Reo
A Channel-Based Model for Component Composition

Seyed Mohammad Mehdi Ahmadpanah

smahmadpanah@aut.ac.ir

Apr. 10, 2018

Outline

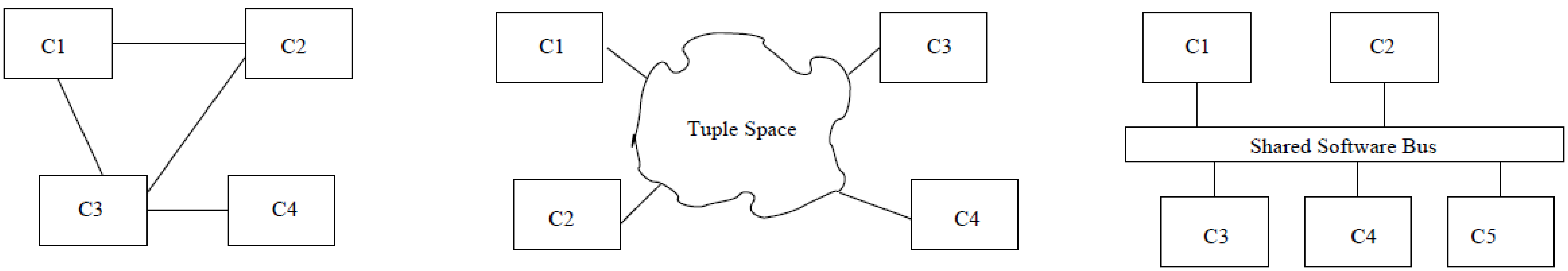
- Motivation
- Introduction
- Syntax
- Semantics
- Tool
- Conclusion

Motivation

- Modular Design Popularity
- Need for a Glue-Language
  - interfacing gaps must be filled with additional code
- Bulk of Specialized Glue-Code in Complex Systems
  - interfacing code for compositional construction
- Maintenance Difficulties
  - hard to evolve, inflexible
- Alternative: construct the glue code compositionally, out of primitive $connectors$
- Reo introduced by Dr. Farhad Arbab in 2004
  - composition of $channels$ to make complex connectors

Motivation (cont.)

- Channel-based communication models are complete

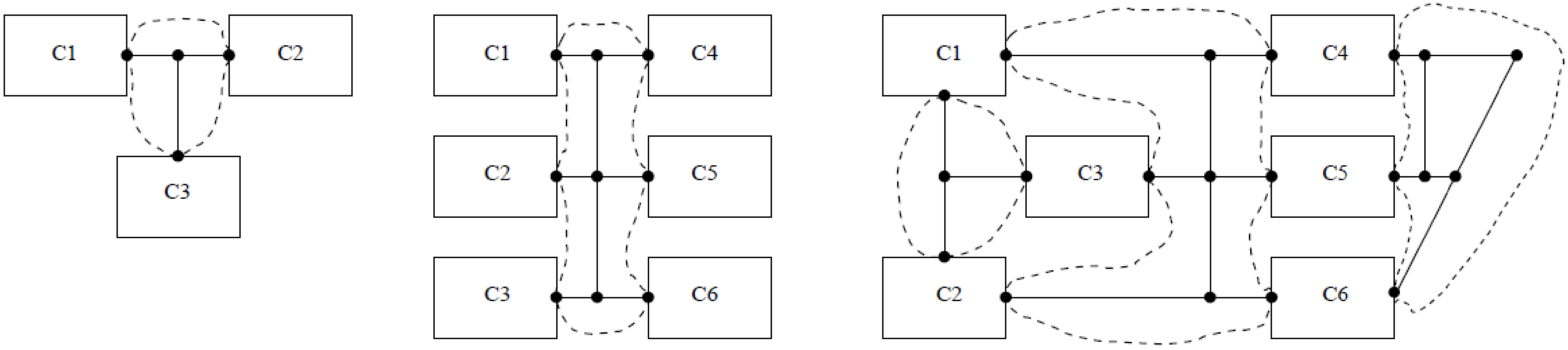


(a) peer−to−peer (b) Shared Data Space (c) Software Bus

- Advantages over Other Communication Models
  - Efficiency
  - Security
  - Architectural Expressiveness
  - Anonymity

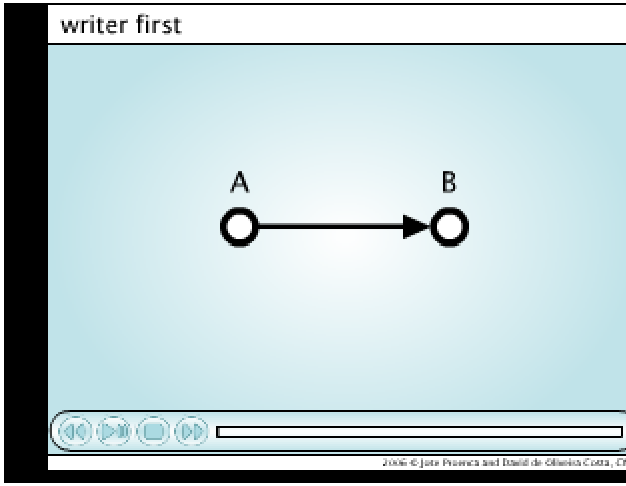
Introduction

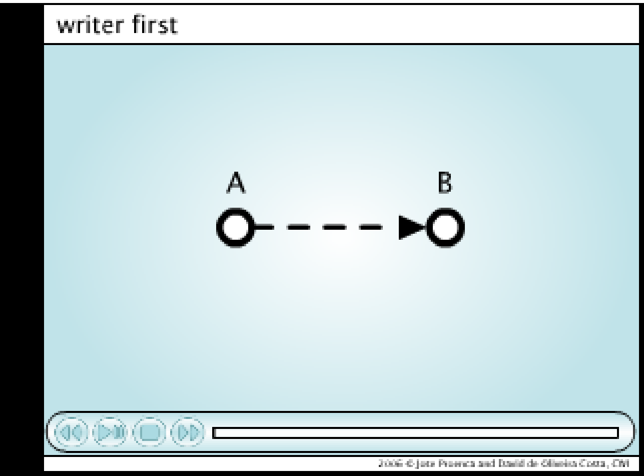
- System consists of Components and Connectors
  - Emphasis on Connectors and their composition only!
  - Coordination from outside
  - Based on calculus of channels
    - complex connectors are constructed through composition of simpler ones

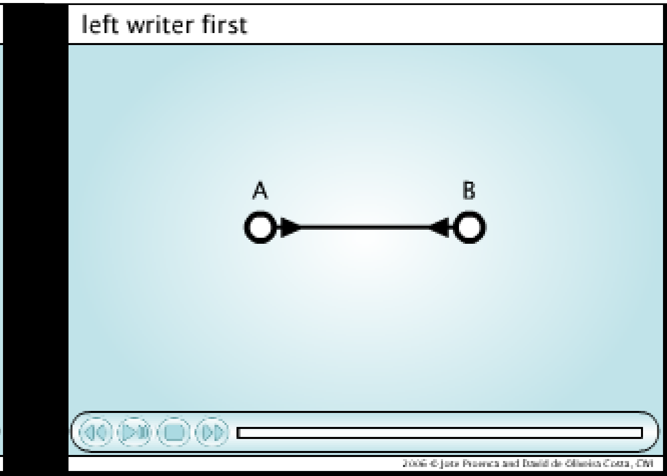


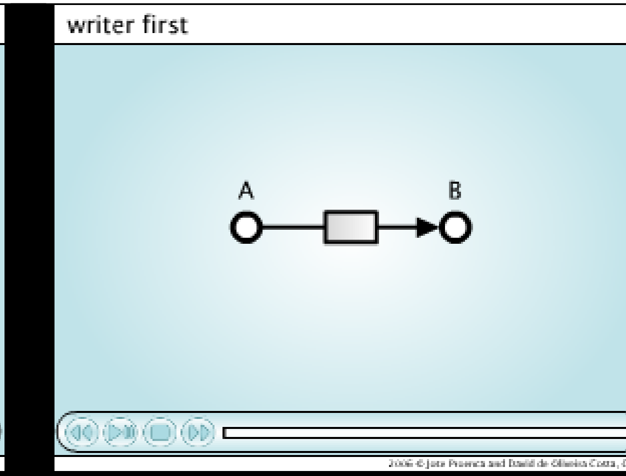
Channel Types

- Sync (write/take)
- Lossy
- SyncDrain (write/write)
- FIFO and FIFOn



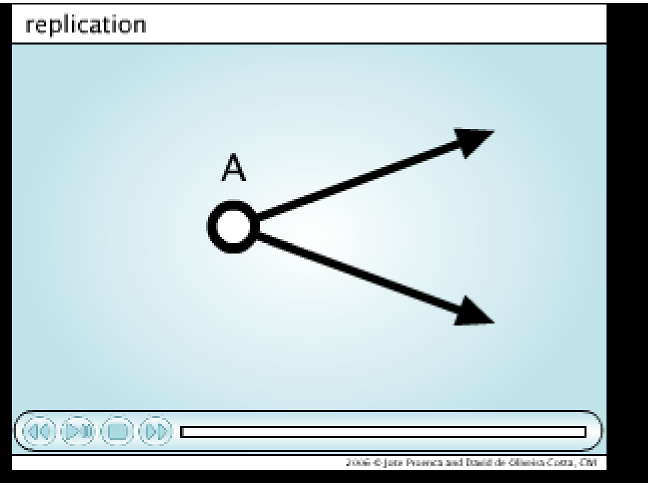


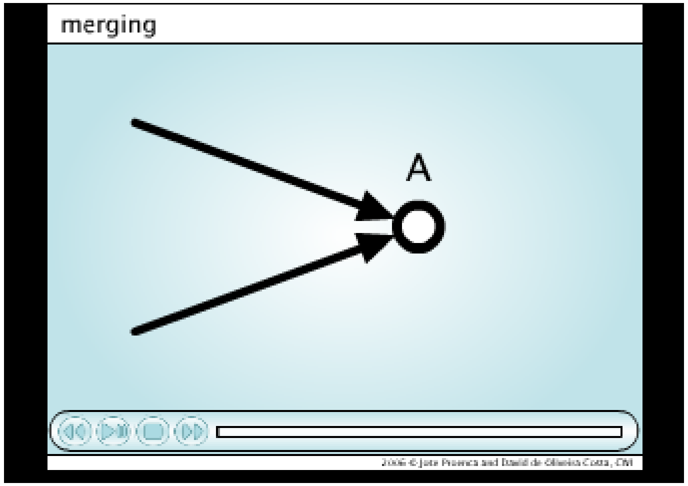


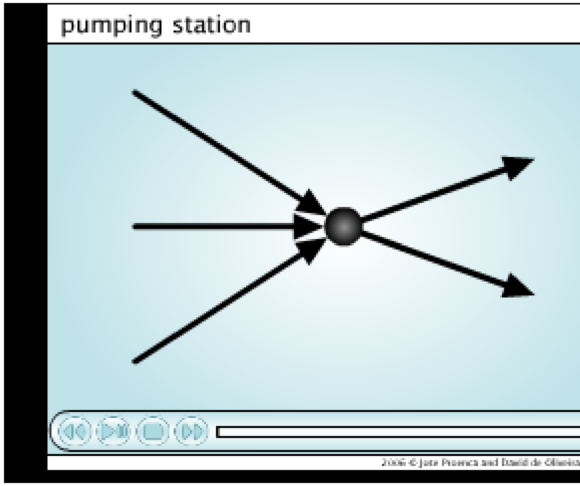


Node Types

- Source: accepts data into its channel
- Sink: dispenses data out of its channel
- Mixed: Source + Sink (no buffer)





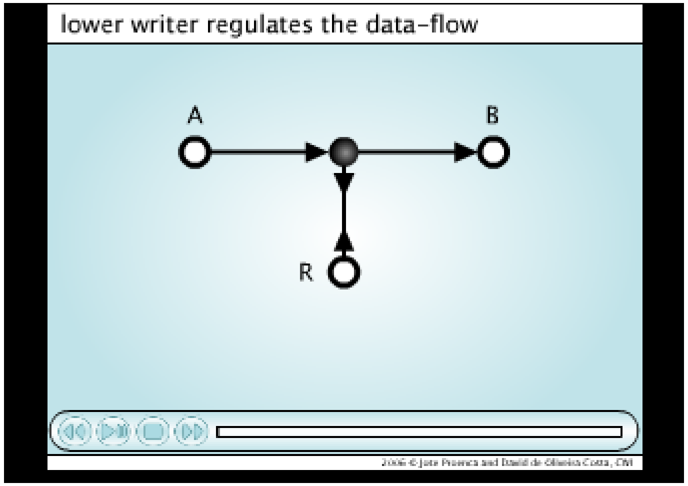


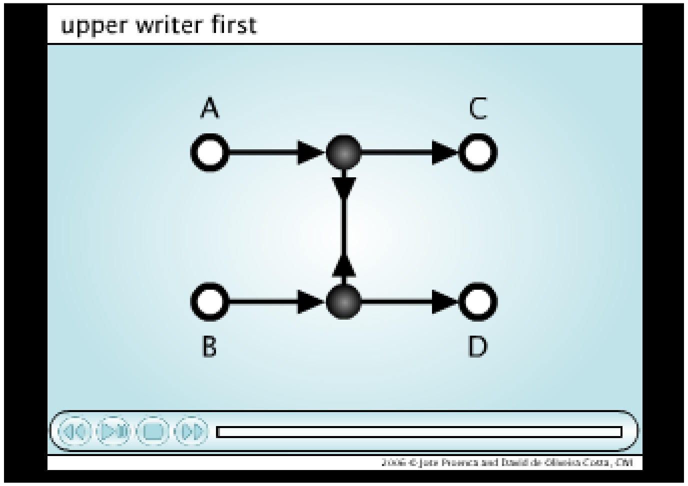
Syntax

- The set of primitive operations on channels and channel ends
  - create(ChanType)
  - forget(e)
  - move(e, loc)
  - connect([t,] e)
  - disconnect(e)
  - wait([t,] conds)
  - read([t,] inp[, v[, pat]])
  - take([t,] inp[, v[, pat]])
  - write([t,], outp, v)
  - join(e1, e2)
  - split(e[, quoin])
  - hide(e)

Channel Composition

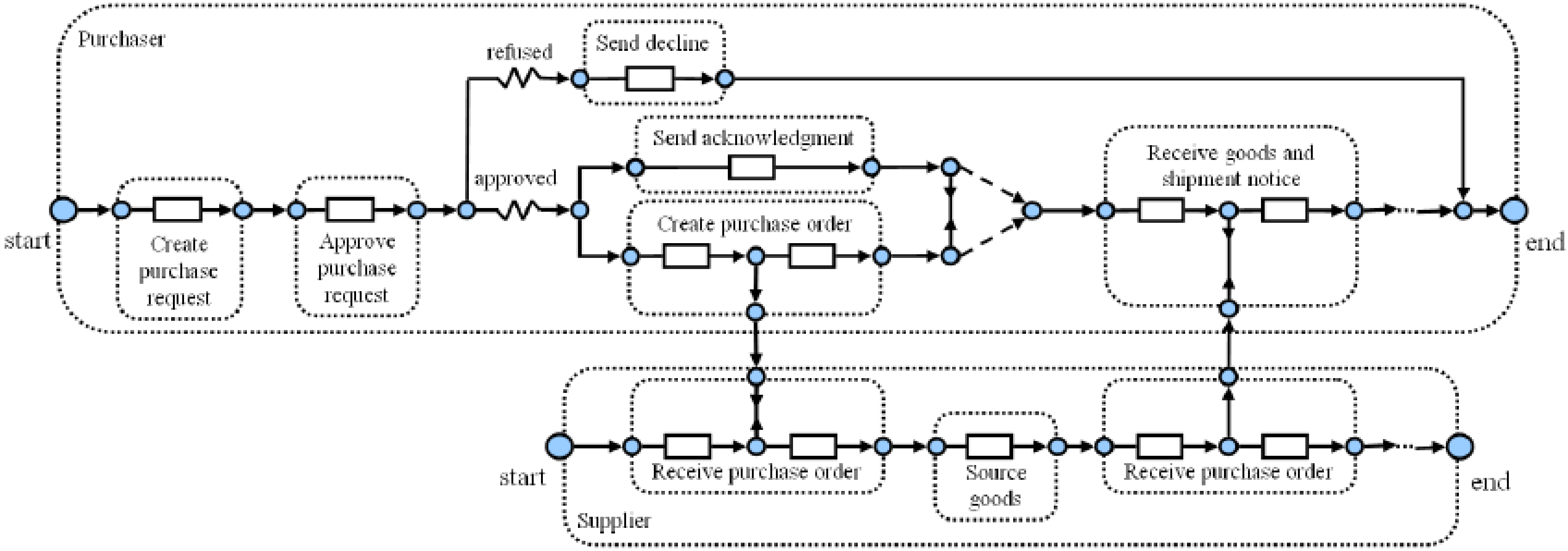
```
WCRegulator(n)
  ⟨a, x1⟩ = create(Sync)
  ⟨x2, b⟩ = create(Sync)
  ⟨x, y⟩ = create(SyncDrain)
  connect(x1)
  connect(x2)
  join(x, x1)
  join(x1, x2)
  hide(x)
  c = ⟨⟩
  for i = 1 to n do
    ⟨u, w⟩ = create(Sync)
    c = c ∘ ⟨u⟩
    connect(w)
    join(y, w)
  done
  hide(y)
  return ⟨a, b, c⟩
```



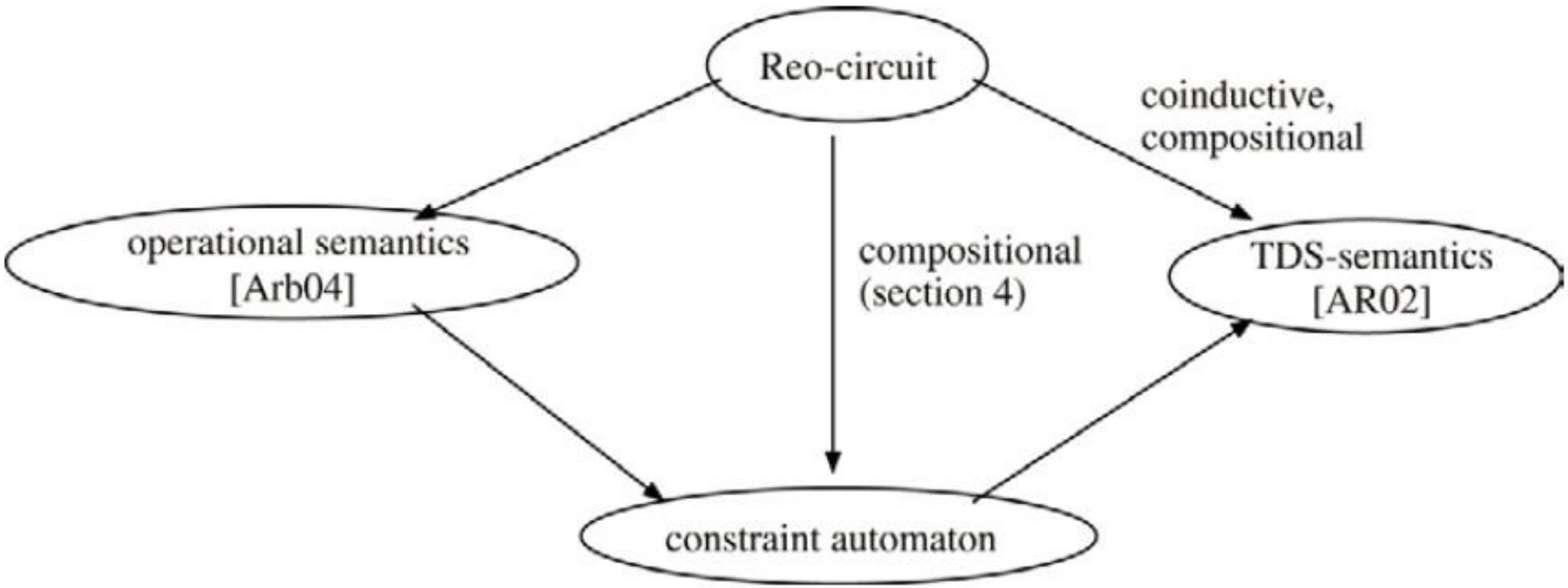


Real World Example

- Reo Circuit for the Purchase Order Scenario



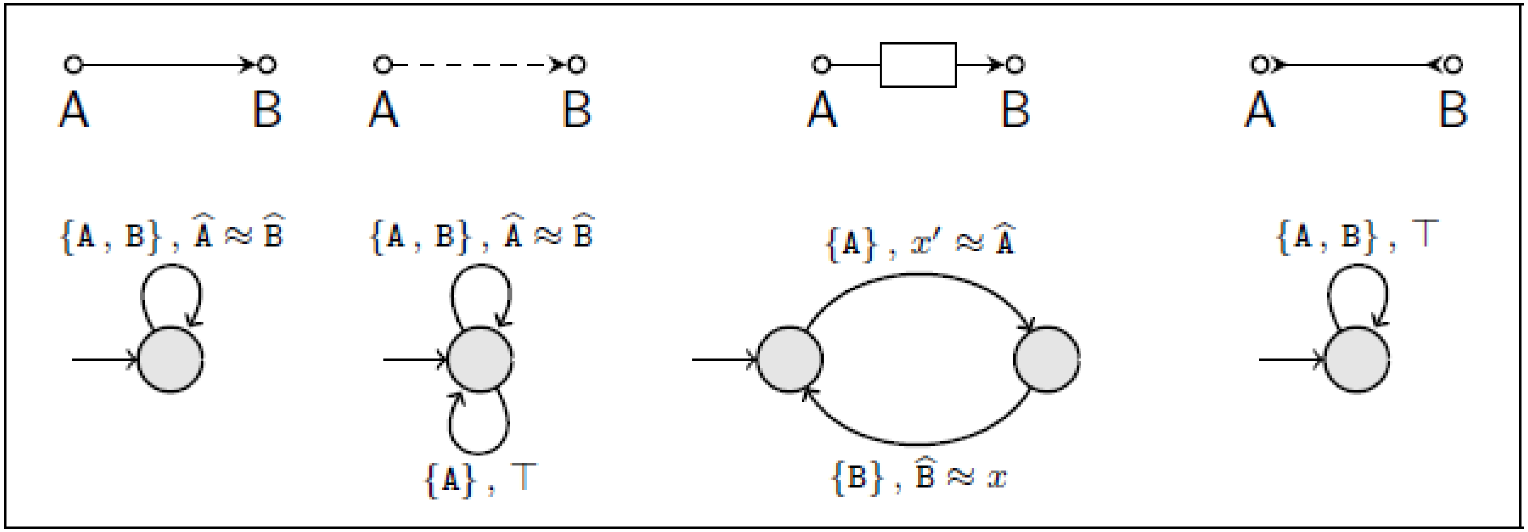
Semantics



$$q \xrightarrow{N,g} p$$
$$g \in DC(N, Data)$$
$$data(A) = d_A : A \in N$$

Semantics (cont.)

- Constraint Automata Examples
  - Synchronous
  - Lossy Synchronous
  - Asynchronous FIFO
  - Synchronous Drain



Semantics (cont.)

- Product Automata (Join)

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{0,1})$$

$$\mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{0,2})$$

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$$

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 p_1, \;\; q_2 \xrightarrow{N_2, g_2}_2 p_2, \;\; N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

$$\frac{q_1 \xrightarrow{N, g}_1 p_1, \;\; N \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle p_1, q_2 \rangle}$$

Semantics (cont.)

- Relation on Timed Data Streams (TDS)

$$TDS = \{ \langle \alpha, a \rangle \in Data^{\omega} \times \mathbb{R}_{+}^{\omega} \mid \forall n \geq 0 : a_n < a_{n+1} \text{ and } \lim_{n \to \infty} a_n = \infty \}$$

- Examples
  - Channels

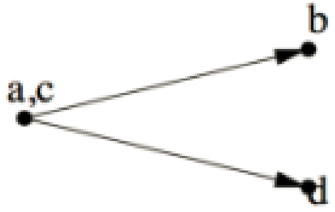
$$\langle \alpha, a \rangle \ \texttt{Sync} \ \langle \beta, b \rangle \equiv \alpha = \beta \wedge a = b$$

$$\langle \alpha, a \rangle \ \texttt{SyncDrain} \ \langle \beta, b \rangle \equiv a = b$$

$$\langle \alpha, a \rangle \ \texttt{FIFO1} \ \langle \beta, b \rangle \equiv \alpha = \beta \wedge a < b < a'$$

  - Replicator

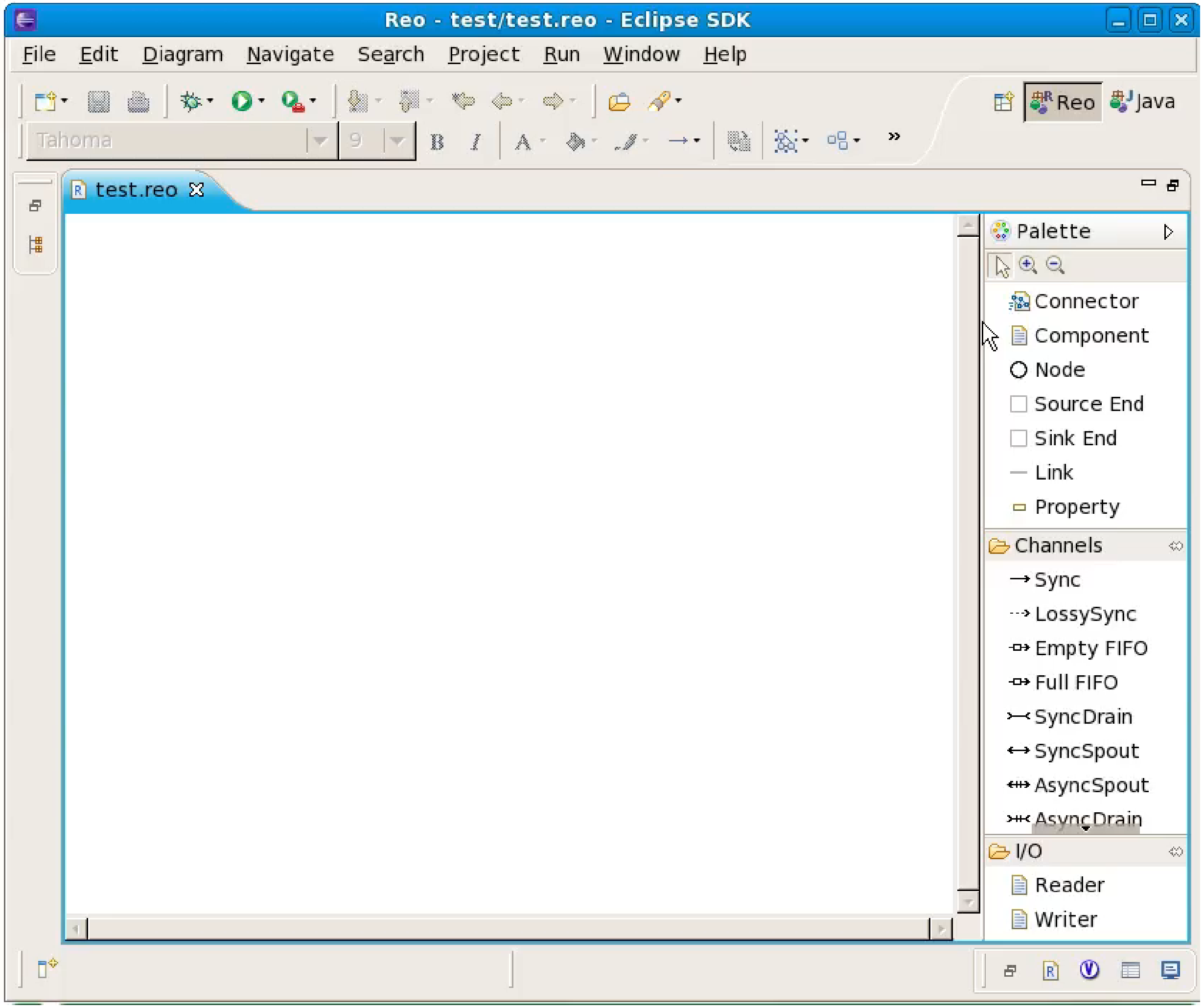
$$R(\langle \alpha, a \rangle; \langle \beta, b \rangle, \langle \gamma, c \rangle) \equiv \alpha = \beta = \gamma \wedge a = b = c$$



Tool

- Extensible Coordination Tools (ECT)
  - set of Eclipse Plugins
  - Graphical Editing of Reo connectors and constraint automata
  - Animation of Reo connectors
  - Code Generation from Reo connectors or constraint automata
  - Model Checking connectors using Vereofy and mCRL2

Tool


Reo - test/test.reo - Eclipse SDK
File Edit Diagram Navigate Search Project Run Window Help
Reo Java
Tahoma
test.reo
Palette
Connector
Component
Node
Source End
Sink End
Link
Property
Channels
Sync
LossySync
Empty FIFO
Full FIFO
SyncDrain
SyncSpout
AsyncSpout
AsyncDrain
I/O
Reader
Writer

Conclusion

- Reo is a language for compositional construction of interaction protocols
- Interaction is the only first-class concept in Reo
- Protocols manifest as a connectors
- In its graphical syntax, connectors are graphs
  - Data items flow through channels represented as edges
  - Boundary nodes permit (components to perform) I/O operations
- Formal semantics given as Constraint Automata (and various other formalisms)
- Tool support: draw, animate, verify, compile

References

[1] F. Arbab "Reo: A Channel-Based Coordination Model for Component Composition," Mathematical Structures in Computer Science, vol 14(03), pp. 329-366, 2004.

[2] C. Baier, M. Sirjani, F. Arbab, and J. Rutten "Modeling Component Connectors in Reo by Constraint Automata," Science of Computer Programming, vol 61(2), pp. 75-113, 2006.

[3] N. Kokash, and F. Arbab, "Formal Behavioal Modeling and Compliance Analysis for Service-Oriented Systems," Formal Methods for Components and Objects, pp. 21-41, 2008.

[4] M. R. Mousavi, M. Sirjani, and F. Arbab, "Formal Semantics and Analysis of Component Connectors in Reo," Electronic Notes in Theoretical Computer Science, vol. 154, pp. 83–99, 2006.

[5] Reo homepage, http://reo.project.cwi.nl/reo/

Animations from http://reo.project.cwi.nl/webreo/

Any Questions?