Architectural design: the coordination perspective

José Proença
HASLab - INESC TEC & UM
Arquitectura e Cálculo 2015-16
Software architecture for reactive systems

There is no general-purpose, universally tailored, approach to architectural design of complex and reactive systems.

In this course:

- introduce different models for reactive systems
- discuss their architectural design
- with (reasonable) tool support for modelling and analysis
Models of Concurrency

Traditional models are action-based
   Petri nets
   Work flow / Data flow
   Process algebra / calculi
   Actor models / Agents
   ...

Interaction appears as an implicit side-effect;
Makes coordination of interaction more difficult to
   Specify
   Verify
   Manipulate
   Reuse
Interaction with process algebra

act
  g, r, b, d : String  % synchronisation points
  print, genG, genR;

proc
  B = b(t) . print(t) . d("done") . B
  G = g(k) . genG(t) . d(j) . r(k) . G
  R = r(k) . genR(t) . g(k) . R

init
  G || R || B || g("token")

Model constructed by composing actions into more complex actions

Where is the INTERACTION?
Interaction with Object Oriented Software

- 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 method.
- The callee must (pretend) to **interpret** the message.
- The caller **suspending** while the callee (pretends to) perform the method and **resumes** when the callee returns a result.
Implicit interaction

Interaction (protocol) is implicit in action-based models of concurrency

Interaction is a by-product of processes executing their actions

Action $a$ of process A **collides** with action $b$ of process B

**Interaction** is the specific (timed) sequence of such collisions in a run

**Interaction protocol** is the (timed) sequence of the *intended* collisions in such a sequence.

---

**How can the sequence of intended collisions (the interaction protocol) can be**

- Manipulated?
- Verified?
- Debugged?
- Reused?
Interaction with components

Shift from class inheritance to object composition

Avoid interference between inheritance and encapsulation and pave the way to a development methodology based on third-party assembly of components

Move from an action-based to an interaction-based model of concurrency

- Black box computation units
- Canvas to drop them
- Connections via wires
Component coordination in Reo

- Exogenous coordination
- Compositional (channel based)

- Synchronous (atomic)
- Coordination is constrained interaction

[Peter Wegner, 2000]
Component coordination in Reo

**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

- Exogenous coordination
- Compositional (channel based)
- Synchronous (atomic)
- Coordination is constrained interaction

[Peter Wegner, 2000]
Discrete atomic steps

Ready to write!

No data yet...

Ready to receive!
Discrete atomic steps

**Go!**

**Take data.**

**Wait.**
Reo: Channel composition
Reo

- 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
- Protocols manifest as 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 (various formalisms - shown later)
- Tool support: draw, animate, verify, compile
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
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
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!
Reo connectors

- **Source end**: through which data enters the connector
- **Sink end**: through which data comes out of the connector

### Examples:

<table>
<thead>
<tr>
<th></th>
<th>Sync</th>
<th>SyncDrain</th>
<th>SyncSpout</th>
<th>LossySync</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>AsyncDrain</td>
<td>AsyncSpout</td>
<td>FIFO₁</td>
<td>FIFO₁(ᵪ)</td>
</tr>
</tbody>
</table>

---

The terms *source* and *sink* designate the senses of the ends of a channel from the point of view of the channel itself. Obviously, the sense of a channel end must be reversed from the point of a user of a channel, i.e., a component that performs an I/O operation on a channel end. Thus, a component writes to the source end of a channel and takes from the sink end of a channel.

---

1. Table 1 contains the Reo channel types we use throughout this thesis. We provide their formal semantics in the following chapters. At this stage, we give an informal...
Composing Reo connectors

Nodes: syntactic sugar for mergers and replicators

- Source nodes
- Sink nodes
- Mixed nodes

join source ends with sink ends

one to one

\[ a \longrightarrow b \otimes b \longrightarrow c = a \longrightarrow b \longrightarrow c \]
Reo eclipse toolset

get Eclipse

update site: http://reo.project.cwi.nl/update