Software architecture for reactive systems: the coordination perspective (An introduction to REO)

Luís S. Barbosa

DI-CCTC
Universidade do Minho
Braga, Portugal

May 2011

Object-orientation

- In OO the architecture is implicit: source code exposes class hierarchies but not the run-time interaction and configuration
- Objects are wired at a very low level and the description of the wiring patterns is distributed among them
- The semantics of method invocation is heavy and non-trivial:
    - The caller must know the callee and the caller must know the method.
    - The callee must (pretend) to interpret the message.
    - The caller suspends while the callee (pretends to) perform the method and resumes when the callee returns a result.

Object-orientation

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.

Component-orientation

- 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

Component-orientation

$\boxed{\text{CBD: the visual metaphor}}$

- 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.

Software as a service

'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 as coordination

- 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

Connectors

- Connectors are interaction controllers: the glue code that makes components interact
- Traditionally, glue code is the most rigid, component specific, special purpose software in component based systems!
- ... component-independent and agnostic, with a well-defined semantics
- ... built compositionally.
- Components communicate with the environment only through read and write operations on the connector ends (or ports), possibly according some behavioural interface description.

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

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);

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

Reo connectors

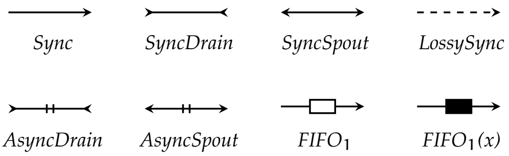
Characterized by

- a number of ends and a constraint defines its interaction protocol through these ends

Ends

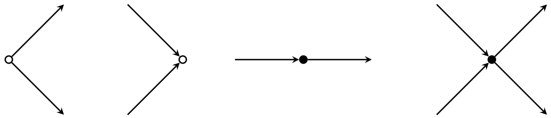
- source end: through which data enters the connector
- sink end: through which data comes out of the connector

Examples (channels)



Connector composition

Connectors are composed by conjoining their ends to form nodes with multiple ends



Connector composition

Nodes

- source node: superposes only source ends and atomically copies incoming data items to all of its outgoing source ends
- sink node: superposes only sink ends and acts as a non-deterministic merger, randomly choosing a data item from one of the sink ends for delivery
- mixed node: combines both acting as pumping station by atomically consuming a data item from one sink end and replicating it to all source ends ($1 : n$ synchronization)

Note: synchrony propagates through connectors

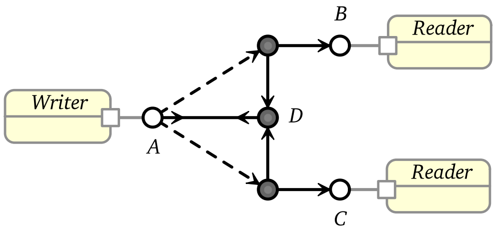
... because nodes do not perform any buffering

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

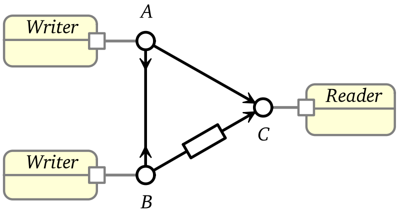
Write and Take operations

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)

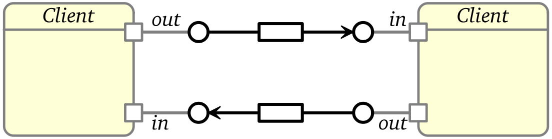
The alternator



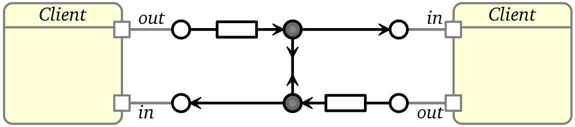
- enforces an ordered output of the data items provided by the two sources
- inputs synchronized through a synchronous drain
- the FIFO1 stores the data item and makes it available in the next execution step; and guarantees alternation (why?)

Messenger patterns

Messages exchanged through two buffered channels

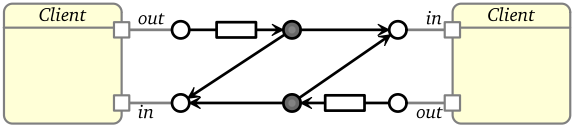


Message retrievals are synchronized



Messenger patterns

Messenger with automatic acknowledgments



Clients get, as an acknowledgment, a copy of their own message when the other client has successfully received it

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 α and a time stream a, with the interpretation that for for all $i \in \boldsymbol{N}$, the input/output of data item $\alpha(i)$ occurs at time $a(i)$

Timed data streams

Connectors

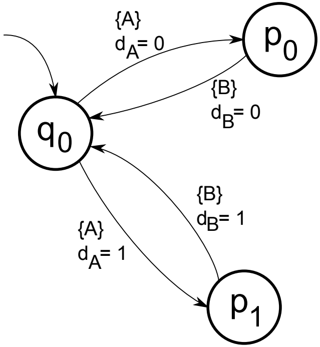
are relations over timed data streams:

$$\begin{aligned}
\langle \alpha, a \rangle \, [\![\mathsf{Sync}]\!] \, \langle \beta, b \rangle &\;\Leftrightarrow\; \langle \alpha, a \rangle = \langle \beta, b \rangle \\
\langle \alpha, a \rangle \, [\![\mathsf{FIFO}]\!] \, \langle \beta, b \rangle &\;\Leftrightarrow\; \alpha = \beta \wedge a < b \\
\langle \alpha, a \rangle \, [\![\mathsf{FIFO}_1]\!] \, \langle \beta, b \rangle &\;\Leftrightarrow\; \alpha = \beta \wedge a < b < a'
\end{aligned}$$

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

Constraint automata

The states represent the configurations of the corresponding circuit, while transitions encode its maximally-parallel stepwise behavior.

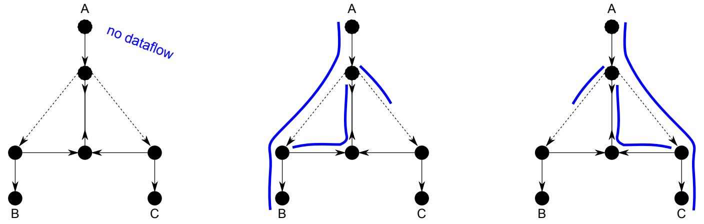


Constraint automata

- 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 statemachine implementing the connector's behavior
- basis for several tools, including models checker Vereofy [Kluppelholz, Baier 2007]

Coulorings

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



Coulorings

- first to capture context-awareness [Clarke, Costa, Arbab 2007]
- basis for the animation tools
- calculus???

Concluding

- tools ... & case studies
- several semantic models ... & incipient calculus
- extensions: timed, stochastic, QoS annotated