Modeling and Verification of Connectors in Complex Systems

Sun Meng

LMAM & Department of Information Science, School of Mathematical Sciences
Peking University
http://www.math.pku.edu.cn/teachers/sunm

Thanks to: B. K. Aichernig (TUG), F. Arbab (CWI), L. Aştefănoaei (INRIA), C. Baier (TUD), L. Barbosa (UM), F. de Boer (CWI), T. Chothia (Birmingham), N. Kokash (CWI), M. Kwiatkowska (Oxford), Y. Li (PKU), Y.-J. Moon (INRIA), H. Qu (Oxford), J. Rutten (CWI), R. van der Mei (VUA), C. Verhoef (CWI)

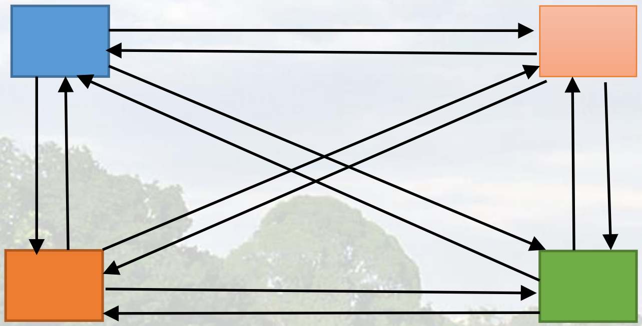
Workshop on Probabilistic and Hybrid System Verification, Beijing, September 26, 2013

Outline

- Coordination in complex systems
- Reo and Eclipse Coordination Tools
- Synthesis of connectors from BPMN / UML models
- Verification and Performance Analysis for connectors
- Conclusion and future work

Sources of Complexity in Systems

- Complexity inherent in task/algorithm/computation
  - Examples:
    - Computations/equations in quantum mechanics, astronomy, engineering, etc.
    - Bit-map to jpeg conversion, sorting, etc.
  - This type of complexity is not bewildering!
    - Many good, intricate mathematical models have been developed to tame the complexity.
- Complexity arising from composition of simple components
  - Example:



  - Bewildering complexity emerges out of **interaction**
  - Good formal models to tame this complexity?

Models of Concurrency

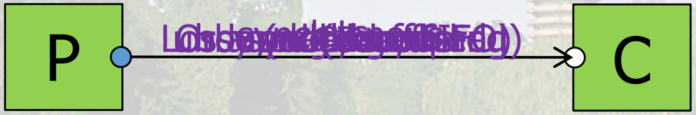
- Traditional models are action based
  - Petri nets
  - Work flow / Data flow
  - Process algebra / calculi
  - Actor models / Agents
  - …
- In prominent models, a system is composed from building blocks that represent actions/processes
- Interaction becomes an implicit side-effect
  - Makes coordination of interactions more difficult to
    - Specify
    - Verify
    - Manipulate
    - Reuse

Interaction Based Concurrency

- Start with a set of primitive interactions as binary constraints
- Define (constraint) composition operators to combine interactions into more complex interactions
- Properties of the resulting model of concurrency depend on
  - Set of primitive interactions
  - Composition operators
- As constraints, interaction protocols can be manifested independently of the processes that they engage
  - **Connectors**
- Imposing an interaction on actors exogenously coordinates their activities

Exogenous Coordination

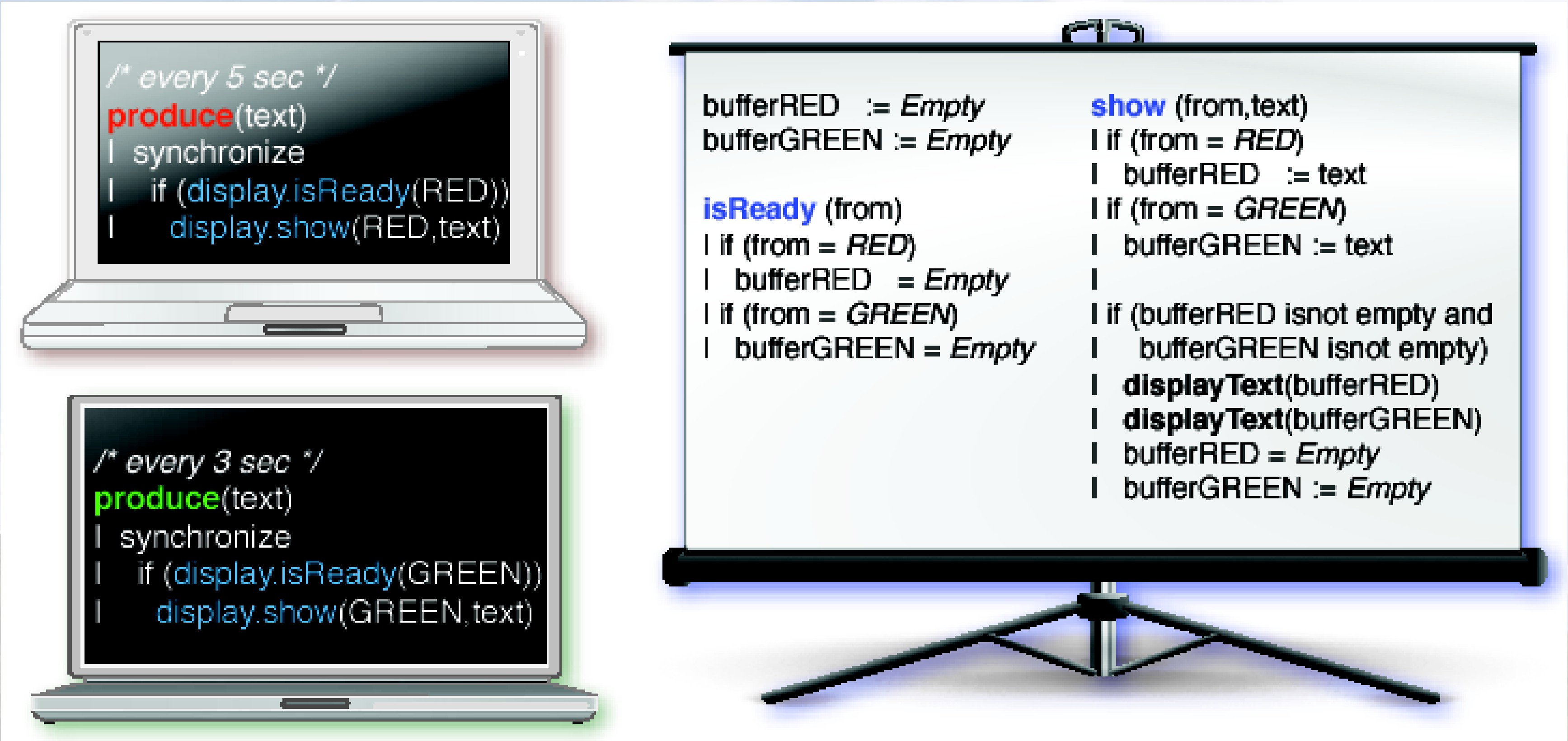
- P and C are black-box components that:
  - Offer no inter-components methods nor make such calls
  - Do not send/receive targeted messages
  - Their only means of communication is through blocking I/O primitives that they can perform on their own ports.
  - Composing P and C with different connectors (that impose different protocols from outside) constructs different systems.



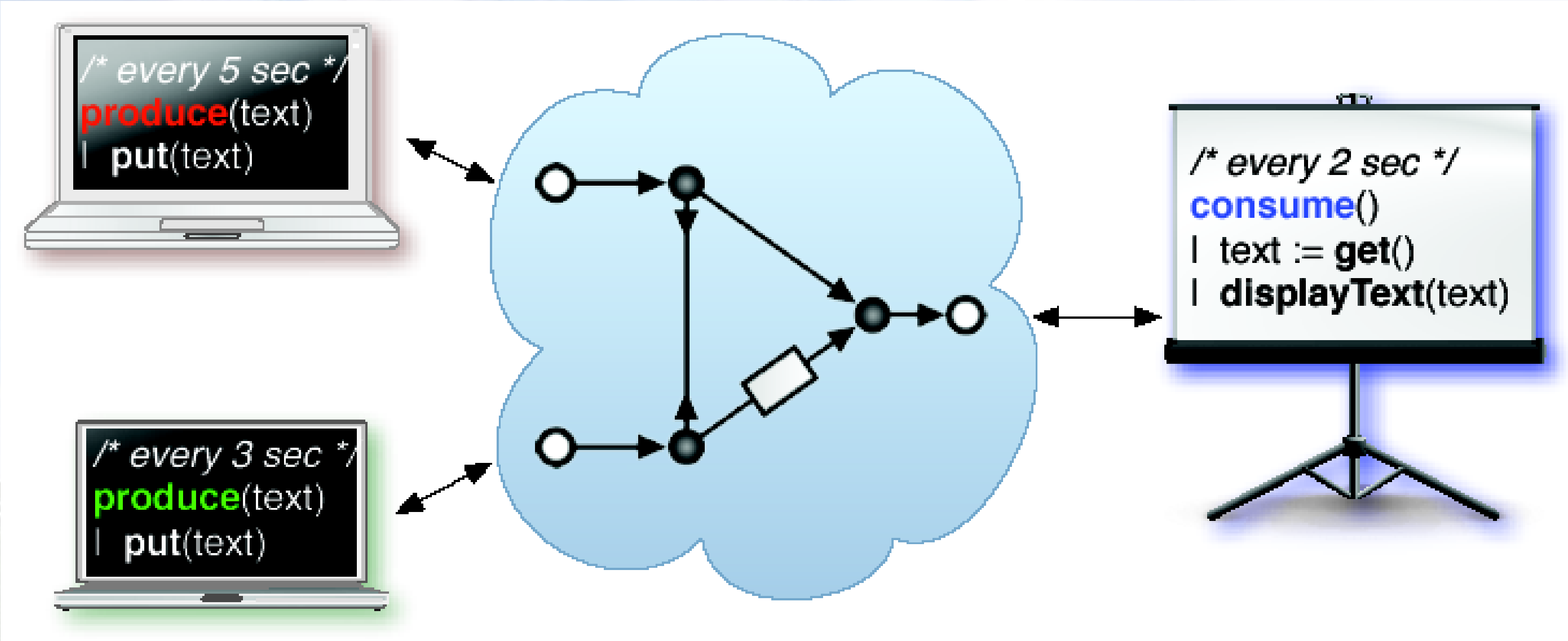
Reo: An Exogenous Coordination Language

- Reo is an exogenous coordination language for compositional construction of interaction protocols.
- Interaction is the only first-class concept in Reo:
  - Explicit constructs representing interaction
  - Composition operators over interaction constructs
- A (coordination or interaction) protocol:
  - manifests as a connector
  - gets imposed on its engaged components/services from outside
  - remains mutually oblivious to its engaged components/services
- Reo offers:
  - Loose(st) coupling
  - Arbitrary mix of asynchrony, synchrony, and exclusion
  - Open-ended user-defined primitive channels
  - Distribution and mobility
  - Dynamically reconfigurable connectors
- http://reo.project.cwi.nl

Reo: A Coordination Language



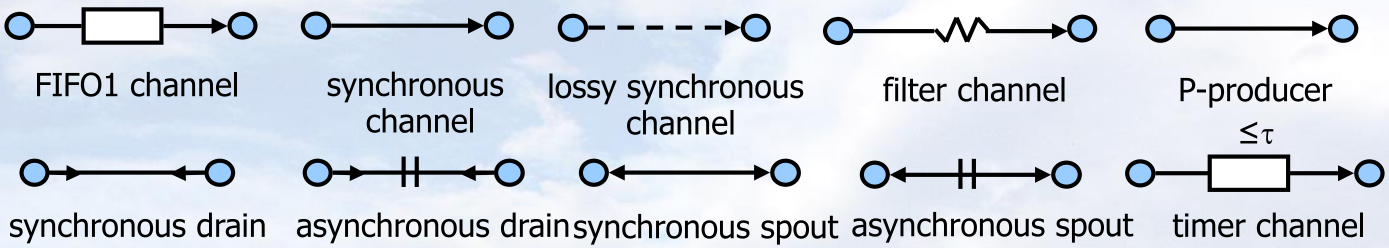
Reo: A Coordination Language

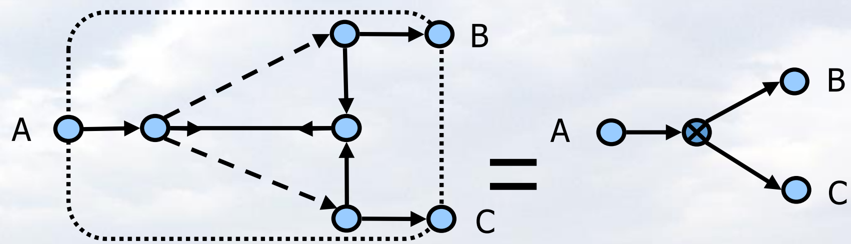


Channels

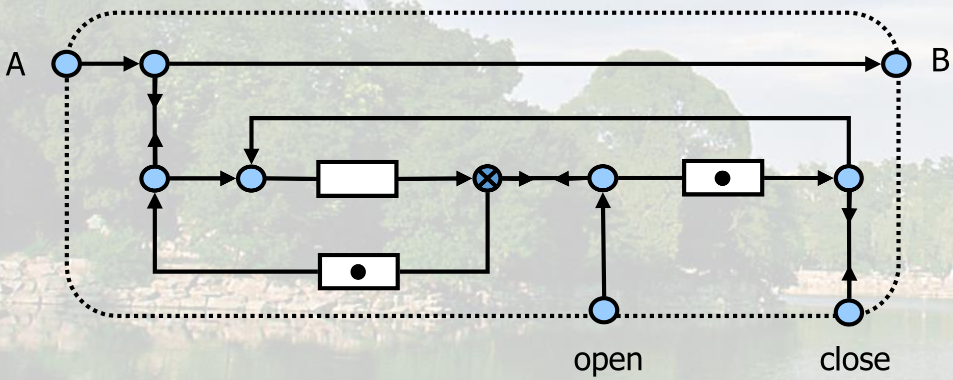
- Atomic connectors in Reo are called *channels*.
- Reo generalizes the common notion of channel.
- A channel is an abstract communication medium with:
  - exactly two ends; and
  - a constraint that relates (the flows of data at) its ends.
- Two types of channel ends
  - Source: data enters into the channel.
  - Sink: data leaves the channel.
- A channel can have two sources or two sinks.
- A channel represents a primitive interaction.

Reo Connectors





Exclusive choice (deffered XOR)



Valve connector:
controls flow from A to B

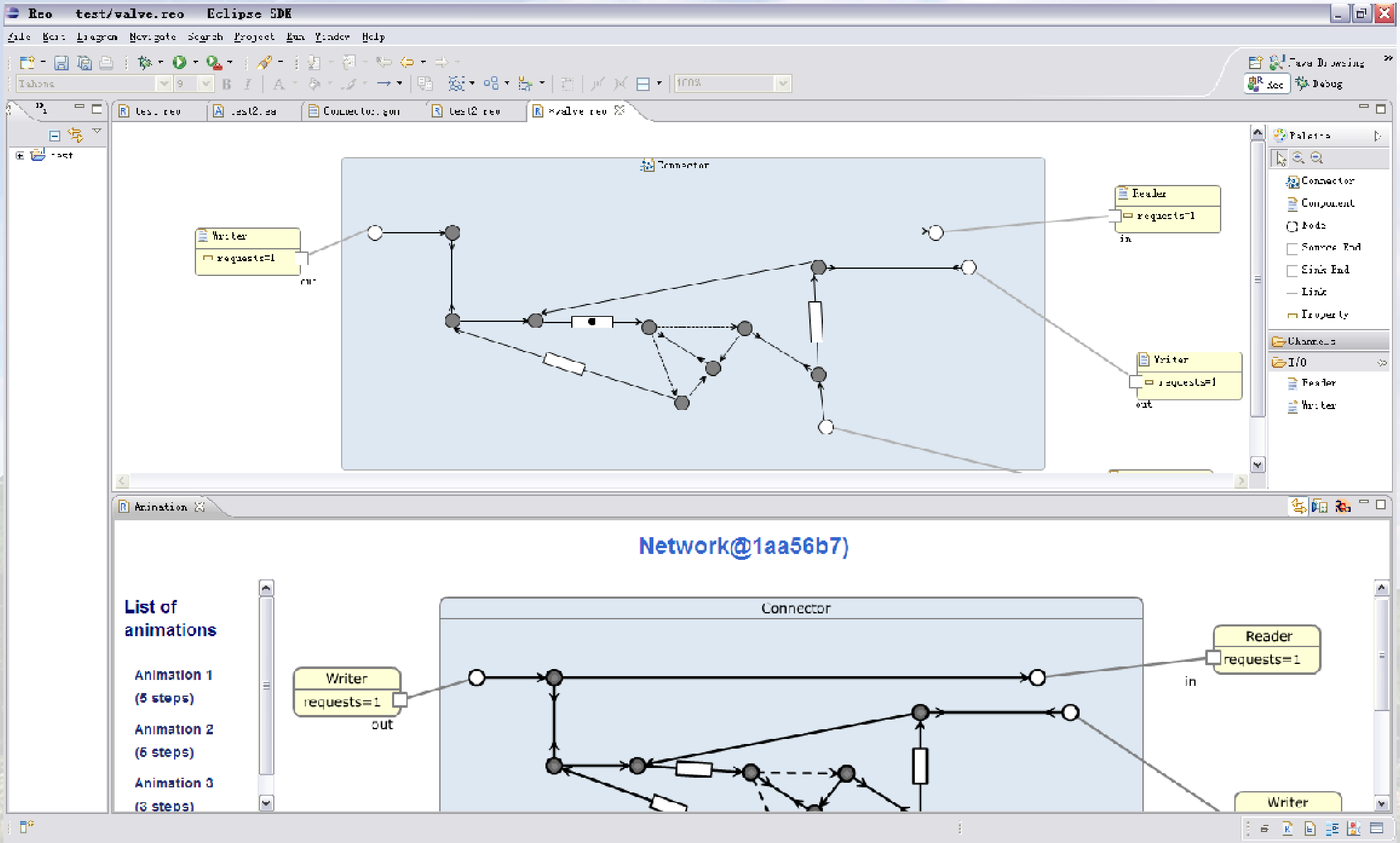
Eclipse Coordination Tools

- A set of Eclipse plug-ins provide the ECT visual programming environment.
- Protocols can be designed by composing Reo circuits in a graphical editor.
- The Reo circuit can be animated in ECT.
- ECT can automatically generate the CA for a Reo circuit.
- Model-checkers integrated in ECT can be used to verify the correctness properties of a protocol.
- ECT can generate executable (Java/C) code from a CA as a single sequential thread.
- http://reo.project.cwi.nl

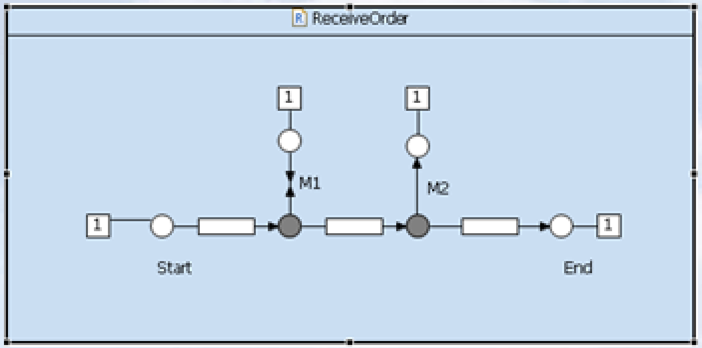
Eclipse Coordination Tools

Tool	Description
Reo graphical editor	Drag and drop editing of Reo circuits
Reo animation plug-in	Flash animation of data-flow in Reo circuits
Extensible Automata editor and tools	Graphical editor and other automata tools
Reo to constraint automata converter	Conversion of Reo to Constraint Automata
Verification tools	•Vereofy model checker (www.vereofy.de) •mCRL model checking •Bounded model checking of Timed Constraint Automata
Java code generation plug-in	State machine based coordinator code (Java, C, and CA interpreter for Tomcat servlets)
Distributed Reo middleware	Distributed Reo code generated in Scala (Actor-based Java)
(UML / BPMN / BPEL) GMT to Reo converter	Automatic translation of UML SD / BPMN / BPEL to Reo
Algebraic Graph Transformation	Dynamic reconfiguration of Reo circuits
Markov chain generator (Reo2MC)	Compositional QoS model based on Reo Analysis using, e.g., probabilistic symbolic model checker Prism (http://www.prismmodelchecker.org)
……	……

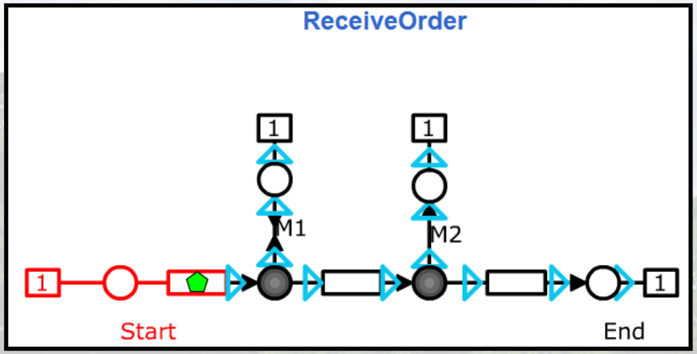
Tool Snapshot



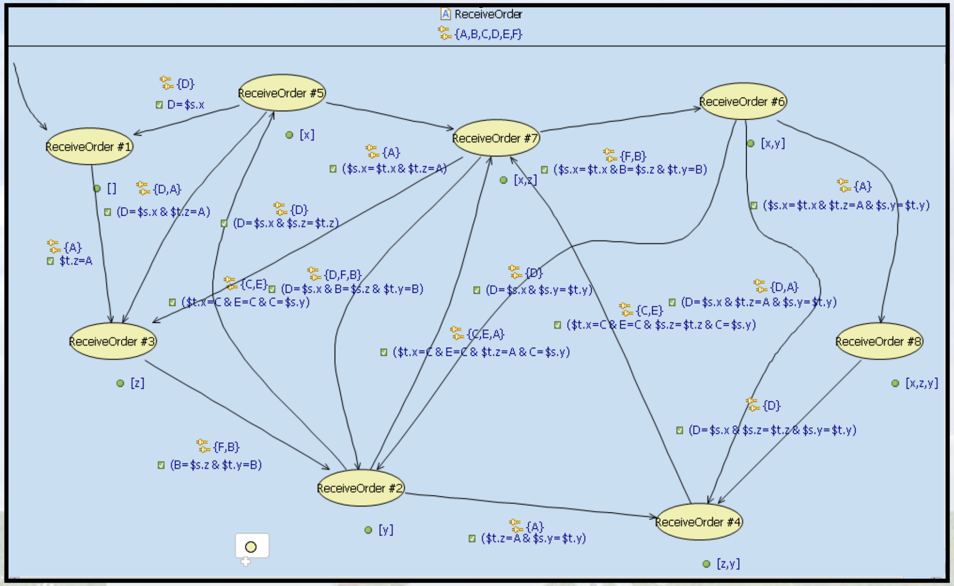
Tool Snapshot



Reo graphical editor

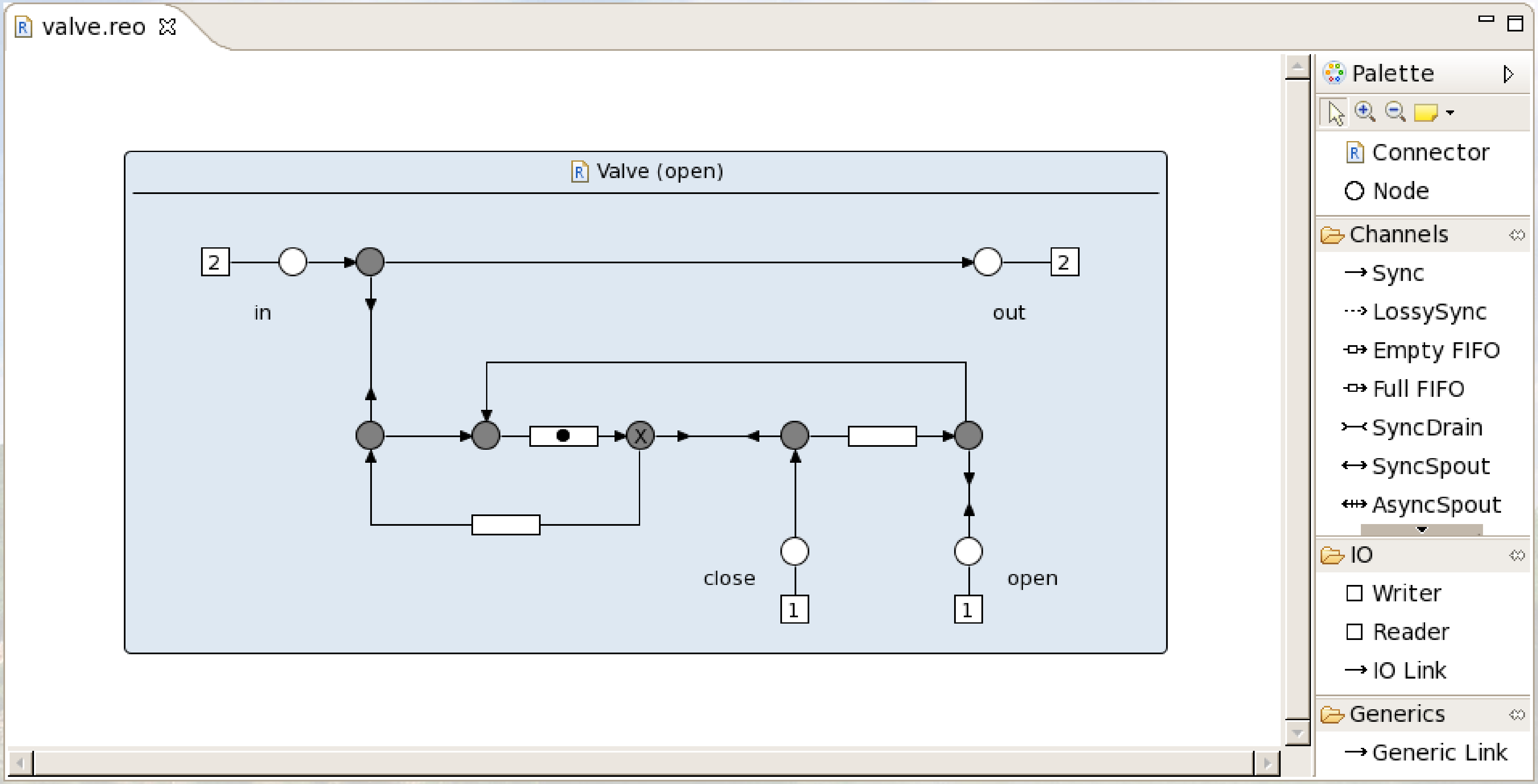


Reo simulation plug-in

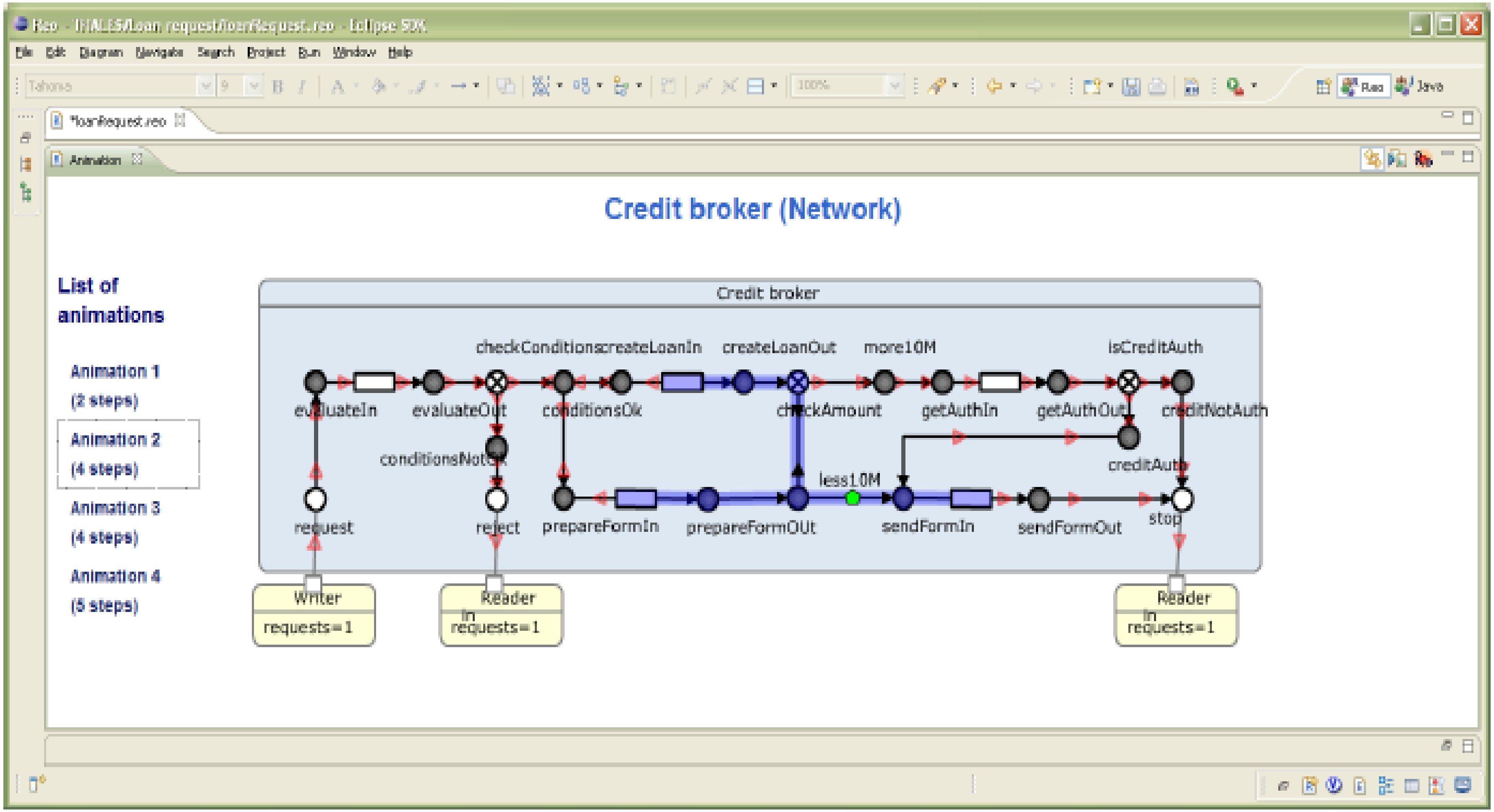


Reo to constraint automata converter

Snapshot of Reo Editor



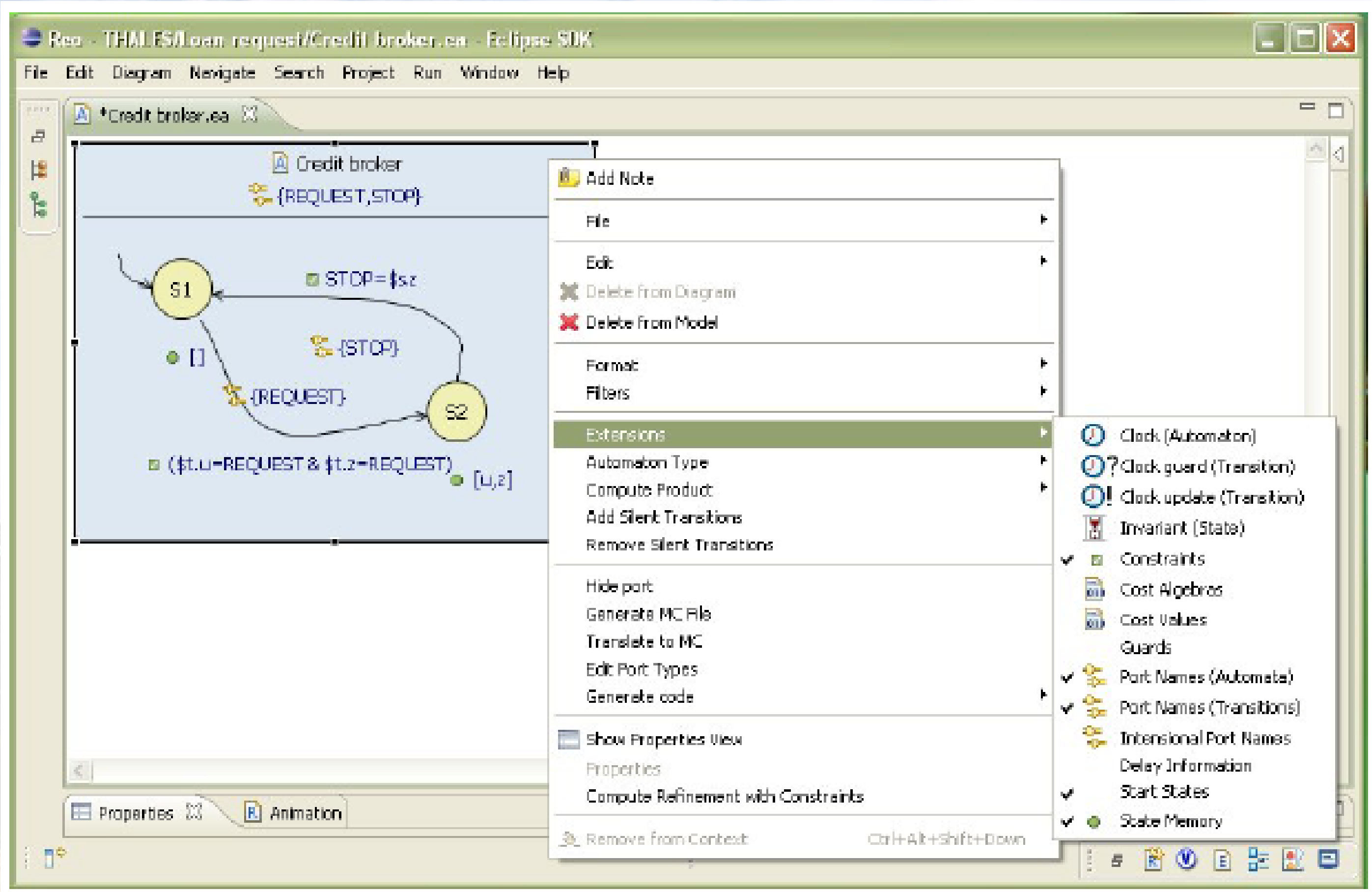
Reo Animation Tool



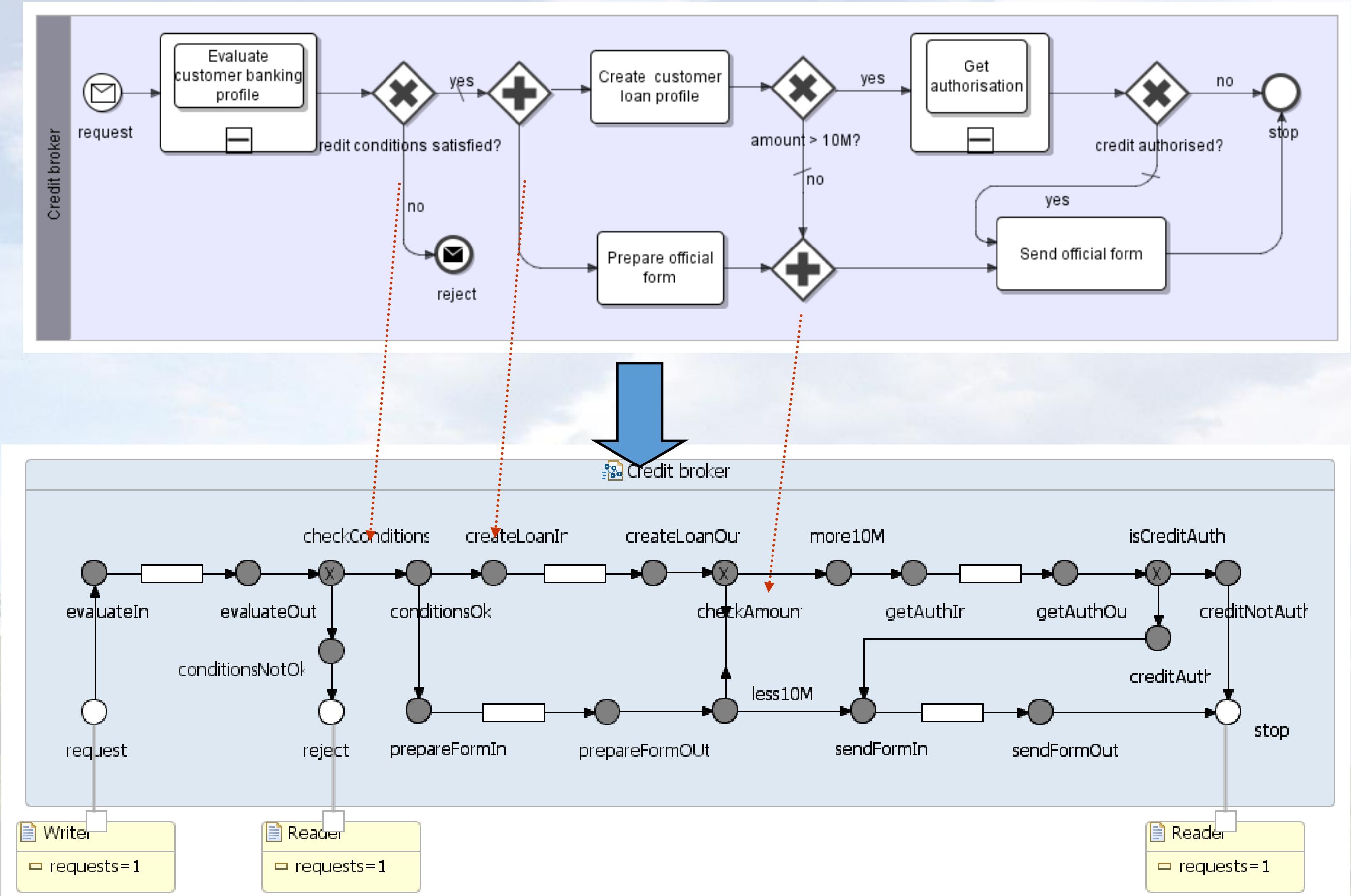
Constraint Automata Tools

- ECT includes a graphical editor for CA and related automata models
  - Create and edit automata graphically
  - Perform product and hiding on automata
- ECT includes tools to automatically derive the CA of a Reo circuit
- ECT includes simulator engines to show automata runs

Constraint Automata Editor

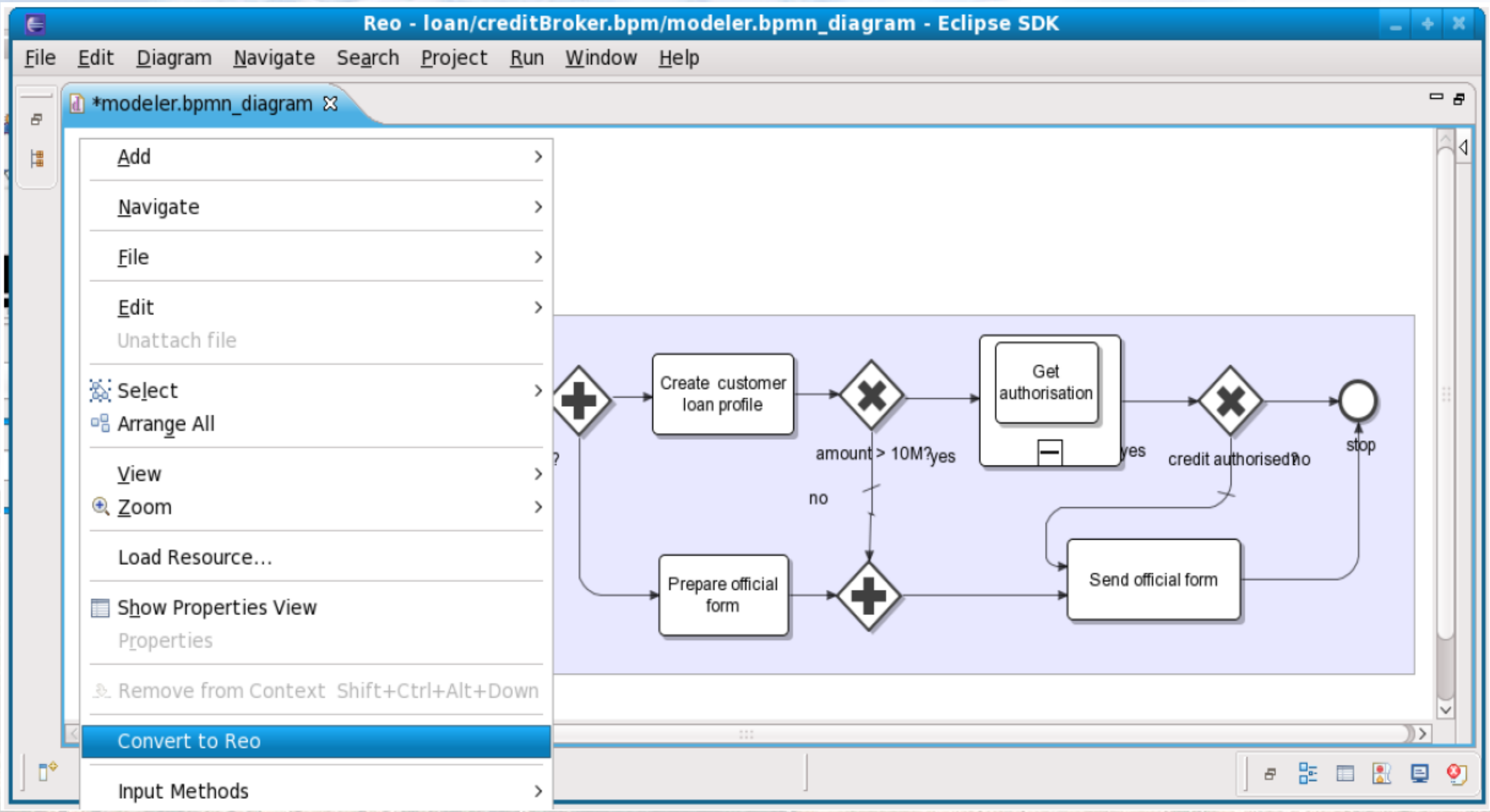


Synthesis from BPMN to Reo

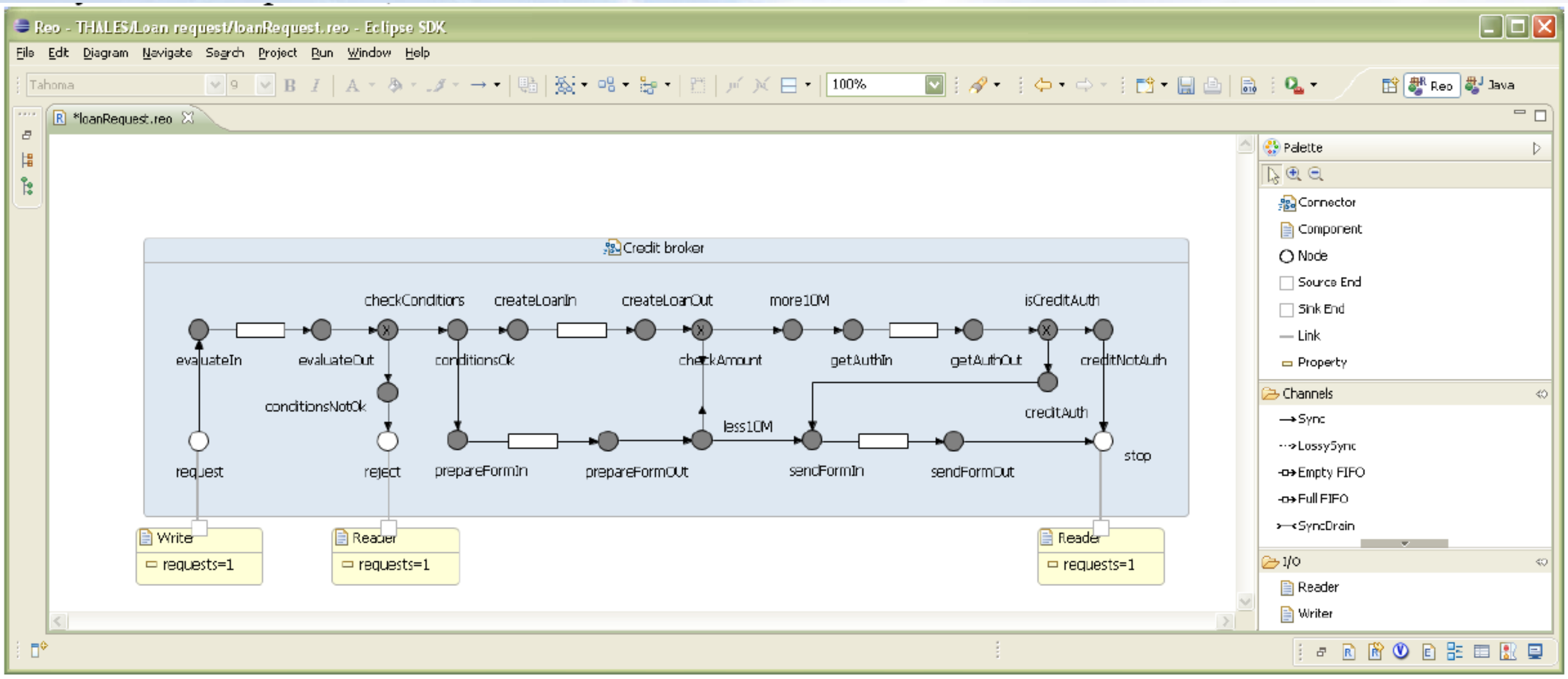


Farhad Arbab, Natallia Kokash and Sun Meng. Towards Using Reo for Compliance-aware Business Process Modeling. In Proceedings of ISoLA'08, pages 108-123, CCIS 17, Springer, 2008.

Input of BPMN-to-Reo Converter

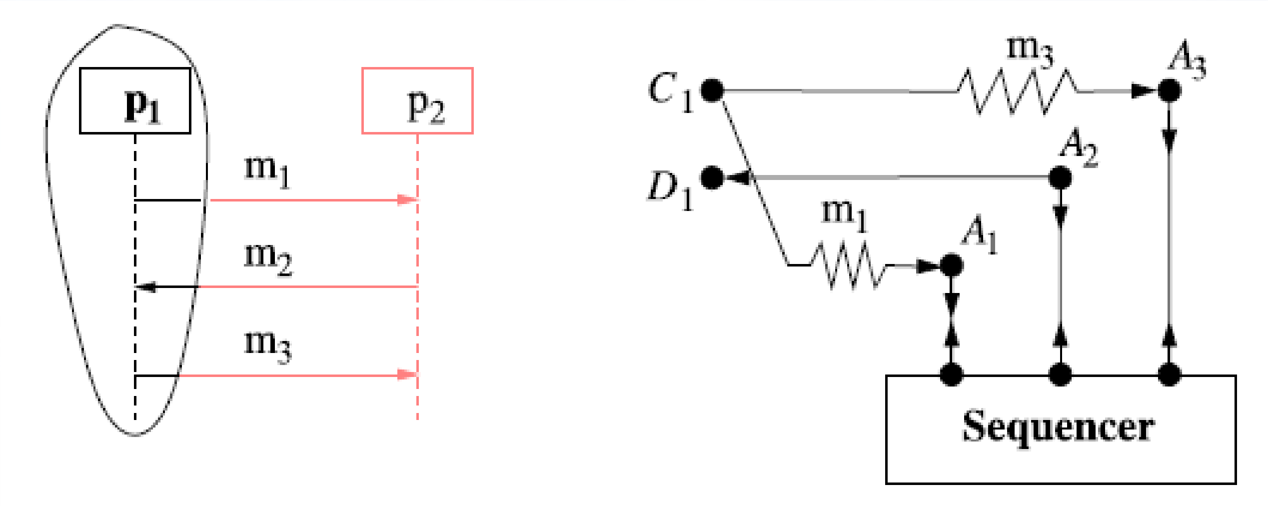


Output of BPMN-to-Reo Converter



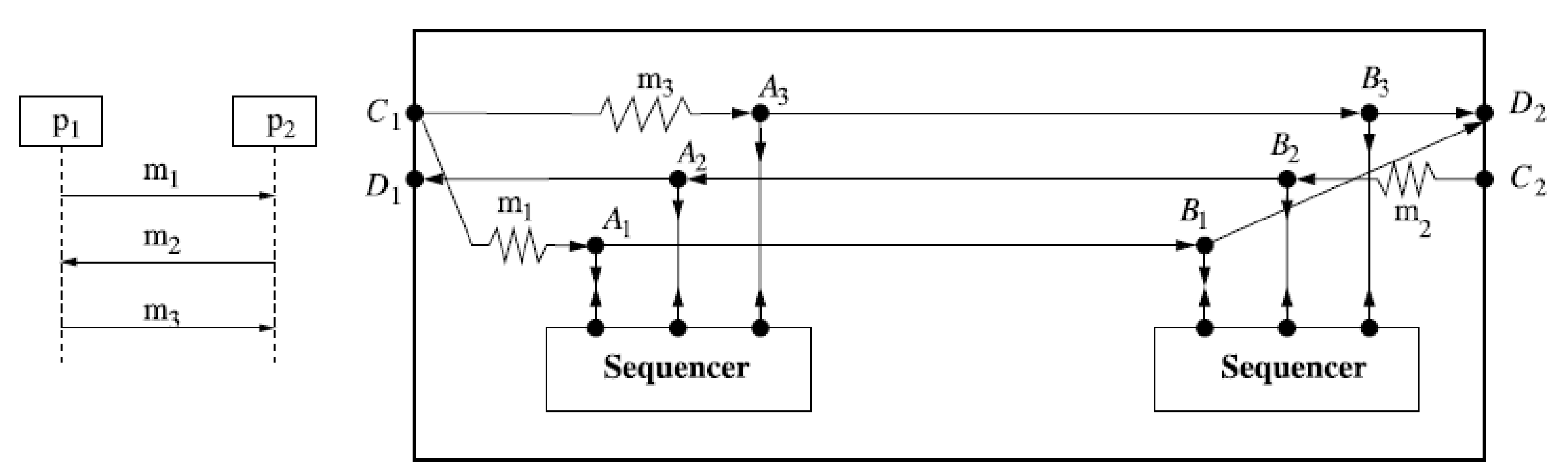
Synthesis from UML SD to Reo

- Sequencers are derived for individual participants



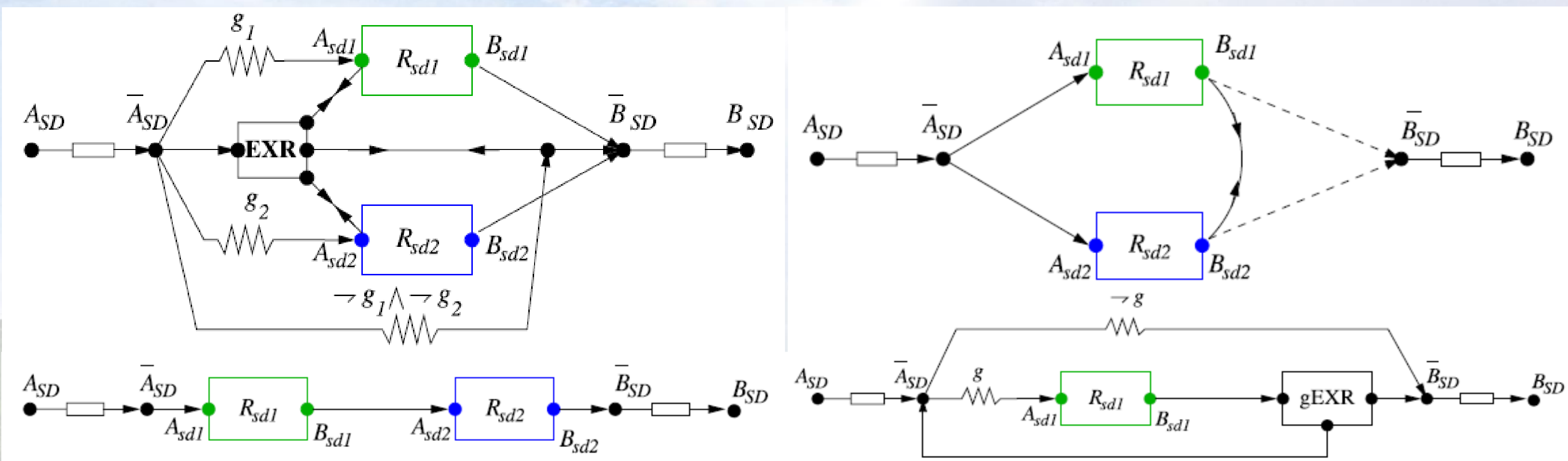
Synthesis from UML SD to Reo

- Nodes for different lifelines are connected pairwise by synchronous or asynchronous channels according to the types and order of messages.



Synthesis from UML SD to Reo

- Reo circuits are structured inductively according to the operators in UML SDs.

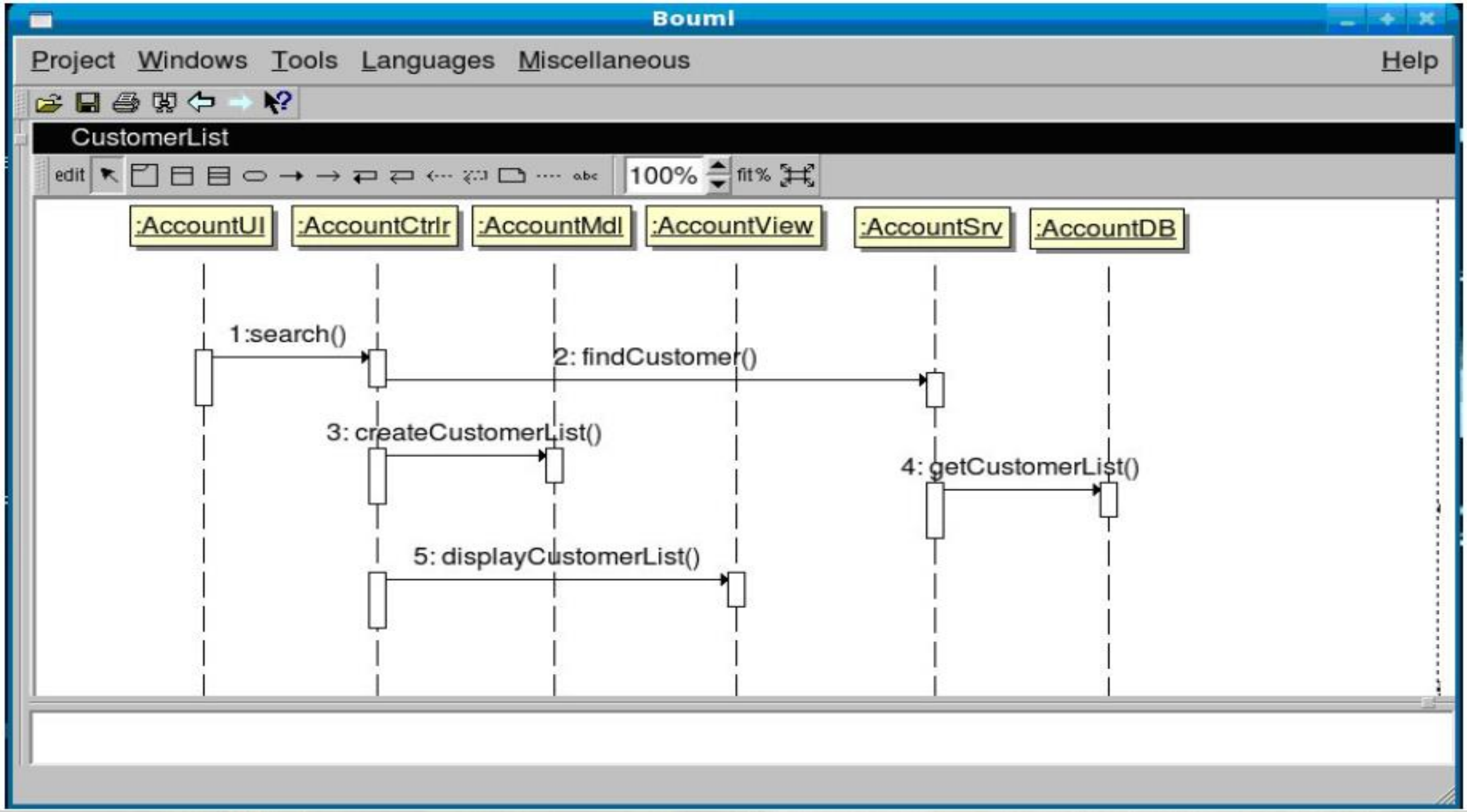


- Correctness of the approach is proved by coinduction.

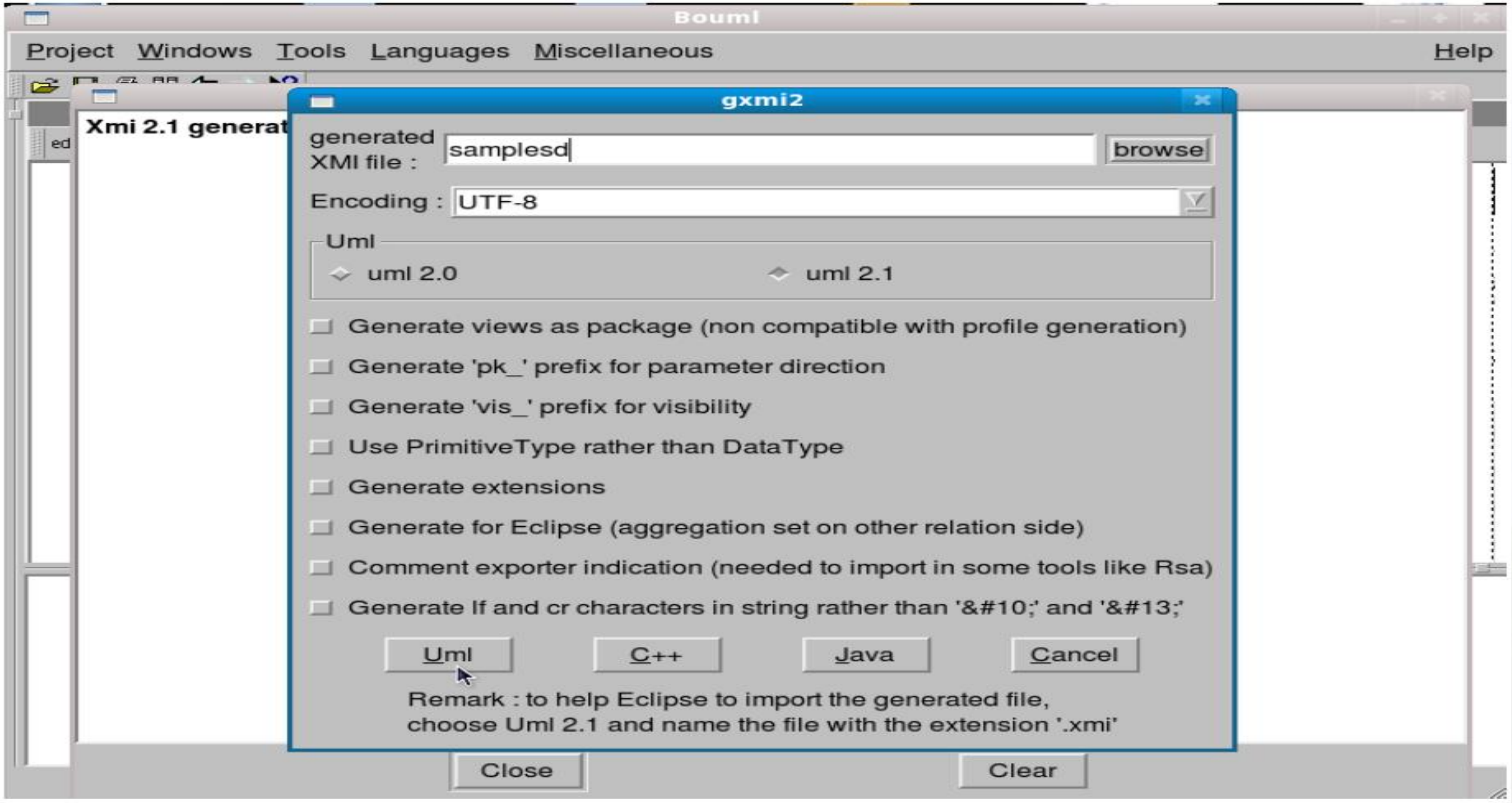
SD-to-Reo Converter

- Accepts UML 2.x SD models as input
- Generates Reo circuits representing the communication protocol
- Can combine SDs for different scenarios and use-cases
- Enables verification and reasoning about the combined protocol
- Originally, a stand-alone tool
- Modified and improved to accept Bouml XMI input
- Support for Eclipse UML2 tool coming

UML SD Editor



SD-to-Reo Converter



References

1. Sun Meng, Farhad Arbab, Christel Baier. Synthesis of Reo circuits from scenario-based interaction specifications. *Science of Computer Programming*, vol. 76, pages 651-680, 2011.
2. Farhad Arbab, Sun Meng and Christel Baier. Synthesis of Reo Circuits from Scenario-based Specifications. In *Proceedings of FOCLASA'08*, Vol. 229 of ENTCS, pages 21-41, 2009.
3. Sun Meng and Luis Barbosa. A Coalgebraic Semantic Framework for Reasoning about UML Sequence Diagrams. In *Proceedings of QSIC'08*, pages 17-26, IEEE Computer Society, 2008.
4. Sun Meng and Luis Barbosa. A Coalgebraic Semantic Framework for Reasoning about Interaction Designs. in Kevin Lano eds. *UML Semantics and its Applications*. Wiley, 2009. (This work is an extension of 3)

Verification

- Connectors as designs for refinement checking and test case generation
- Vereofy: Model checker for Reo built in TU-Dresden:
  - Symbolic model, LTL, and CTL-like logic for specification
  - Can also verify properties such as deadlock-freeness and behavioral equivalence
- SAT-based bounded model checking of Timed Constraint Automata
- Translation of Reo to mCRL2 for model checking
- Translation of Reo to Coq for proving properties

Connectors as Designs

- Every connector **R** can be represented as
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$
- $P(in_{\mathbf{R}})$ ($Q(in_{\mathbf{R}}, out_{\mathbf{R}})$) is the pre-condition (post-condition) that should be satisfied by inputs $in_{\mathbf{R}}$ (outputs $out_{\mathbf{R}}$) on the source (sink) nodes of **R**.
- $in_{\mathbf{R}}$ and $out_{\mathbf{R}}$ are mappings from sets of source and sink node names of **R** to timed data streams respectively.

Connectors as Designs

- Implication of predicates establishes a refinement order over connectors. More concrete implementations imply more abstract specifications.
- For two connectors

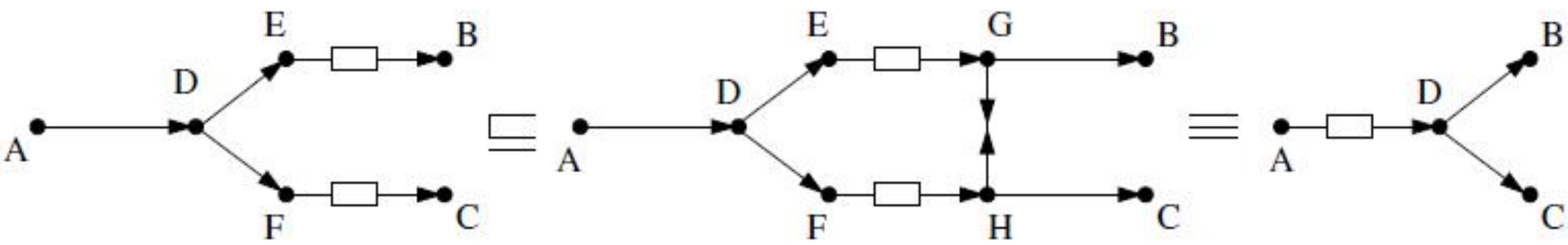
$$\begin{array}{rl} \textbf{con}: & \mathbf{R}_i(in : in_{\mathbf{R}_i}; out : out_{\mathbf{R}_i}) \\ \textbf{pre}: & P_i(in_{\mathbf{R}_i}) \\ \textbf{post}: & Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i}) \end{array}$$

where $i = 1,2$, if $in_{\mathbf{R}_1} = in_{\mathbf{R}_2}$ and $out_{\mathbf{R}_1} = out_{\mathbf{R}_2}$, then

$$\mathbf{R}_1 \sqsubseteq \mathbf{R}_2 =_{df} (P_1 \Rightarrow P_2) \wedge (P_1 \wedge Q_2 \Rightarrow Q_1)$$

- Pre-conditions on inputs of connectors are weakened under refinement, and post-conditions on outputs of connectors are strengthened.

Connectors as Designs



(a) ⊑ (b) ≡ (c)

```
testRefinement of
connector ABCa = R(( a , sa ) : ( c , sc ), ( b , sb )) , D< sa > |-
        D< comp(dc,tc) > and tc =t between2(ta) and D< comp(db,tb) >
          and tb =t between2(ta) and db =d dc and da =d db
and
connector ABCc = R(( a , sa ) : ( b , sb ), ( c , sc )) ,
    D< comp(da,ta) > |- D< comp(db,tb) > and between2(ta) =t tb and
    da =d db and D< comp(dc,tc) > and between2(ta) =t tc and da =d dc
is:
True() and True()
```

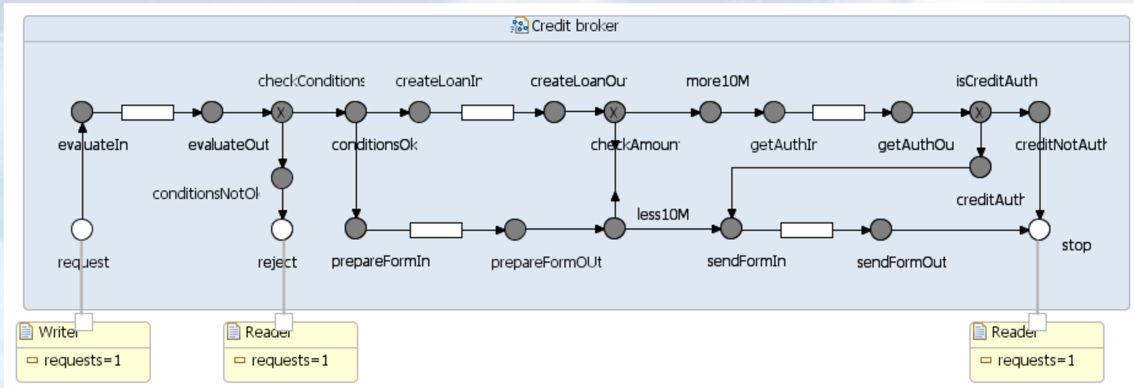
References

1. Sun Meng，Farhad Arbab, Bernhard K. Aichernig, Lacramioara Astefanoaei, Frank S. de Boer and Jan Rutten. Connectors as Designs: Modeling, Refinement and Test Case Generation. In *Science of Computer Programming*. vol. 77(7-8), pages 799-822, 2012.
2. Sun Meng. Connectors as Designs: the Time Dimension. In *Proceedings of TASE 2012*, pages 201-208, IEEE Computer Society, 2012.
3. Bernhard K. Aichernig, Farhad Arbab, Lacramioara Astefanoaei, Frank S. de Boer, Sun Meng and Jan Rutten. Fault-based Test Case Generation for Component Connectors. In *Proceedings of TASE 2009*, pages 147-154, IEEE Computer Society, 2009.
4. Sun Meng and Farhad Arbab. Connectors as Designs. In *Proceedings of FOCLASA’09*, Vol. 255 of ENTCS, pages 119-135, 2009.

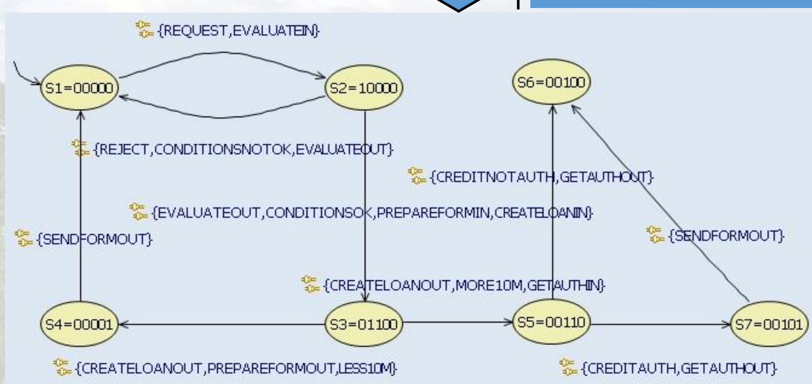
Vereofy Model Checker

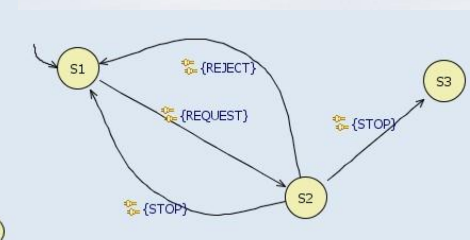
- Symbolic model checker for Reo:
  - Based on constraint automata
  - Developed at the University of Dresden
  - LTL and CTL-like logic for property specification
- Modal formulae
  - Branching time temporal logic:
    - AG[EX[*true*]]
    - check for deadlocks
  - Linear temporal logics:
    - $\mathbf{G}(\mathit{request} \rightarrow \mathbf{F}\ (\mathit{reject} \cup \mathit{sendFormOut}))$
    - check that admissible states *reject* or *sendFormOut* are reached
- http://www.vereofy.de

Verification with Vereofy



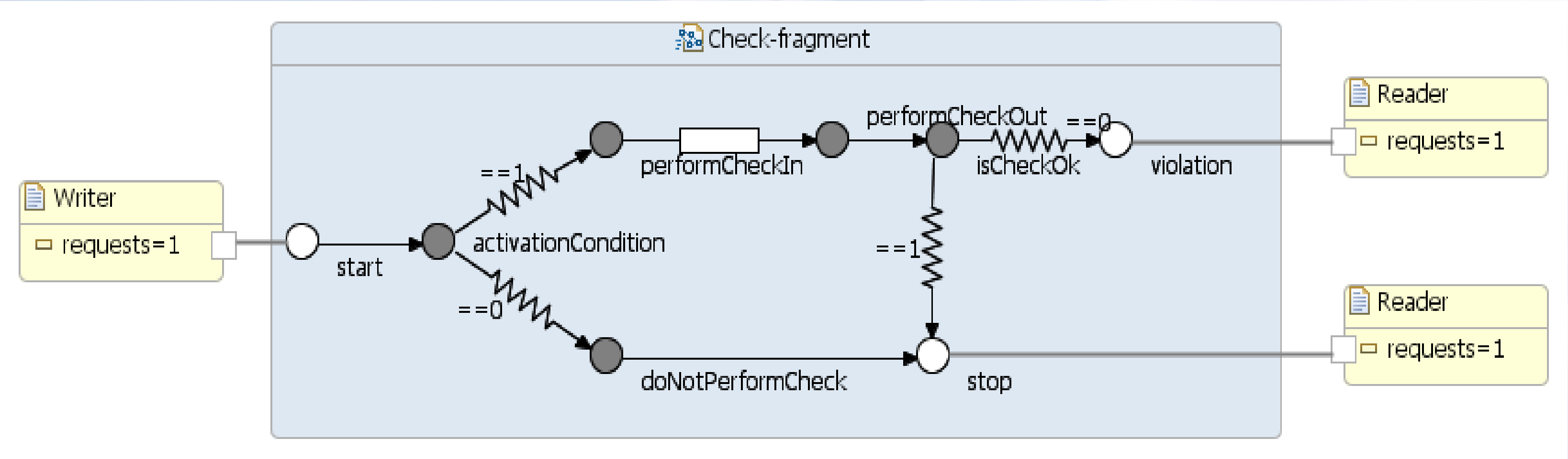
Reo2ConstraintAutomata





- Modal formulae
  - Branching time temporal logic: AG[EX[*true*]] – check for deadlocks
  - Linear temporal logics: **G**(*request* → **F** (*reject* ∪ *sendFormOut*)) – check that admissible states *reject* or *sendFormOut* are reached

Data-Dependent Control-Flow

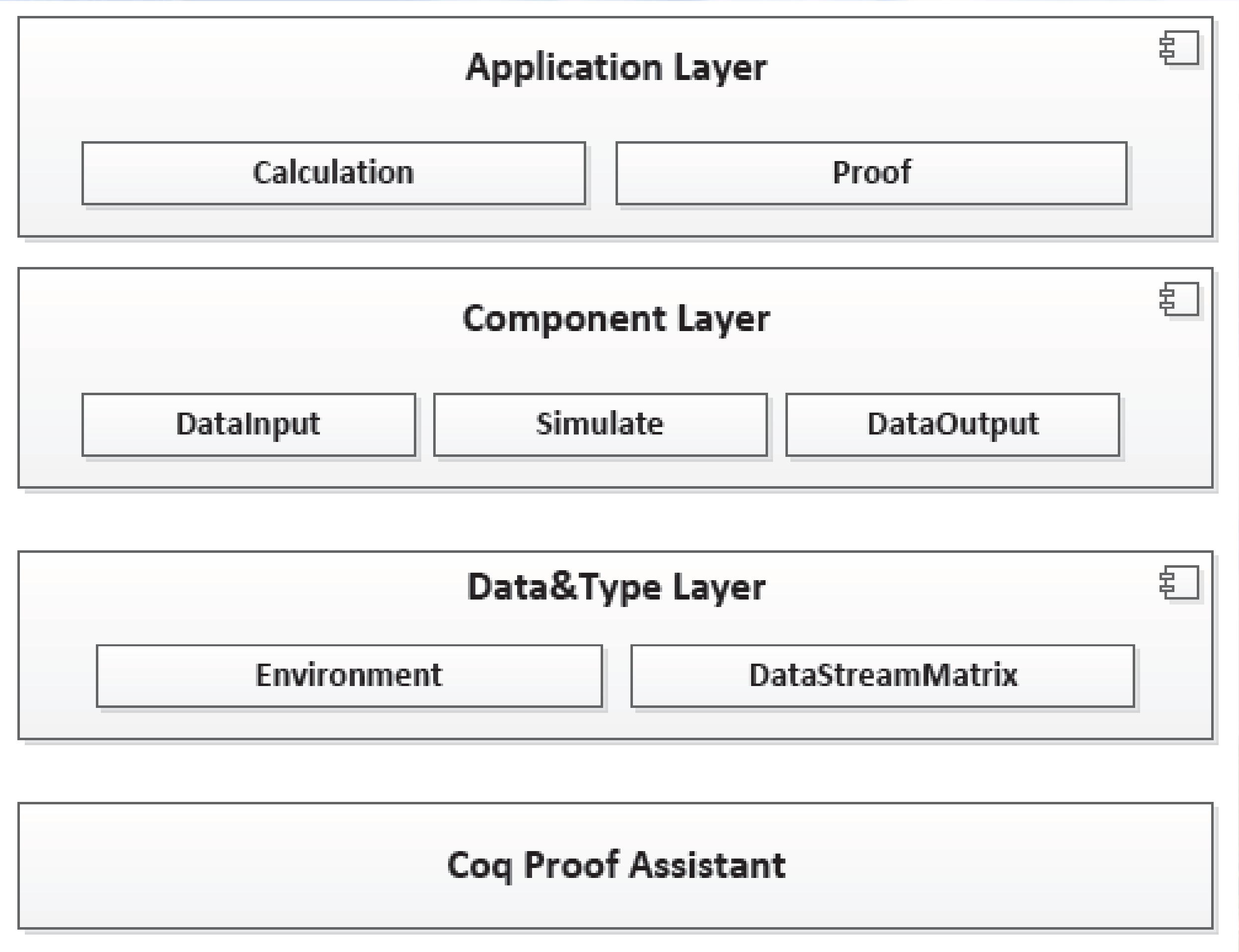


- Input parameters:
  - Activation condition
    - **Data:** b: Boolean
    - **Filter condition:** b==true, b==false
  - Check condition
    - **Data:** x, y: Real; (e.g., credit amount, maximal amount)
    - **Filter condition:** x < y
- Problems:
  - Data constraint specification language is needed
  - Properties that include conditions:
    - **G [**(b & !(x < y)) **→ F** violation]

Verification with mCRL2

- mCRL2 behavioral specification language and associated toolset developed at TU Eindhoven
  - http://www.mcrl2.org
  - Based on the Algebra of Communicating Processes (ACP)
  - Extended with data and time
  - Expressive property specification language (μ calculus)
  - Abstract data types, functional language (λ calculus)
- Automated mapping from Reo to mCRL2

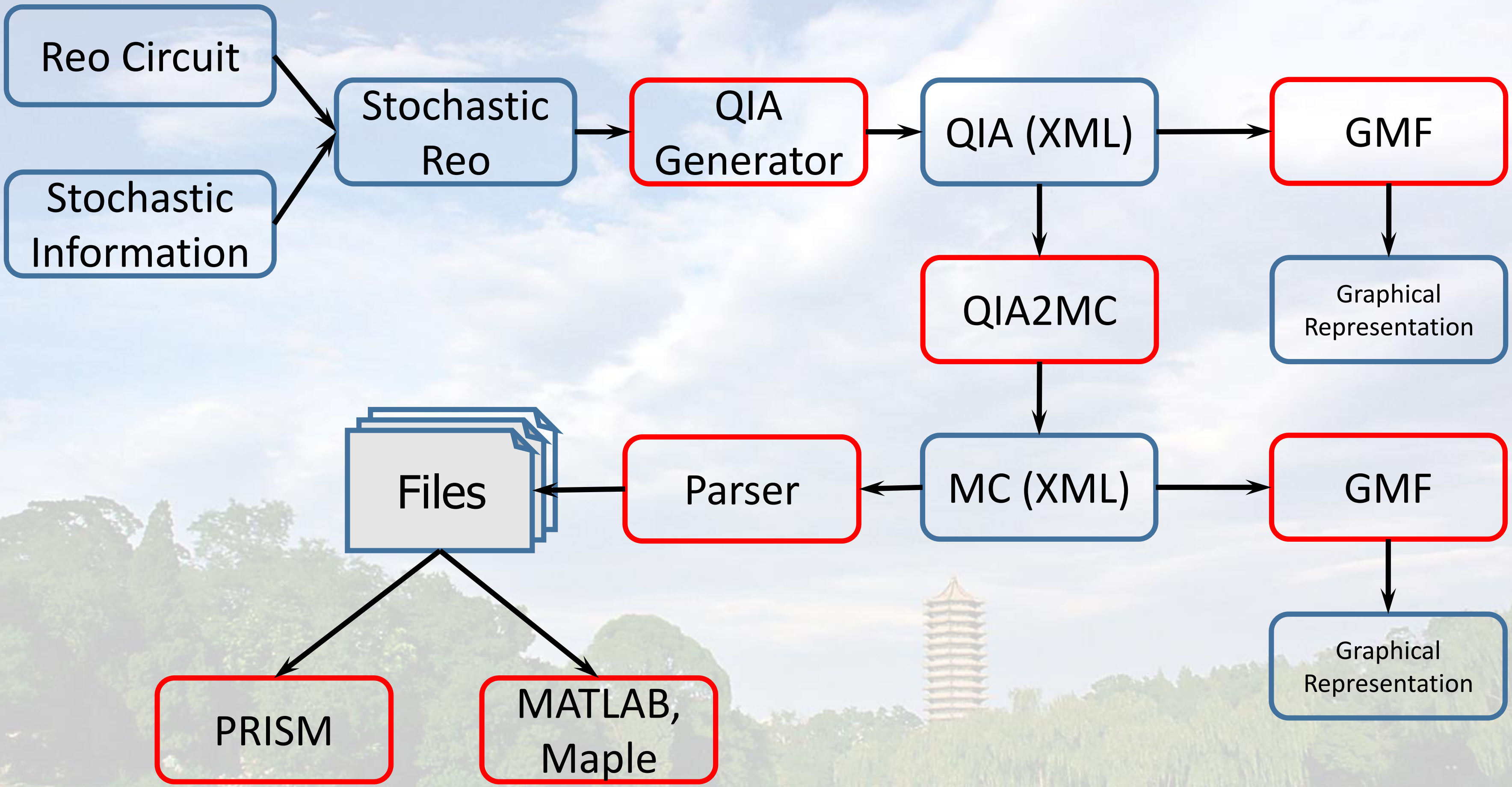
Verification with Coq



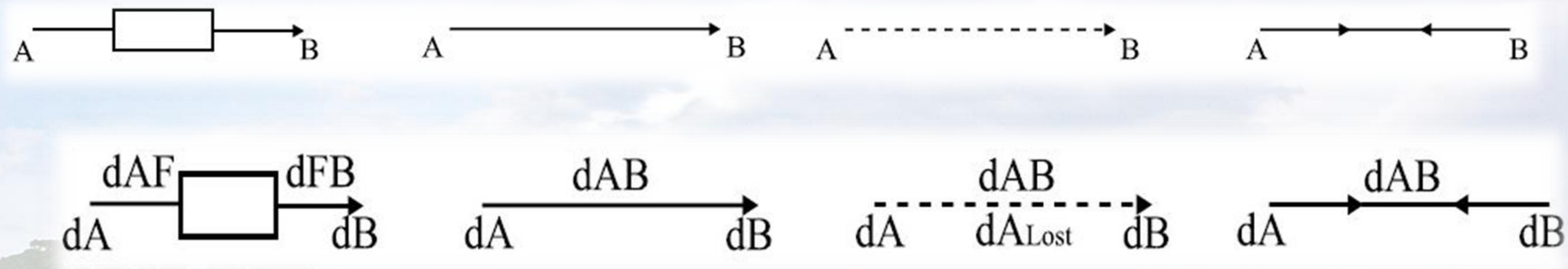
Performance Analysis

- Quantitative Intentional Automata (QIA) extend CA with quantitative properties:
  - arrival rates at ports
  - average delays of data-flows between ports
- Quantified Reo circuits are converted to QIA
- Markov Chain models are derived from QIA
  - Resulting Markov Chains are very compact: efficient model checking
- PRISM is used for analysis of MC models
  - Farhad Arbab, Sun Meng, Young-Joo Moon, Marta Kwiatkowska and Hongyang Qu. Reo2MC: a Tool Chain for Performance Analysis of Coordination Models. *Proceedings of ESEC/FSE'09*, pages 287-288, ACM, 2009.
  - Farhad Arbab, Tom Chothia, Rob van der Mei, Sun Meng, Young-Joo Moon and Chretien Verhoef. From Coordination to Stochastic Models of QoS. *Proceedings of Coordination'09*, LNCS 5521, pages 268-287, Springer, 2009.

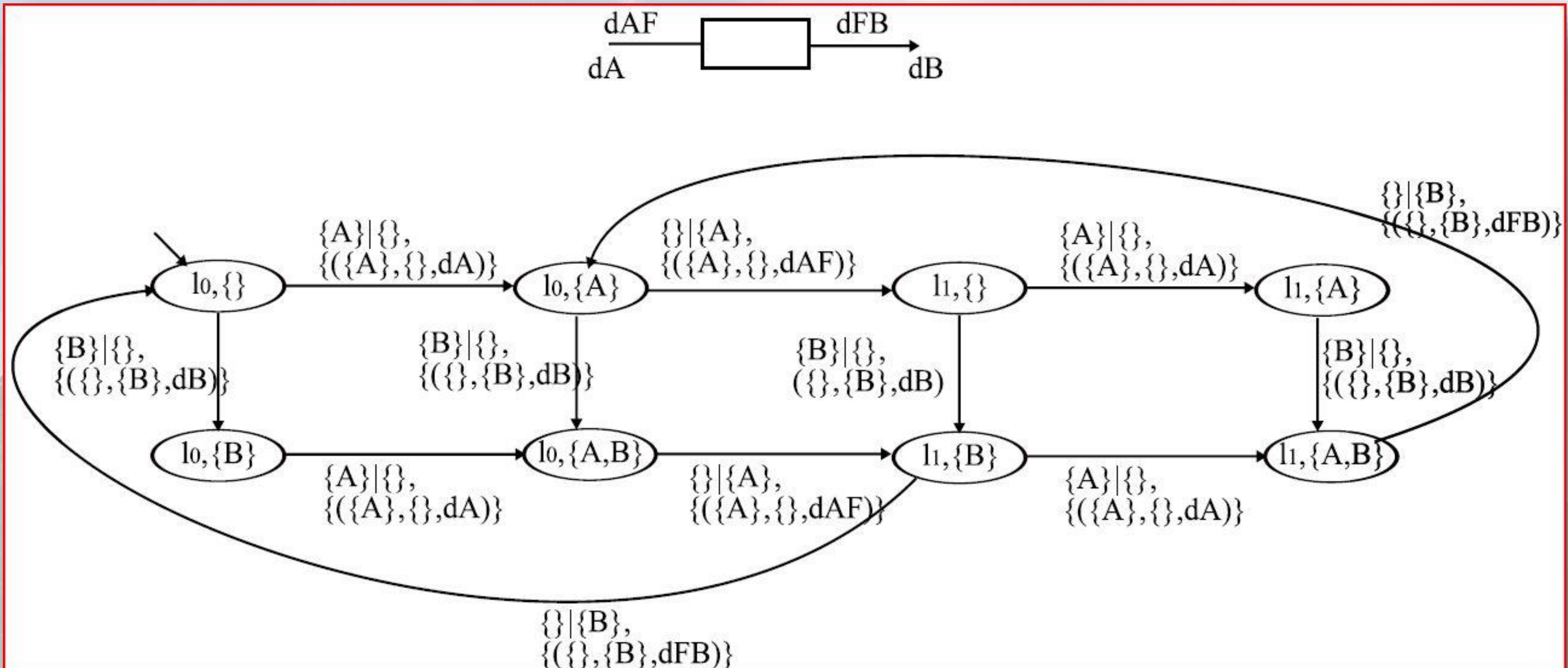
Performance Analysis



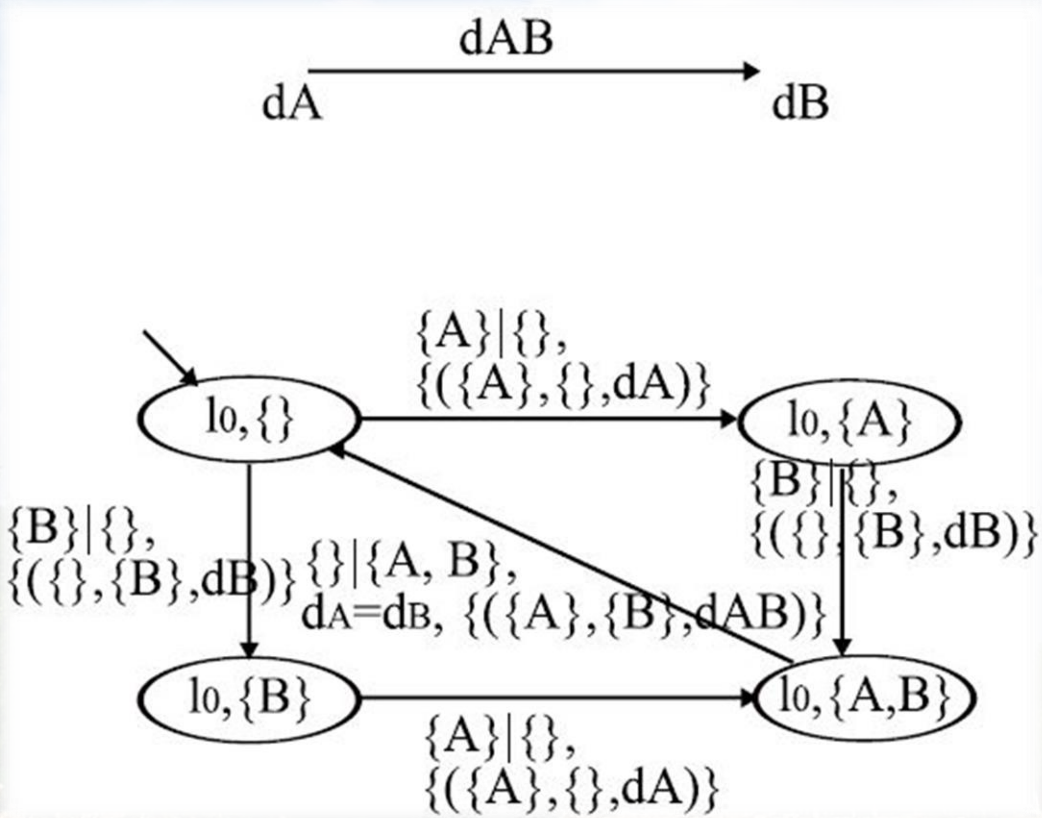
Reo Primitives with Delays



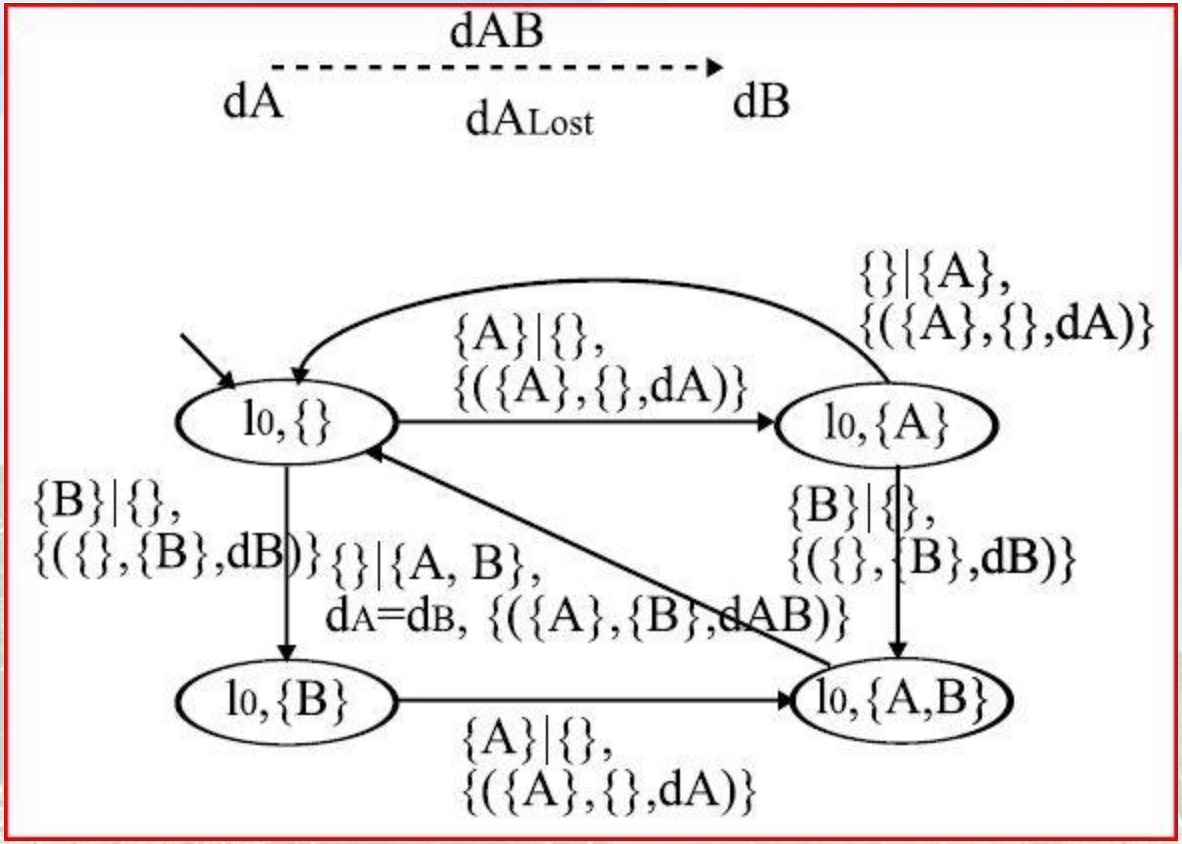
QIA for FIFO1



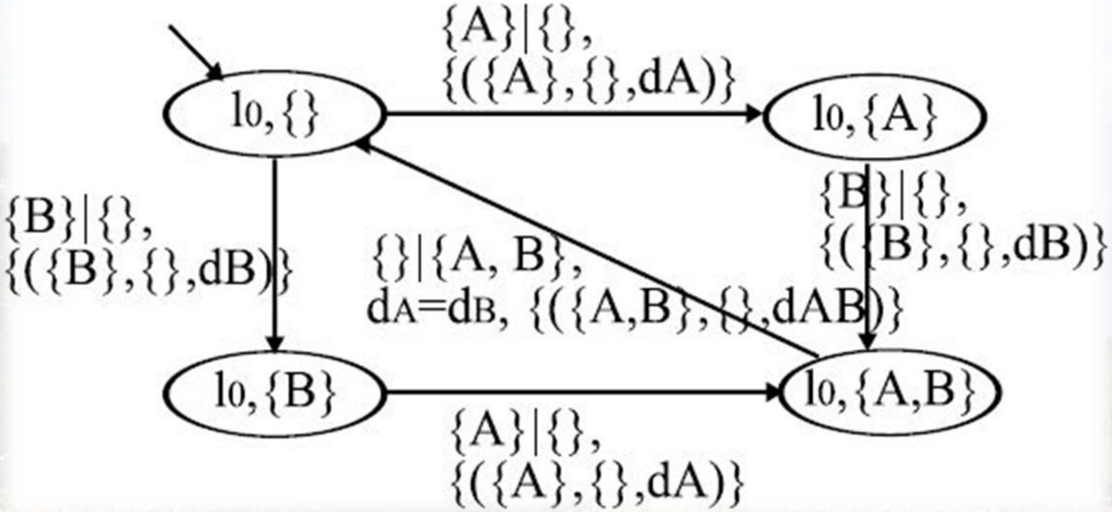
QIA for Sync



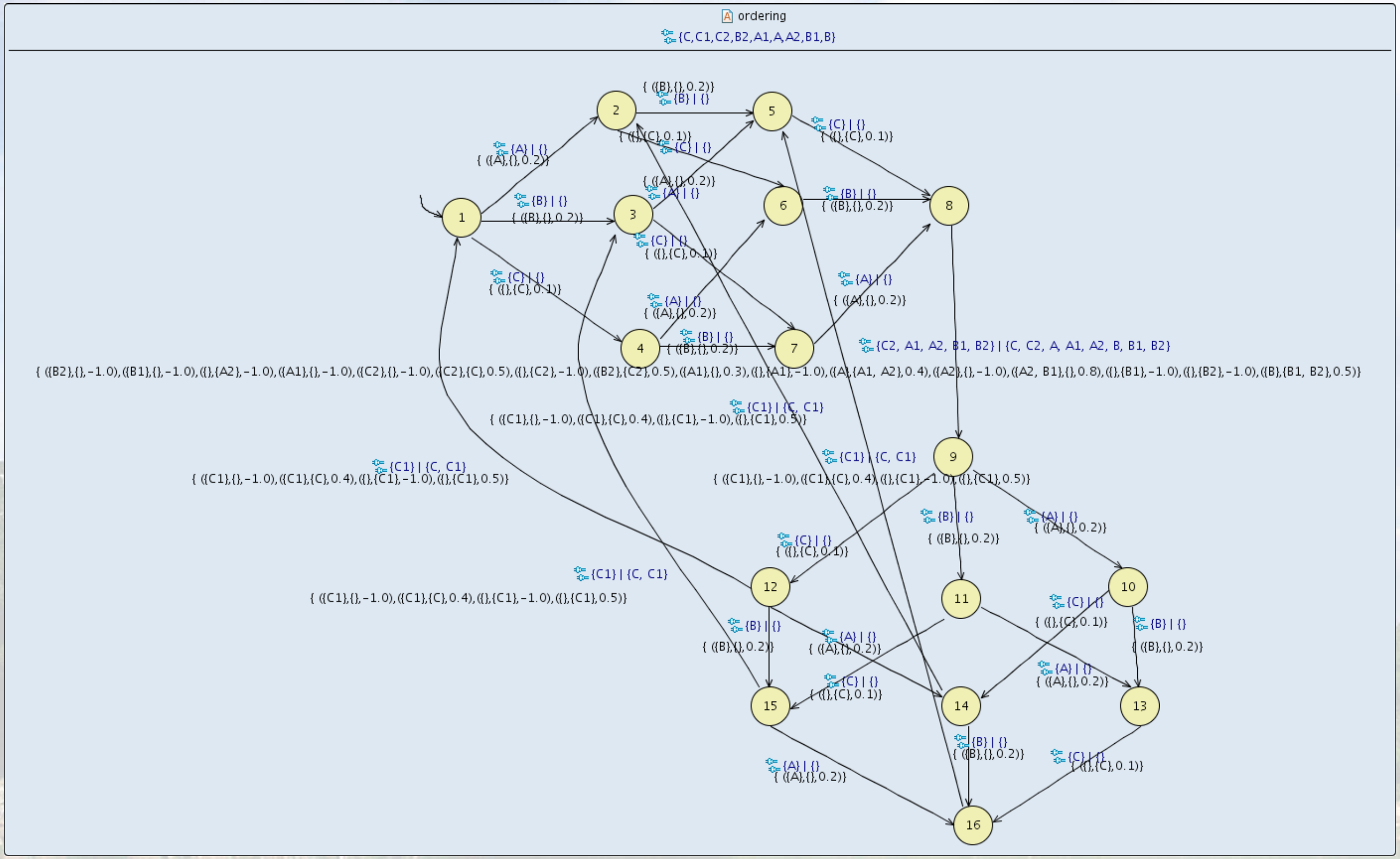
QIA for LossySync



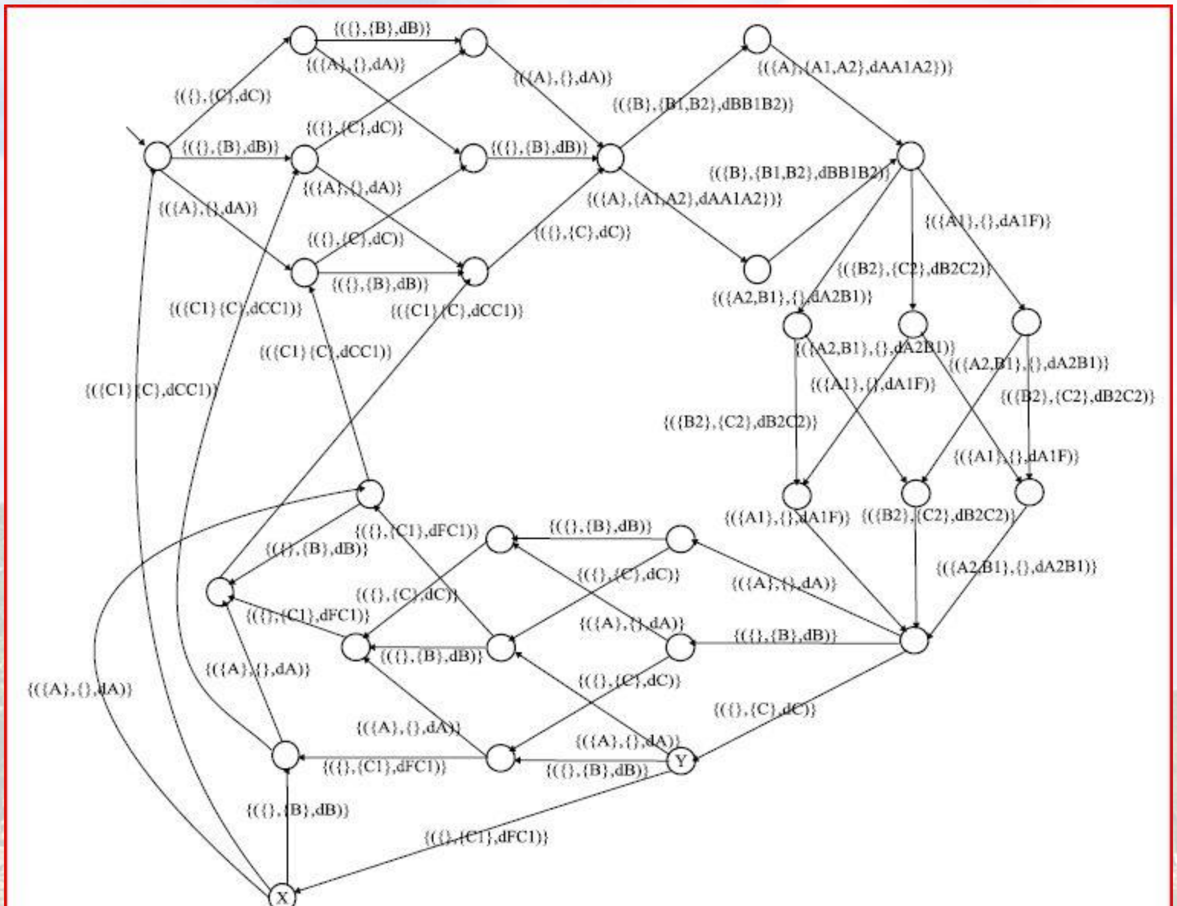
QIA for SyncDrain



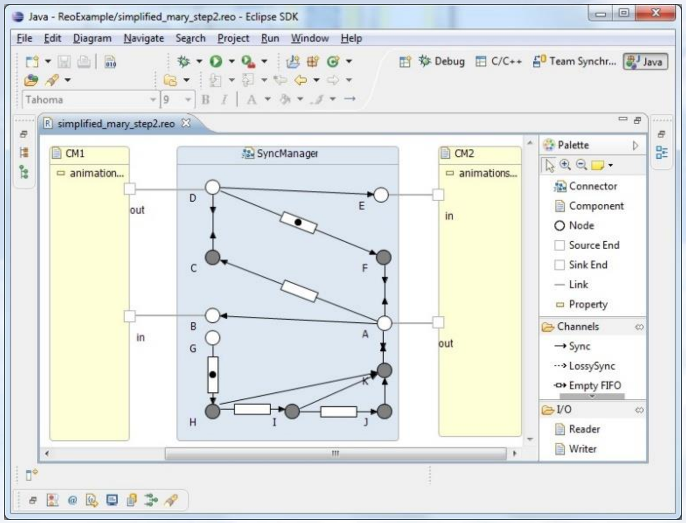
QIA of Alternator Reo Circuit

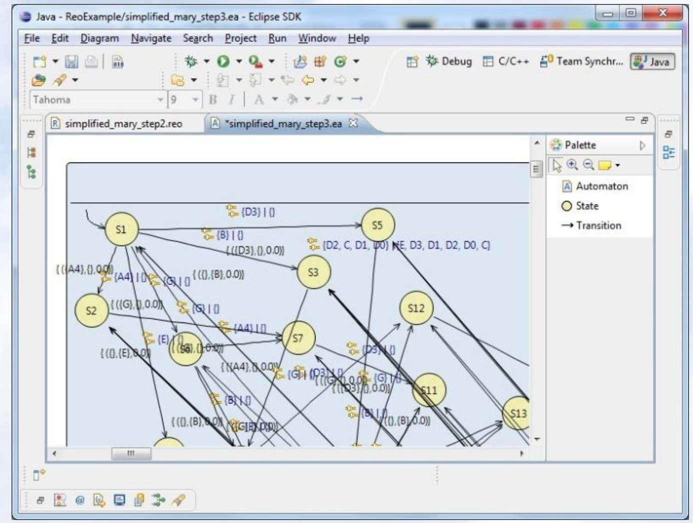


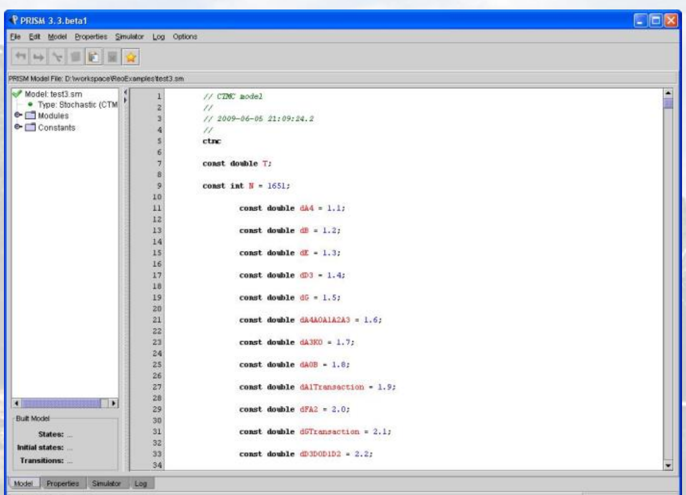
Markov Chain for Alternator

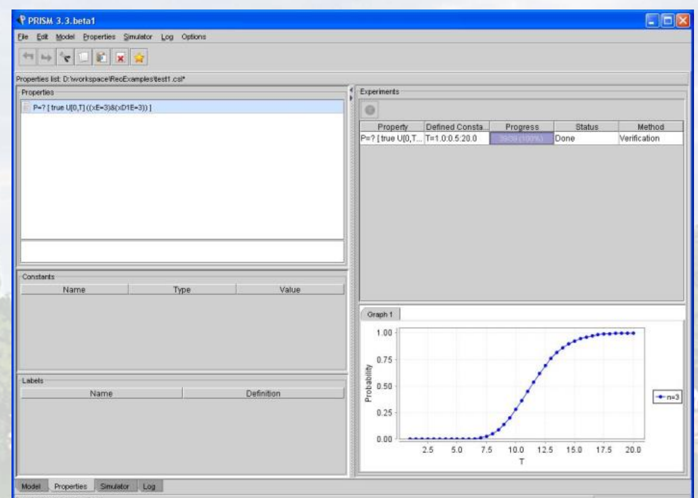


Experiment


Java - ReoExample/simplified_mary_step2.reo - Eclipse SDK
simplified_mary_step2.reo
CM1
SyncManager
CM2
Palette
Connector
Component
Node
Source End
Sink End
Link
Property
Channels
Sync
LossySync
Empty FIFO
I/O
Reader
Writer


Java - ReoExample/simplified_mary_step3.ea - Eclipse SDK
simplified_mary_step2.reo
*simplified_mary_step3.ea
Palette
Automaton
State
Transition


PRISM 3.3.beta1
// CTMC model
ctmc
const double T;
const int N = 1651;


PRISM 3.3.beta1
Properties
Experiments
Constants
Labels
Probability
T

Conclusion & Future Work

- Making interaction explicit in concurrency allows its direct
  - Specification
  - composition
  - Analysis
  - Verification
  - reuse
- Reo is a simple, rich, versatile, and surprisingly expressive language for compositional construction of pure (coordination or concurrency) protocols.
- Extension of the language for hybrid systems and related tools development.

Thanks!
Heel hartelijk bedankt!
ich danke Ihnen sehr!
谢谢!

Questions?
Je hebt een problem?
Sie haben ein Problem?
问题？