Interaction-based Paradigm

José Proença
Seminar Series, 11 April 2019
2005
Lic. @ UMinho
Braga
Mathematics &
Computer Science

2011
PhD @ CWI
Amsterdam (NL)
Coordination
Formal methods
Concurrency
Software Engineering

2015
Postdoc @
KU Leuven (BE)
Programming languages
Variability
Wireless Sensor Netw.
Reactive programming

2019
Postdoc @ INESC TEC
Braga
Softw. Architectures
Design Calculi

Feb’19
Postdoc @ CISTER
Outline

What is Coordination?
  Context & motivation
  Reo coordination language

Recent research tracks
  Analysis tools
  Composing families of Timed Automata
  Composing tasks in a RTOS
Software architecture for reactive systems

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

How to build and maintain a system built out of a composition of reactive entities.
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

```plaintext
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) . b(t) . d(j) . r(k) . G
  R = r(k) . genR(t) . b(t) . d(j) . g(k) . R

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

Model constructed by composing actions into more complex actions.

Where is the interaction?
Interaction with shared memory

- Where is the green text computed?
- Where is the red text computed?
- Where is the text printed?
- Where is the protocol?
- What determines who goes first?
- What determines producers alternate?

```java
private final Semaphore bufferSemaphore = new Semaphore(1);
private final Semaphore redSemaphore = new Semaphore(0);
private final Semaphore greenSemaphore = new Semaphore(1);
private String buffer = Empty;

while (true) {
    sleep(5000);
    greenText = ...;
    greenSemaphore.acquire();
    bufferSemaphore.acquire();
    buffer = greenText;
    bufferSemaphore.release();
    redSemaphore.release();
}

while (true) {
    sleep(5000);
    redText = ...;
    redSemaphore.acquire();
    bufferSemaphore.acquire();
    buffer = redText;
    bufferSemaphore.release();
    greenSemaphore.release();
}

while (true) {
    sleep(4000);
    bufferSemaphore.acquire();
    if(buffer != EMPTY) {
        println(buffer);
        buffer = EMPTY;
    }
    bufferSemaphore.release();
}
```

**Producer 1**
while (true) {
    sleep(5000);
    greenText = ...;
    greenSemaphore.acquire();
    bufferSemaphore.acquire();
    buffer = greenText;
    bufferSemaphore.release();
    redSemaphore.release();
}

**Producer 2**
while (true) {
    sleep(5000);
    redText = ...;
    redSemaphore.acquire();
    bufferSemaphore.acquire();
    buffer = redText;
    bufferSemaphore.release();
    greenSemaphore.release();
}

**Consumer**
while (true) {
    sleep(4000);
    bufferSemaphore.acquire();
    if(buffer != EMPTY) {
        println(buffer);
        buffer = EMPTY;
    }
    bufferSemaphore.release();
}
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 *intended* and the *coincidental* be differentiated?

How can the sequence of *intended collisions* (the interaction protocol) 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
Discrete atomic steps

Ready to write!

Ready to receive!

No data yet...
Discrete atomic steps

Go!

Take data

Wait

W_r → connector → Rd

W_r → connector → W_r

W_r → connector → W_r
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 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** (various formalisms - shown later)
- **Tool support**: draw, animate, verify, compile
Reo connectors

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

Examples:

- **Sync**
- **SyncDrain**
- **SyncSpout**
- **LossySync**
- **AsyncDrain**
- **AsyncSpout**
- **FIFO**
- **FIFO** (x)
Composing Reo connectors

\[ a \quad \text{-----} \quad b \quad \Box \quad b \quad \xrightarrow{\text{sink ends}} \quad c \]

\[ = a \quad \bullet \quad \text{-----} \quad \bullet \quad b \quad \bullet \quad \xrightarrow{\text{source ends}} \quad c \]

join

source ends

with

sink ends

one to one

Nodes act as pumping-stations

merger
duplicator
Composing Reo connectors
Reo eclipse toolset

Eclipse plug-in

http://reo.project.cwi.nl/update
Reo Live

JavaScript

https://reolanguage.github.io/ReoLive/snapshot/
Reo semantics

Jongmans and Arbab 2012

Overview of Thirty Semantic Formalisms for Reo
Reo semantics

- **Coalgebraic models**
  - Timed data streams
  - Record streams

- **Coloring models**
  - Two colors
  - Three colors
  - Tile models

- **Other models**
  - Process algebra
  - Constraints
  - Petri nets & intuitionistic logic
  - Unifying theories of programming
  - Structural operational semantics

- **Operational models**
  - Constraint automata
  - Variants of constraint automata
    - Port automata
    - Timed
    - Probabilistic
    - Continuous-time
    - Quantitative
    - Resource-sensitive timed
    - Transactional
  - Context-sensitive automata
    - Büchi automata
    - Reo automata
    - Intentional automata
    - Action constraint automata
    - Behavioral automata
  - Structural operational semantics
2CM : Coloring models with two colors [28, 29, 33]
3CM : Coloring models with three colors [28, 29, 33]
ABAR : Augmented BAR [39, 40]
ACA : Action CA [46]
BAR : Behavioral automata [61]
BA : Behaviorally ambiguous automata [61]
CA : Constraint automata [10, 17]
CASM : CA with state memory [60]
CCA : Continuous-time CA [18]
Constr. : Propositional constraints [30, 31, 32]
GA : Guarded automata [20, 21]
IA : Intentional automata [33]
ITLL : Intuitionistic temporal linear logic [27]
LCA : Labeled CA [44]
mCRL2 : Process algebra [47, 48, 49]

PA : Port automata [45]
PCA : Probabilistic CA [15]
QCA : Quantitative CA [12, 53]
QIA : Quantitative IA [13]
RS : Record streams [38, 40]
rSTCA : Resource-sensitive timed CA [51]
SGA : Stochastic CA [56, 57]
SOS : Structural operational semantics [58]
SPCA : Simple PCA [15]
TCA : Timed CA [8, 9]
TDS : Timed data streams [4, 5, 14, 62]
Tiles : Tile models [11]
TNCA : Transactional CA [54]
UTP : Unifying theories of programming [55, 52]
ZSN : Zero-safe nets [27]
Outline

What is Coordination?
  Context & motivation
  Reo coordination language

Recent research tracks
  Analysis tools
  Composing families of Timed Automata
  Composing tasks in a RTOS
IFTA
interface feature timed automata

Timed Automata
IFTA

interface feature timed automata

Features + Timed Automata
IFTA

**interface** feature **timed automata**

Features + Interfaces (Composition) + Timed Automata
Programming real-time
With VirtuosoNext @ Altreonic

Visual IDE to compose tasks

Precise time traces on embedded SW
C Code

Hubs

C Code

Port  Event  Semaphore  Fifo
Data Event  Resource  Blackboard

...
Hubs ++

Automata semantics

Semaphore

\[ \mathcal{X} = \{ c : \mathbb{N} \} \]

\( \text{signal} \)
\[ \langle c < \text{MAXINT} \rangle \]
\[ c \leftarrow c + 1 \]

\( \text{test} \)
\[ \langle c > 1 \rangle \]
\[ c \leftarrow c - 1 \]

Communication between nodes?
Wrap up

What is Coordination?
- Context & motivation
- Reo coordination language

Recent research tracks
- Analysis tools
- Composing families of Timed Automata
- Composing tasks in a RTOS

More:
- Reactive Programming for IoT
- Dynamic Logics
- Hybrid Programs
  (continuous + discrete behaviour)