

*Luís Soares Barbosa*

*This lecture concludes the course with an introduction to the coordination of reactive components. For this we resort to the Reo model of exogenous coordination proposed by F. Arbab. Its methodology, semantics and toolset is presented in detail.*

## 1 Composition as coordination

Lecture notes for *Arquitetura e Cálculo*, MEI profile in *Formal Methods in Software Engineering*, 2014-15.

## Composition in object-orientated software

The operations/methods provided by a class-interface impose a **tight semantic binding** which, at the inter-component level

- Weakens independence of components;
- Contributes to breaking of encapsulation;
- Tightens component inter-dependence.

## Composition in component-based software

- CBD retains the basic encapsulation of **data** and **code** principle to increase modularity
- ... but shifts the emphasis from **class inheritance** to **object composition**
- to avoid interference between inheritance and encapsulation and pave the way to a development methodology based on **third-party assembly** of components

## Composition in component-based software

- a **palette** of computational units (eg **robust** collections of objects) treated as **black boxes**
- and a **canvas** into which they can be dropped
- **connections** are established by drawing **wires**
- inter-component communication is through messages that invoke remote methods, typically given some suitable triggering condition on the source.

## Composition in coordination-based software

- a **palette** of computational units (eg **robust** collections of objects) treated as **black boxes**
- and a **canvas** into which they can be dropped
- **connections** are established by **specific devices** (with complex logic, memory, etc)
- inter-component communication becomes **anonymous** and **externally coordinated**

## Composition as coordination

### Example scenario

What to do when the user starts scanning bar-codes at a pace that exceeds the rate at which the LCD can display the product names?

- Do we force the bar-code scanner to wait for when its output port is not busy to read another bar-code? (**forced synchronisation**)
- Do we buffer the excess data and display it on a read first, display first order?
- Do we disregard bar codes that are input while the LCD is busy displaying a previous product name?
- Do we combine the approaches 2, and 3 and provide a limited buffer where a finite amount of bar codes can be buffered while the LCD is busy displaying a product name?

## Composition as coordination

### Lesson learned

Building a system out of independent components **does not simply amount to wiring properly their ports together.**

Special **glue** code is necessary to **coordinate their interactions.**

### Coordination

- **Endogenous:** provide primitives that must be incorporated within a computation for its coordination
- **Exogenous:** ensure that the conceptual separation between computation and coordination is suitably respected

Composition as coordination   Introduction to REO   Examples   Semantics   Further examples

## Coordination


Carriero and Gelernter, 1986

Coordination is the process of building programs by gluing together active pieces

- distinguish computation from interaction (in massive parallel networks)
- focus on the emergent behaviour
- amenable to external, third-party control

Peter Wegner, 2000

Coordination is constrained interaction




Composition as coordination   Introduction to REO   Examples   Semantics   Further examples

## Composition in service-oriented software

'entails the need of managing an open-ended structure of autonomous components, possibly distributed and highly heterogeneous.

This means developing software components that are autonomous and can be interconnected with other components, software or otherwise, and managing the interconnections themselves as new components may be required to join in and others to be removed.' (Fiadeiro, 05)

- interaction as a first-class citizen
- composition as exogenous coordination



## Composition in service-oriented software

- interacting components **need not know each other**.  
(cf traditionally communication is targeted, making the sender semantically dependent on (the scheme used to identify) the receiver)
- communication becomes **anonymous**: components exchange identifiable sequences of passive messages with the environment only
- therefore **third parties can coordinate interactions** between senders and receivers of their own choice

## Composition as coordination

### Components

- loci of **computation**
- are kept independent of each other and of their environment
- Components communicate with the environment only through **read** and **write** operations on the connector **ends** (or **ports**), possibly according some **behavioural** interface description.

## Composition as coordination

### Connectors

- act as **interaction controllers**: the **glue code** that makes components interact  
i.e., they coordinate the activities of individual components to ensure their proper interaction with one another to form a coherent system that behaves according to its requirements
- have no relevant role in the computation carried out by the overall system: they are component-independent and agnostic wrt the underlying computation model
- provide systems-independent interaction protocols (whereas components provide systems-specific functionality)
- ... built **compositionally**.
- but traditionally, glue code is the most rigid, component specific, special purpose software in component based systems!

## 2 Introduction to Reo

Composition as coordination   Introduction to Reo   Examples   Semantics   Further examples

REO

`reo.project.cwi.nl/`

- A compositional, **connector**-based coordination language for plugging together **components** in an **exogenous** discipline (from outside and without participants' knowledge);
- Primitive circuit-like connectors are composed to build complex coordination patterns
- Key concepts are synchrony ('**happens together**') and mutual exclusion;
- Connectors implement interaction protocols (dealing with aspects of concurrency, buffering, ordering, data flow and manipulation);

◀ ◻ ▶ ◀ ⌂ ▶ ◀ ⏪ ▶ ◀ ⏩ ▶ ◀ 🔍 ▶

Composition as coordination   Introduction to Reo   Examples   Semantics   Further examples

REO

`reo.project.cwi.nl/`

- Several formal **semantics**:
  - **relations between timed streams** (2002)
  - **constraint automata** (2004) and several variants
  - **colours** (2004) to capture **context awareness**
  - **reo automata** (2009) and **intensional automata** (2010)
  - ...
- Eclipse **toolset** available

◀ ◻ ▶ ◀ ⌂ ▶ ◀ ⏪ ▶ ◀ ⏩ ▶ ◀ 🔍 ▶



Composition as coordination   Introduction to REO   Examples   Semantics   Further examples

# REO

[reo.project.cwi.nl/](http://reo.project.cwi.nl/)

- Several formal **semantics**:
  - **relations between timed streams** (2002)
  - **constraint automata** (2004) and several variants
  - **colours** (2004) to capture **context awareness**
  - **reo automata** (2009) and **intensional automata** (2010)
  - ...
- Eclipse **toolset** available

Navigation icons: back, forward, search, etc.

Composition as coordination   Introduction to REO   Examples   Semantics   Further examples

# REO connectors

Characterized by

- a number of **ends** and a **constraint** which defines an **interaction protocol** through these ends

**Ends**

- **source** end: through which data enters the connector
- **sink** end: through which data comes out of the connector

**Examples (channels)**

<i>Sync</i>	<i>SyncDrain</i>	<i>SyncSpout</i>	<i>LossySync</i>
<i>AsyncDrain</i>	<i>AsyncSpout</i>	<i>FIFO<sub>1</sub></i>	<i>FIFO<sub>1</sub>(x)</i>

Navigation icons: back, forward, search, etc.

Composition as coordination
Introduction to REO
Examples
Semantics
Further examples

## REO connectors


Characterized by


- a number of **ends** and a **constraint** which defines an **interaction protocol** through these ends


Ends


- source** end: through which data enters the connector
- sink** end: through which data comes out of the connector


Examples (channels)



  
Sync



  
SyncDrain



  
SyncSpout


  
LossySync


  
AsyncDrain


  
AsyncSpout


  
FIFO<sub>1</sub>


  
FIFO<sub>1</sub>(x)

Composition as coordination
Introduction to REO
Examples
Semantics
Further examples

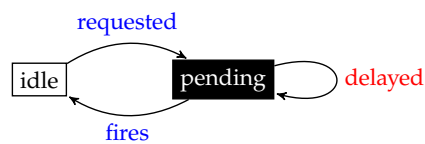
## Connector configurations

The configuration of a connector is the (abstract) structure that describes its global state:

- internal**: describes the connector **memory**
- external**: describes the **environment** in which the connector is currently being evaluated, i.e. the status of its **ports**

## Ports

- are the **only** medium for interacting with a connector (through **io operations**)
- a connector can have at most one component connected at each of its ports performing io requests
- upon the arrival of an io operation request at one of its ports, the connector decides whether the io operation can be fired or has to be delayed (becoming **pending**) because the **interaction constraints** that the connector imposes are not satisfiable in the present configuration.



## Connector memory

- Connectors with **memory** can store data in its buffer cells
- A buffer cell has two configurations: **full** or **empty**
- Connectors without **memory** cannot store data: any datum either flows through the connector to another port where it is output or it is lost





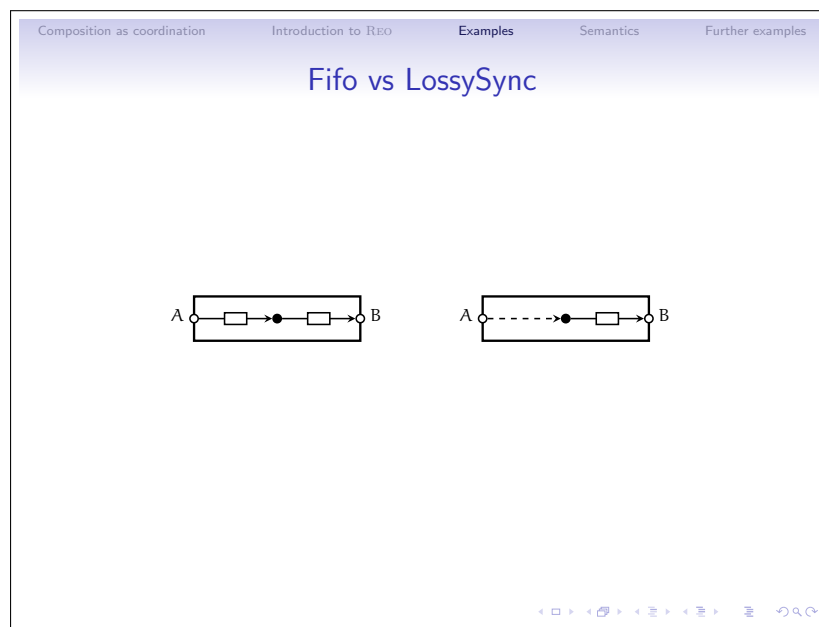
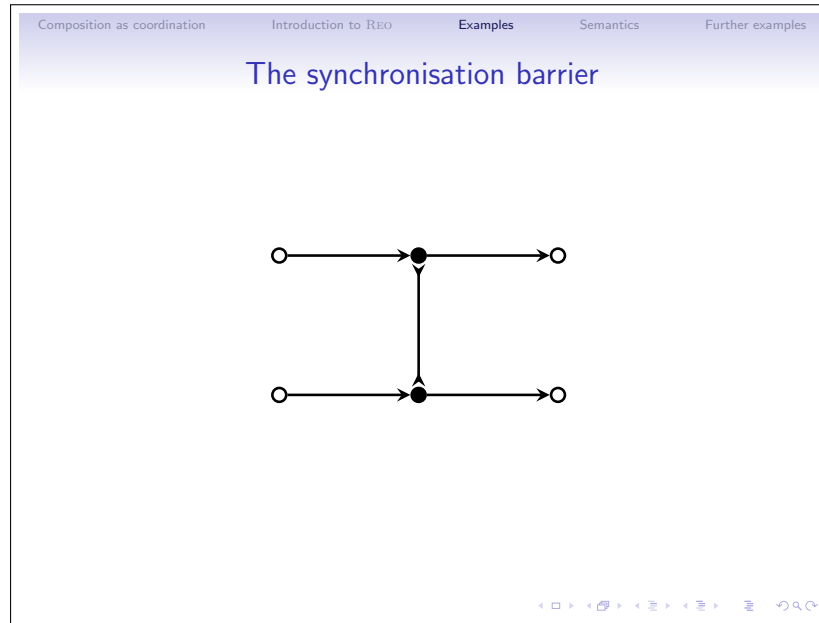
## Components

- active (computational) entities with a fixed interface that consists of a number of source and sink ends
- often (but **not necessarily**) interpreted as **black boxes**, i.e., no assumptions about their behavior
- actually, for analysis it is often beneficial to take into account the behavior of components (e.g. to detect potential deadlocks or to validate temporal properties) — may be annotated with a specification that reflects its behaviour

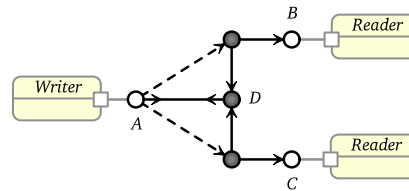
### Write and Take operations



### 3 Examples

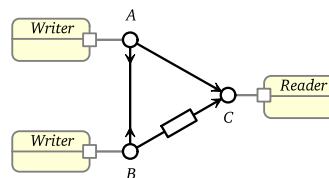


## The exclusive router



- routes data items synchronously from the source to exactly one of the two sinks;
- if both of them are ready to accept data, the choice of where the data item goes is made non-deterministically (merge goes without a priority)

## The alternator



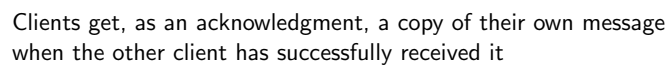
- enforces an ordered output of the data items provided by the two sources
- inputs synchronized through a **synchronous drain**
- the **FIFO1** stores the data item and makes it available in the next execution step; and guarantees alternation (why?)



## Messages exchanged through two buffered channels



### Messenger with automatic acknowledgments



## 4 Semantics

Composition as coordination   Introduction to REO   Examples   **Semantics**   Further examples

### Timed data streams

**Time streams**  
constrained streams over (positive) real numbers, representing moments in time such that

- **strictly increasing**:  $a(i) < a(i+1)$
- **progressive**:  $\forall n \exists N \ a(n) > N$

**Timed data stream**  
pair  $\langle \alpha, a \rangle$  consisting of a **data stream**  $\alpha$  and a **time stream**  $a$ , with the interpretation that for all  $i \in \mathbb{N}$ , the input/output of data item  $\alpha(i)$  occurs at time  $a(i)$

Composition as coordination   Introduction to REO   Examples   **Semantics**   Further examples

### Timed data streams

Formally,

$$TDS = \{ \langle \alpha, a \rangle \in Data^\omega \times \mathbf{R}_+^\omega \mid \forall i \geq 0. \ a(i) < a(i+1) \text{ and } \lim_{i \rightarrow \infty} a(i) = \infty \}$$

**Notes**

- A timed data stream is associated to each connector port
- No distinction between input and output actions

## Timed data streams

### Connectors

are **relations** over timed data streams:

$$\begin{aligned}\langle \alpha, a \rangle \llbracket \text{Sync} \rrbracket \langle \beta, b \rangle &\equiv \langle \alpha, a \rangle = \langle \beta, b \rangle \\ \langle \alpha, a \rangle \llbracket \text{FIFO} \rrbracket \langle \beta, b \rangle &\equiv \alpha = \beta \wedge a < b \\ \langle \alpha, a \rangle \llbracket \text{FIFO}_1 \rrbracket \langle \beta, b \rangle &\equiv \alpha = \beta \wedge a < b < a'\end{aligned}$$

- coalgebraic semantics [Arbab, Rutten, 2002; Arbab 2003] with incipient calculus
- cannot capture **context-awareness**

## Constraint automata

Automata labelled by

- a **data constraint** which represents a set of data assignments to port names

$$g ::= \text{true} \mid d_A = v \mid g_1 \vee g_2 \mid \neg g$$

**Note:** other constraints, as

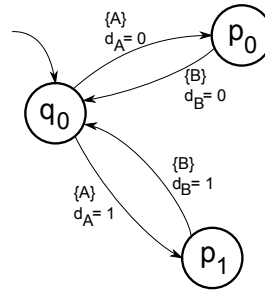
$$d_A = d_B \stackrel{\text{abv}}{=} \bigvee_{d \in \text{Data}} (d_A = d \wedge d_B = d) \text{ are derived.}$$

- a **name set** which represents the set of port names at which io can occur

States represent the configurations of the corresponding connector, while transitions encode its maximally-parallel stepwise behavior.

## Constraint automata

### Example: Fifo1



## Constraint automata

### Definition

$$A = \langle S, N, \rightarrow, S_0 \rangle$$

- $S$  is a set of states
- $S_0 \subseteq S$  is the set of initial states
- $N$  is a (finite) set of (port) names
- $\rightarrow: S \times \mathcal{P}N \times DC \times S$  such that  $s \xrightarrow{P, g} s'$  iff
  1.  $P \neq \emptyset$
  2.  $g \in DC(P, Data)$   
 ( $DC(P, Data)$  is the set of data constraints over  $Data$  and  $P$ )



## Constraint automata

### Acceptors for timed streams

Given  $A$  and  $t \in TDS^N$  as its input, find out whether  $t$  describes a possible data flow of  $A$

- $A$  starts in one of its initial states and waits until data items occur at some of its io ports
- Data occurring at a subset of ports triggers checking the guard
- ... choose a transition with a validated guard
- ... if no data constraint is fulfilled then  $A$  rejects  $t$

Accepted language is composed by all input streams that have at least one non-rejecting run in  $A$

## Constraint automata as a semantics for Reo

- cannot capture context-awareness [Baier, Sirjani, Arbab, Rutten 2006], but forms the basis for more elaborated models (eg, Reo automata)
- captures all behavior alternatives of a connector; useful to generate a state-machine implementing the connector's behavior
- basis for several tools, including the model checker Vereofy [Kluppelholz, Baier 2007]

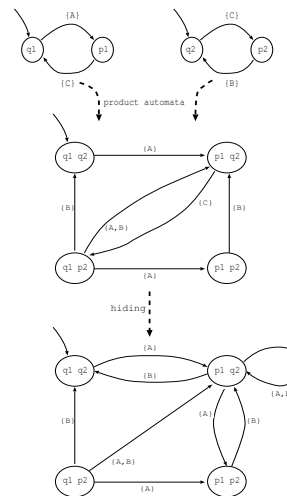


## Constraint automata as a semantics for Reo

### Connector construction

A 2-bounded FIFO obtained from two 1-bounded FIFO channels via product and hiding (assume  $Data = \{d\}$  for simplicity)

## Constraint automata as a semantics for Reo

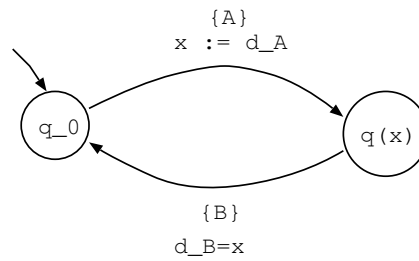




## Parametrized constraint automata

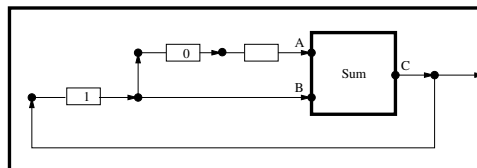
States are parametric on data values ... therefore capturing complex constraint automata emerging from data-dependencies

Example: 1 bounded FIFO



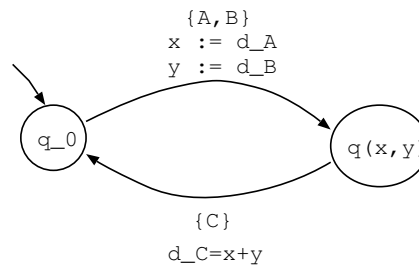
## Parametrized constraint automata

Example: Fibonacci generator



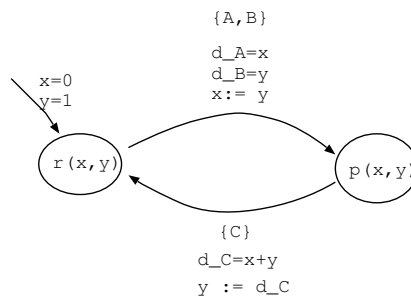
## Parametrized constraint automata

### Example: Fibonacci generator (Sum)



## Parametrized constraint automata

### Example: Fibonacci generator (complete)



## Constraint automata: bisimulation

### Definition

A **bisimulation** on a constraint automata  $A = \langle S, N, \rightarrow, S_0 \rangle$  is an equivalence relation  $R$  on  $S$  such that for all pairs  $\langle s, s' \rangle$ , all  $R$ -induced equivalence classes  $P \in S/R$  and every  $Ns \subseteq N$ ,

$$dc(s, Ns, P) = dc(s', Ns, P)$$

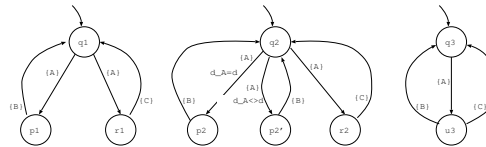
where

$$dc(s, Ns, P) = \bigvee \{g \mid s \xrightarrow{Ns, g} s' \text{ for some } s' \in P\}$$

i.e., the **weakest** data constraint ensuring a  $Ns$ -transition from  $s$  to a state in  $P$ .

## Constraint automata: bisimulation

### Example



The equivalence  $R$  induced by partition

$$S/R = \{\{q_1, q_2\}, \{q_3\}, \{p_1, p_2, p_2'\}, \{r_1, r_2\}, \{u_3\}\}$$

is a bisimulation. Why?

## Constraint automata: bisimulation

### Bisimilarity

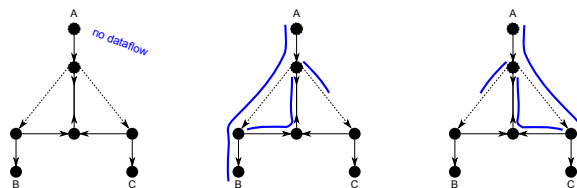
As usual, two sates are **bisimilar** if contained in a bisimulation.

### Theorem

Bisimilarity is strictly finer than language equivalence (TDS acceptance), but for deterministic automata for which they coincide.

## Coulorings

Based on the set of all of **dataflow alternatives** of the connector, represented by different colours meaning **data flowing** and **no data flowing**



## 5 Further examples

Composition as coordination      Introduction to REO      Examples      Semantics      Further examples

## Merger

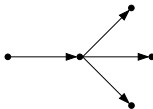
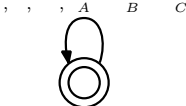

**Reo circuit**

**Semantics**

$$\langle (\alpha, \rangle, \langle \beta, \rangle; \langle , \rangle) \equiv$$

$$(0) \neq (0) \wedge$$

$$\begin{cases} \alpha(0) = (0) \wedge (0) = (0) \wedge & \langle (\alpha', \rangle, \langle \beta, \rangle; \langle ', \rangle) \quad \text{if } (0) < (0) \\ \beta(0) = (0) \wedge (0) = (0) \wedge & \langle (\alpha, \rangle, \langle \beta', \rangle; \langle ', \rangle) \quad \text{if } (0) < (0) \end{cases}$$

Composition as coordination	Introduction to Reo	Examples	Semantics	Further examples
Replicator				
Reo circuit				
				
Semantics				
$(\langle \alpha, \rangle; \langle \beta, \rangle, \langle \ , \rangle) \equiv \alpha = \beta = \ \wedge \ = \ =$				
				
				



Composition as coordination
Introduction to REO
Examples
Semantics
Further examples

## Ordering

Reo circuit

Semantics

$$\begin{aligned}
OC(\langle \alpha, \rangle, \langle \beta, \rangle; \langle \cdot, \rangle) &\equiv \\
\alpha(0) = (0) \wedge \beta(0) = (1) \wedge (0) = (0) &= (0) \wedge (1) = (1) > (1) \wedge \\
OC(\langle \alpha', \rangle, \langle \beta', \rangle; \langle \cdot, \rangle) &= (1)
\end{aligned}$$

Navigation icons

Composition as coordination
Introduction to REO
Examples
Semantics
Further examples

## Sequencer

Reo circuit

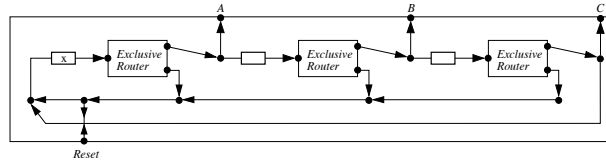
Semantics

$$SQ(\langle \alpha, \rangle, \langle \beta, \rangle, \langle \cdot, \rangle) \equiv \langle \cdot \rangle < \langle \cdot \rangle < \langle \cdot \rangle$$

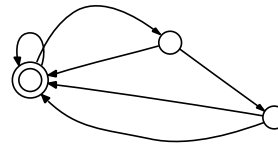
Navigation icons

## Sequencer with reset

### Reo circuit

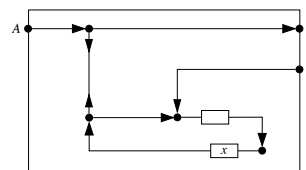


### Semantics



## Inhibitor

### Reo circuit



### Semantics

$$Ih(\langle \alpha, \rangle, \langle \cdot, \rangle; \langle \beta, \rangle) \equiv \begin{cases} (0) = (0) \wedge \alpha(0) = \beta(0) \wedge Ih(\langle \alpha', \rangle, \langle \cdot, \rangle; \langle \beta', \rangle) & \text{if } (0) < (0) \\ \alpha = \beta = \cdot & \text{if } (0) < (0) \end{cases}$$

