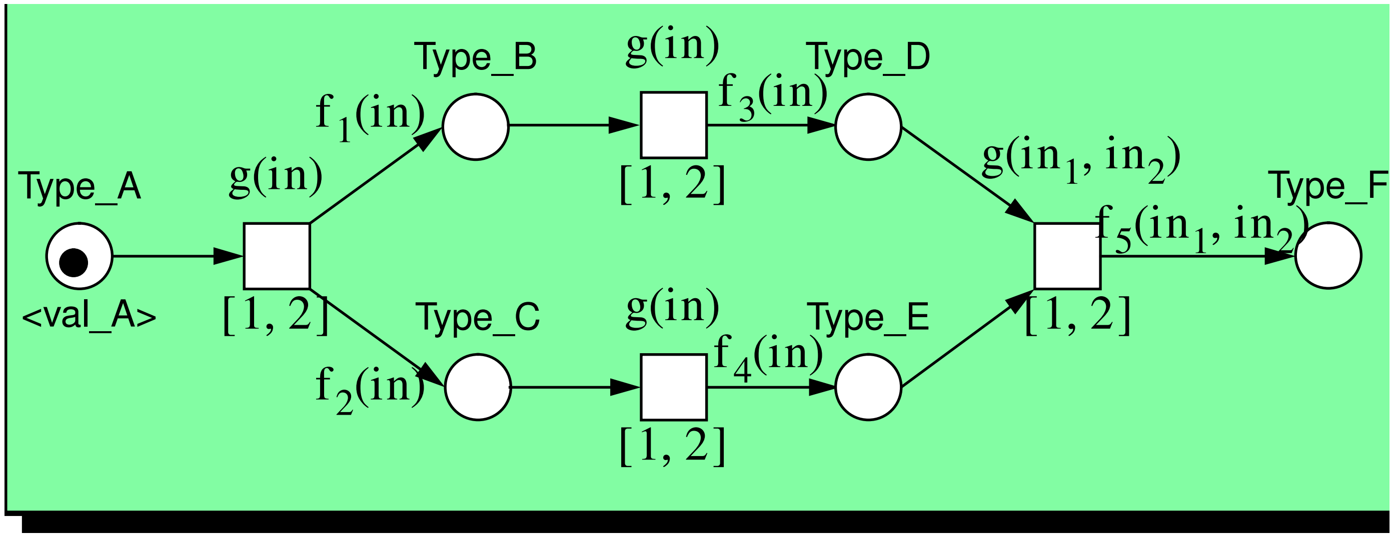
- ... cover a variety of model domains
  - data flow, material flow, mechanics, continuous physical systems, electronics, optics, logistics, ...
- ... are composed of parts with very different characteristics
  - control-oriented *vs* flow-oriented, concurrent *vs* sequential, discrete *vs* continuous, ...
- ... can be pretty complex in the real world

Requirements

- support for multiple formalisms
- hierarchical/compositional modeling concepts
- executable models (simulation, validation, evaluation)
- formal semantics (verification)
- common formal base description
- graphical representation (where appropriate)
- configurable modeling concepts
- reuse of modeling components

Basic Time Petri Net Model

- tokens carry data values
- transitions are labeled with firing intervals, guards, and functions
- transitions may be enabled more than once

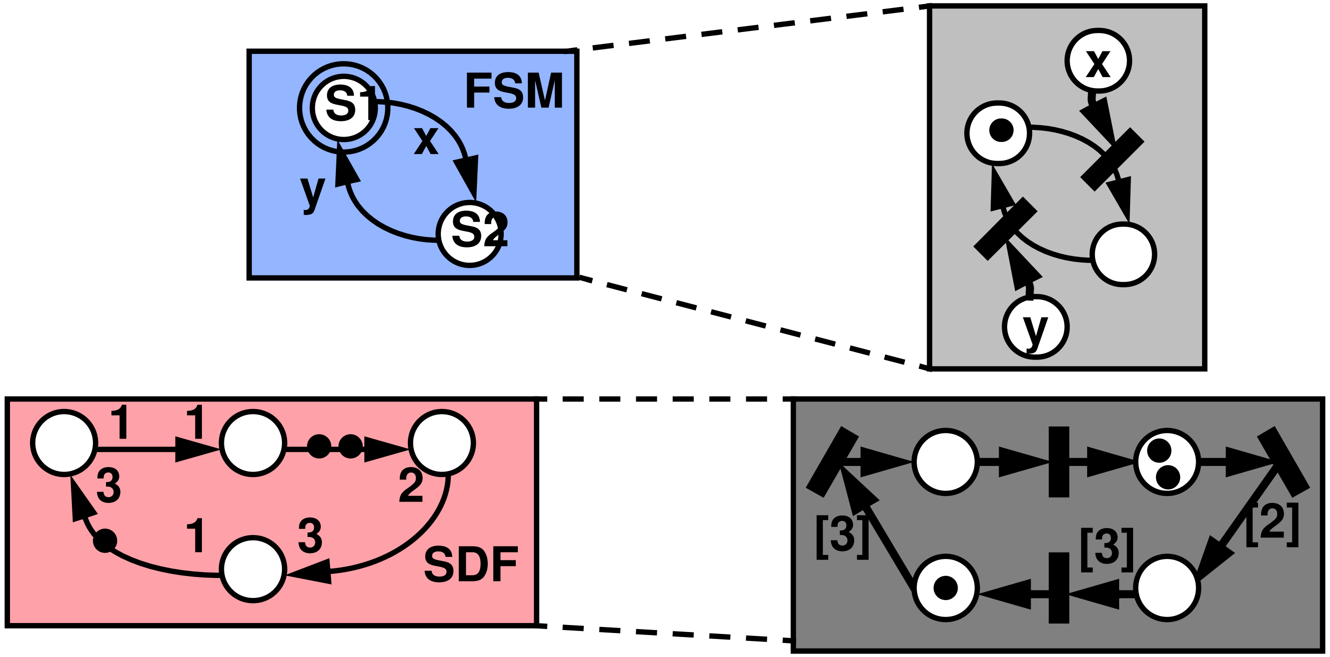


Object-Oriented Time Petri Nets

- one simple concept: ‘Everything is an object’
- objects have fields (slots, instance variables) which are again objects
- classes are object templates containing Interfaces, functions and a Petri net
- a system is a hierarchical composition of objects/components
- a system is configured (via functions) during system instantiation

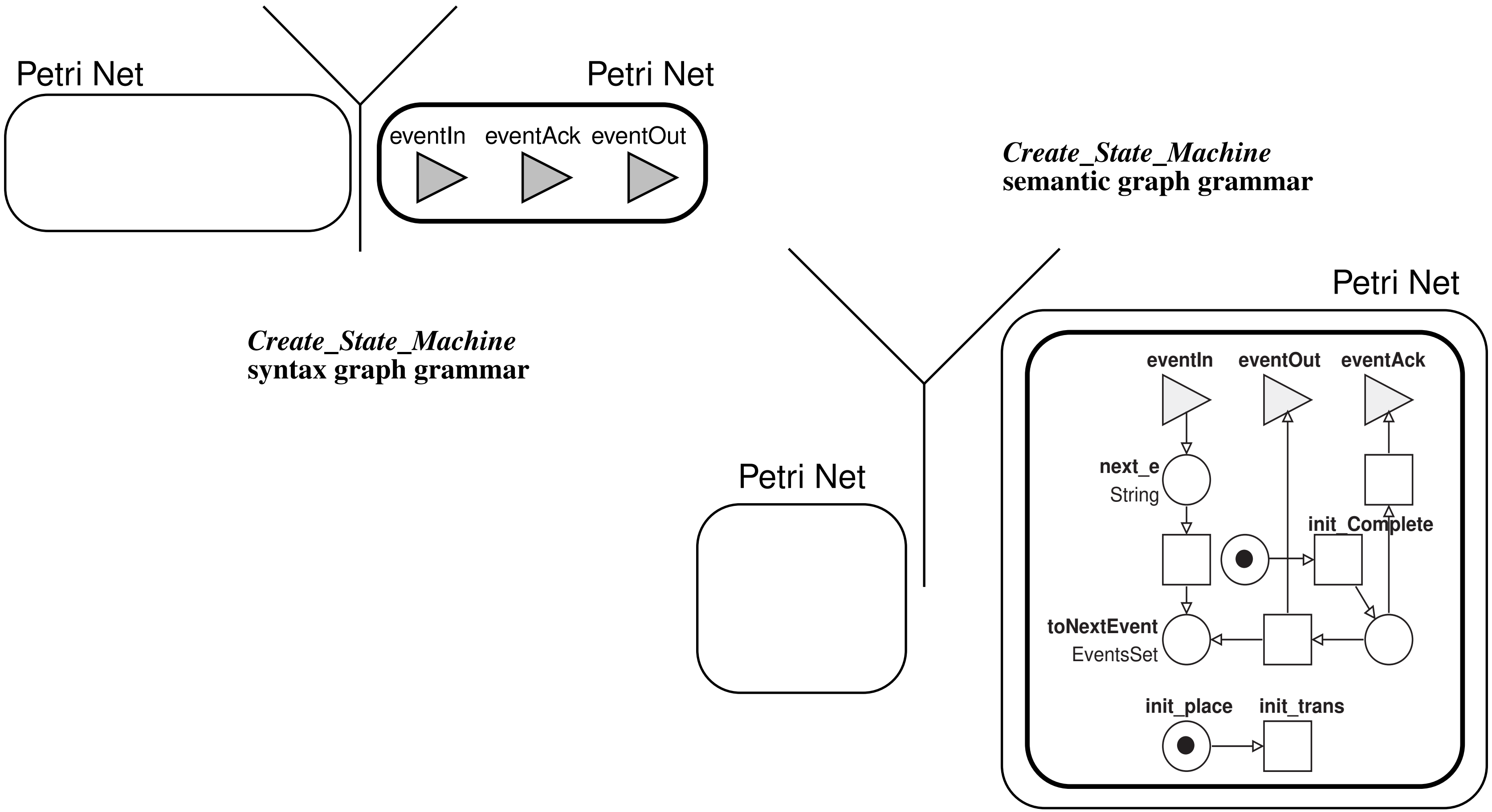
Supporting other Formalisms

- ... by translating them to time Petri nets

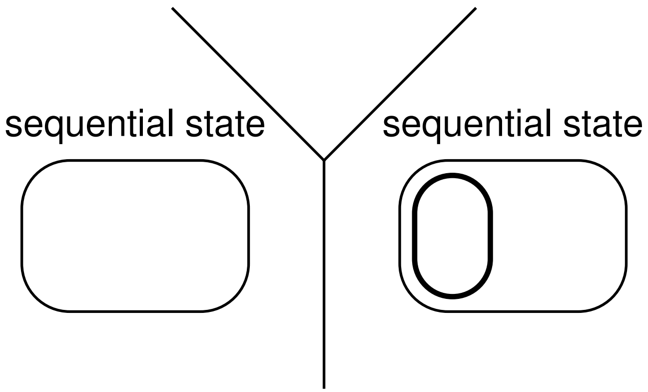


- ... and thus seamlessly embedding components specified with them

Supporting other Formalisms

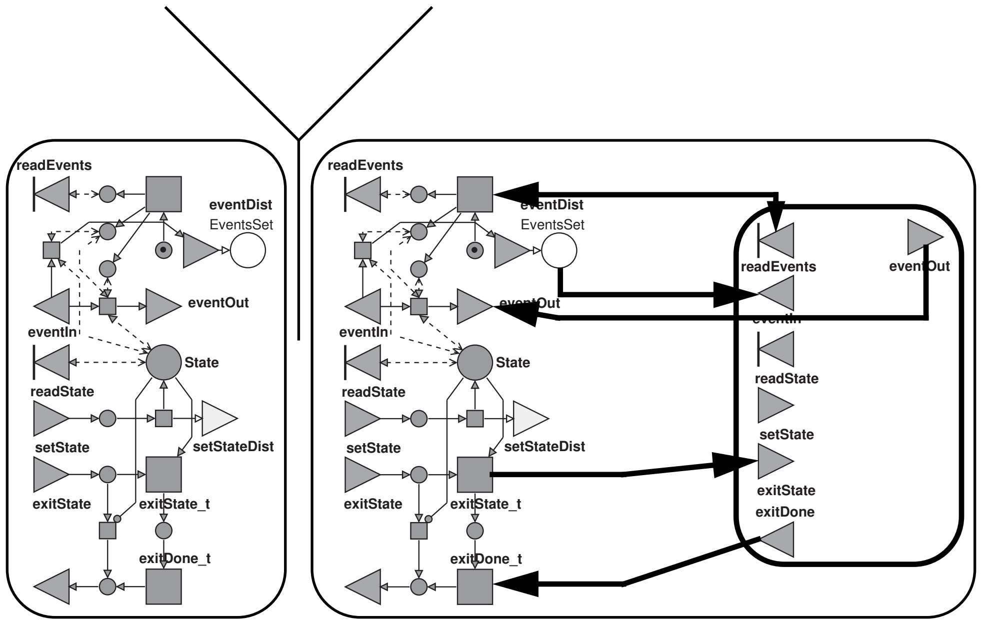


Supporting other Formalisms

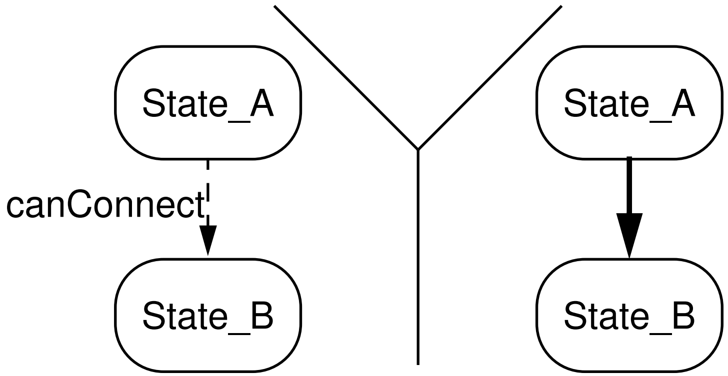


Insert_State_Into_Sequential
syntax graph grammar

Insert_State_Into_Sequential
semantic graph grammar

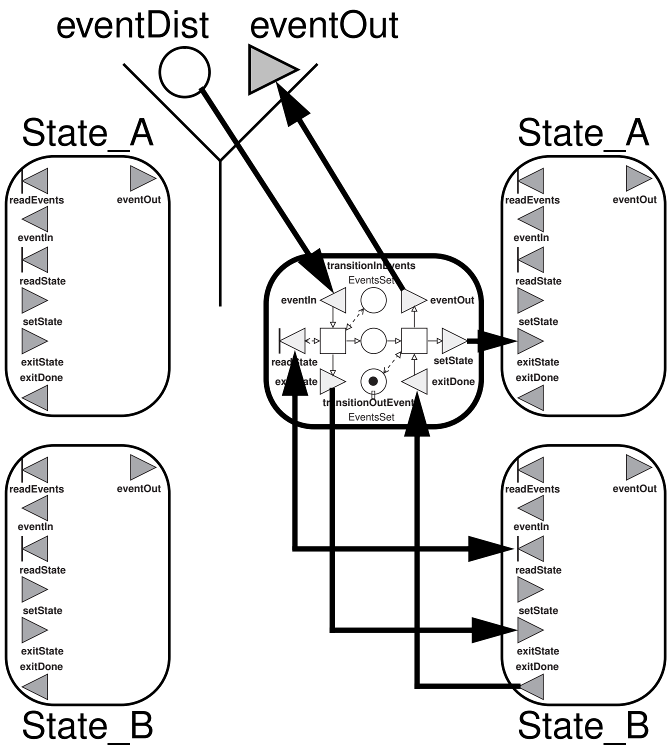


Supporting other Formalisms



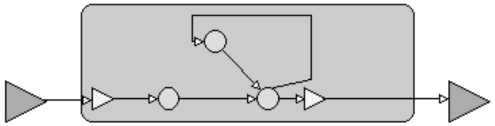
Insert_Transition
syntax graph grammar

Insert_Transition
semantic graph grammar

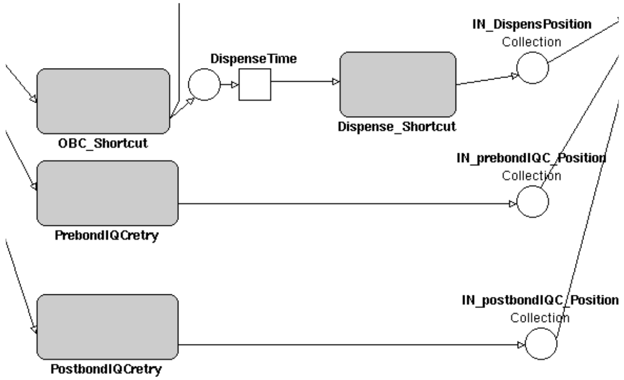


Embedding Components

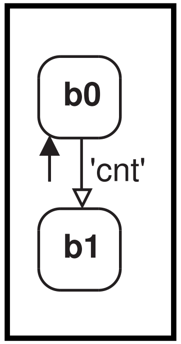
- ... components may be formulated in one of several formalisms (e.g. Petri nets, finite state machines, SDF)



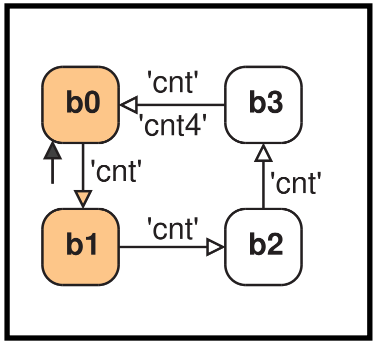
- ... and be embedded into other components



Containment & Inheritance

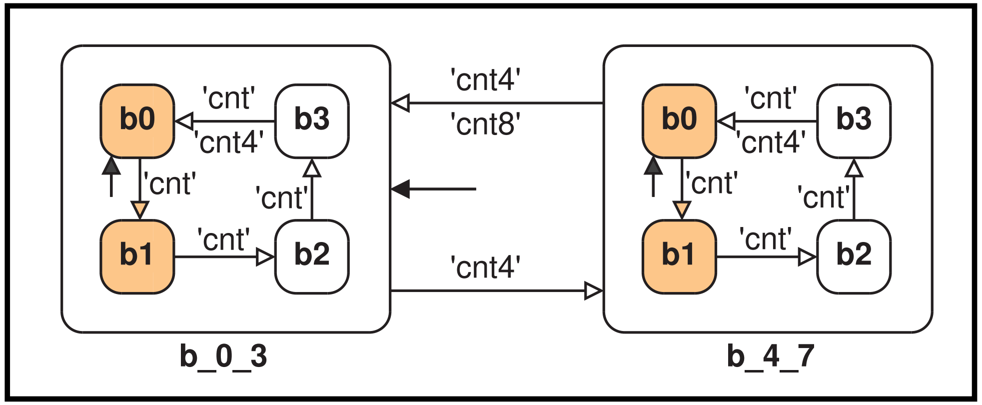


Class *Counter_2*

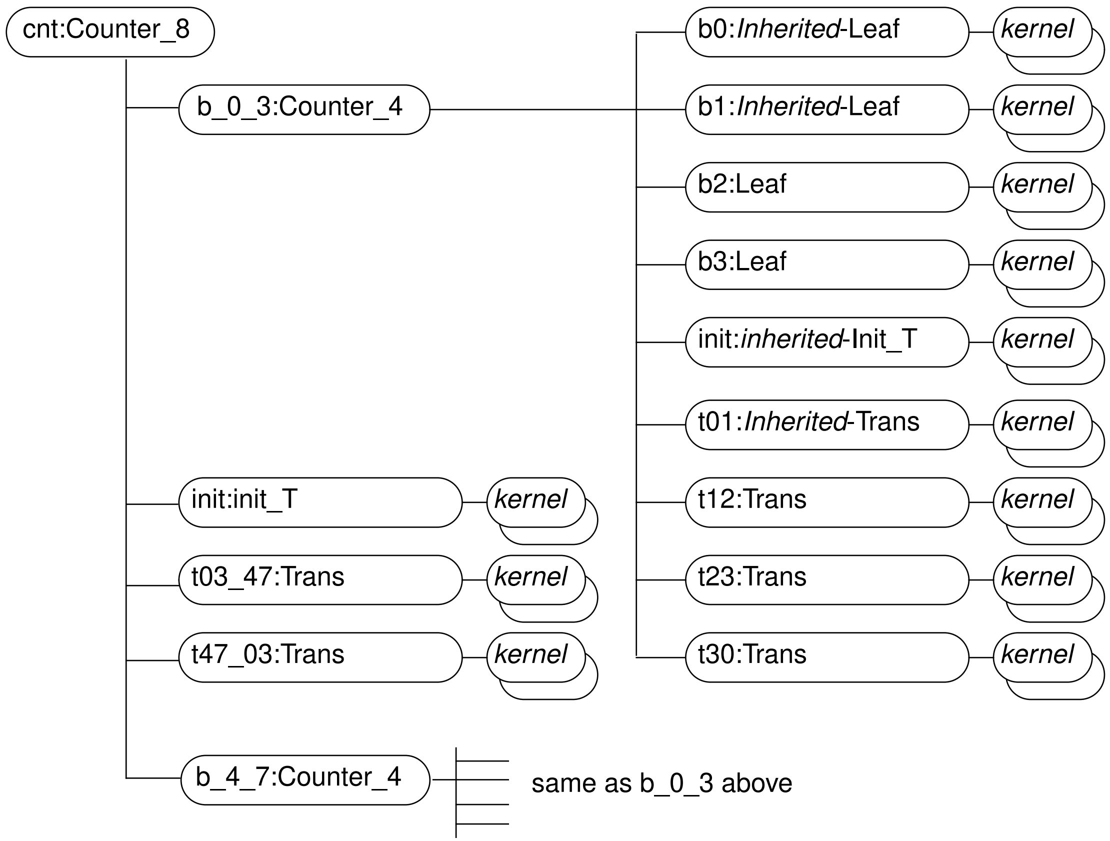


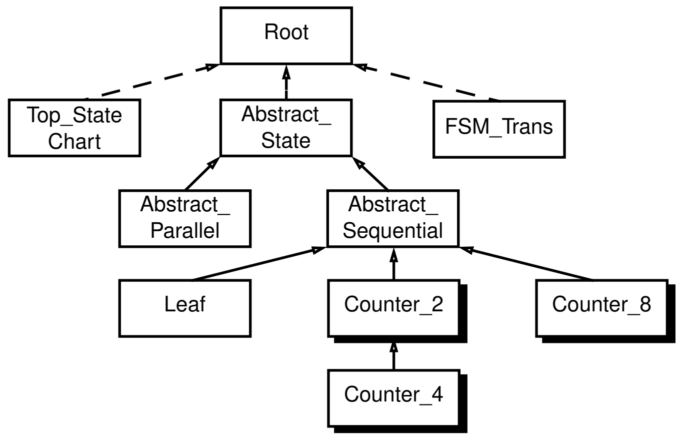
Class *Counter_4*

Class *Counter_8*

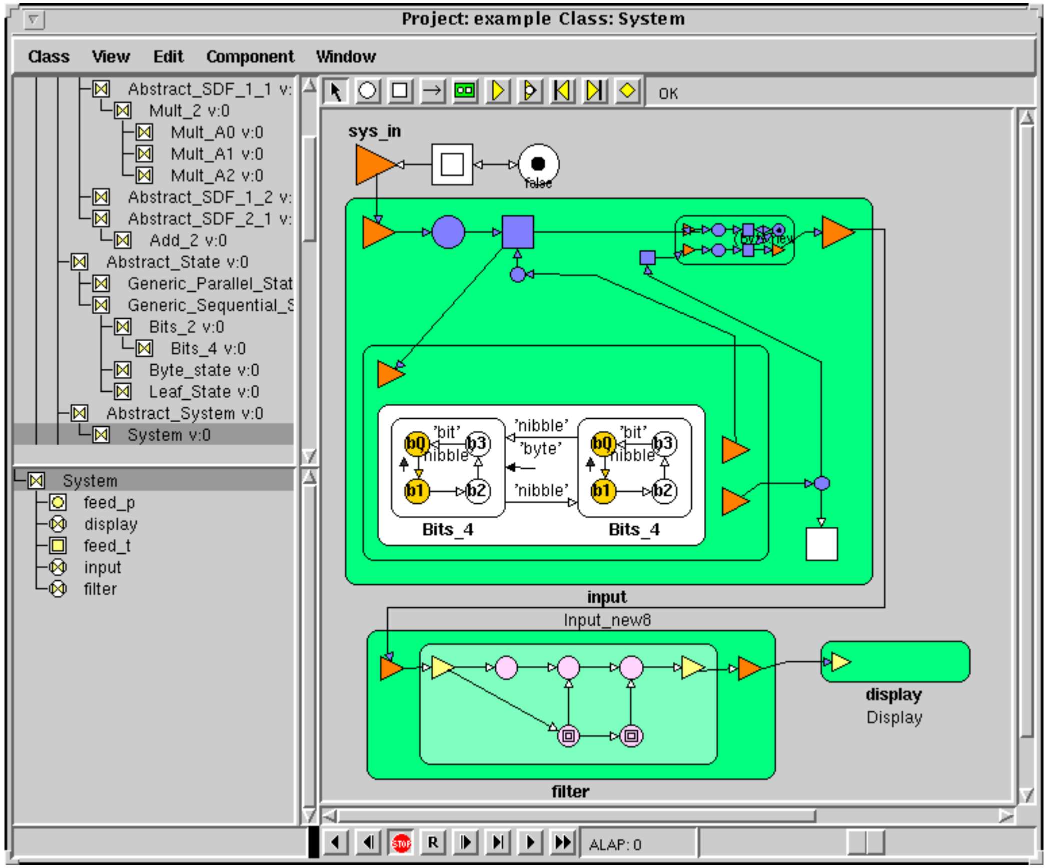


Containment & Inheritance



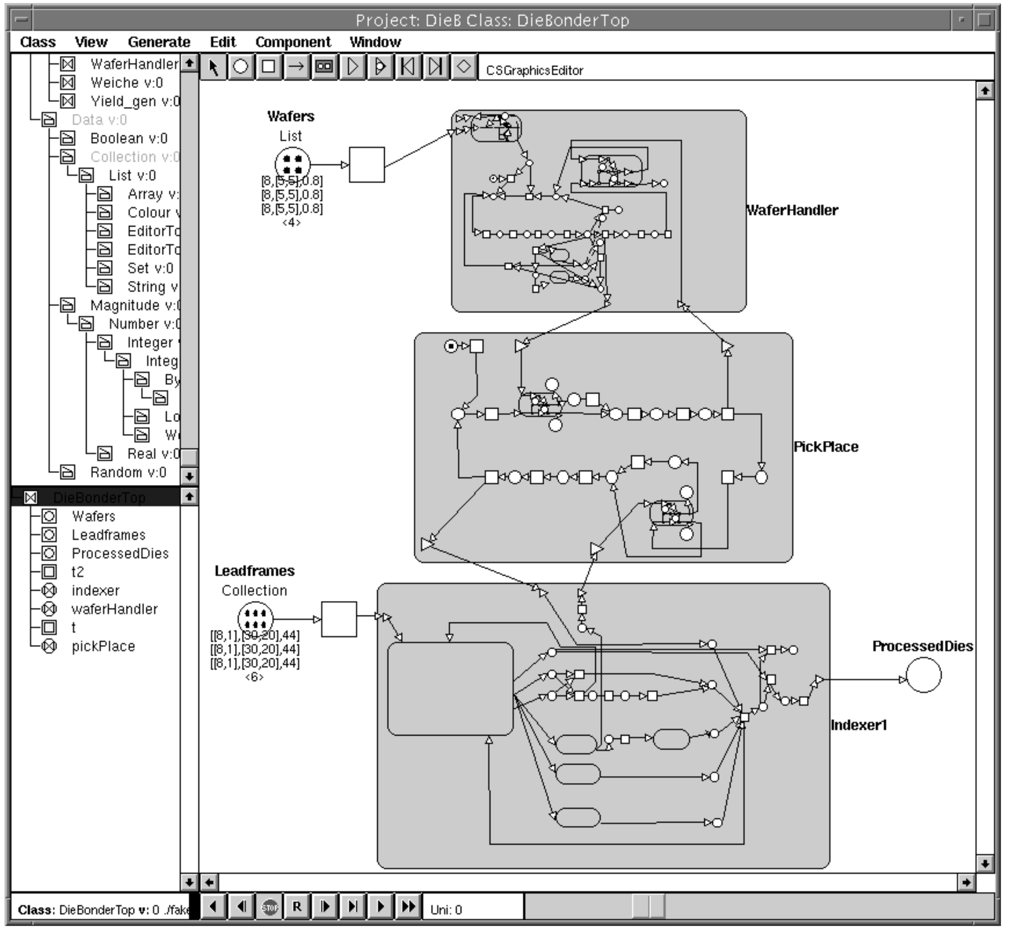


The CodeSign Tool



http://www.tik.ee.ethz.ch/~codesign/

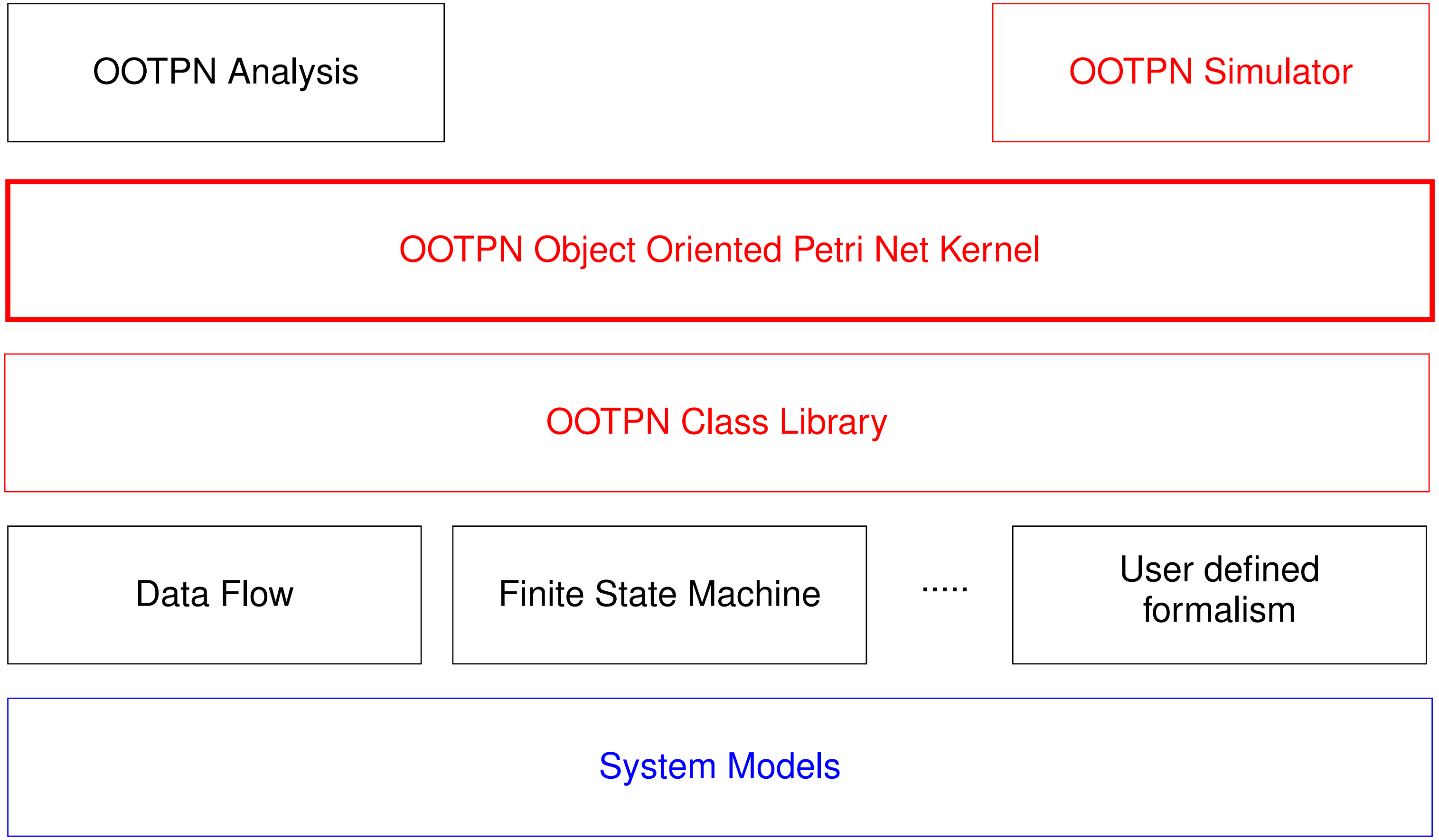
The Case Study



Experiences

- modeling/visualization
  - increased understandibility
  - multiple formalisms useful for design and communication
  - abstraction, hierarchy, refinement help conquer complexity
- simulation/validation/evaluation
  - parameterization/configuration used in exploring the design space
  - more extensive statistics needed

Conclusion


OOTPN Analysis
OOTPN Simulator
OOTPN Object Oriented Petri Net Kernel
OOTPN Class Library
Data Flow
Finite State Machine
.....
User defined formalism
System Models