Ways to react:
Comparing Reactive Languages and Complex Event Processing

Alessandro Margara
High Performance Distributed Computing Group
Vrije Universiteit Amsterdam

Guido Salvaneschi
Software Technology Group,
University of Darmstadt
Ways to react:
Comparing Reactive Languages and Complex Event Processing

BACKGROUND
Reactive Applications

- **Continuous** external/internal $\rightarrow$ reactions
  - User input, network packet, interrupts, sensors, ...

![Diagram showing the process of observation, notification, processing, and reaction.]
Complex Event Processing

Specific case of stream processing to...
...detect high-level situations of interest:

**composite events**

- Starting from low-level events.

- Central role of time:
  - Timestamped events
  - Sequences
  - Temporal patterns

---

**Rule R**

```plaintext
define Fire(area: string)
from Smoke(area=$a) and
  Avg(Temp(area=$a).value within 5 min. from Smoke) > 45 and
  not Rain(area=$a) within 10 min. from Smoke
where area=Smoke.area
```
Reactive Languages

• Overcome the limitations of the Observer pattern
  – Not composable, no return type, Inversion of control, ...

  val tick = new Var(0)
  val hour = Signal{ tick() % 24 }
  val day = Signal{ (tick()/24) % 7 + 1 }

• Time changing values: signals or behaviors

• Graphical animations,
  robotics, sensor networks, ...
So far...

- Separate communities
  - OOPSLA, ECOOP, ICFP, ...
  - DEBS, MIDDLEWARE, ...

- Different analysis models
CEP & RLs

- It’s all about **reactive applications**

  ![Diagram](image)

  - Observation → Notification → Processing → Propagation → Reaction

- Common analysis framework?
- Synergies? Differences?
- Integration?
Ways to react:
Comparing Reactive Languages and Complex Event Processing

COMPARING CEP AND RLs
Back to Reactive Applications

<table>
<thead>
<tr>
<th></th>
<th>CEP</th>
<th>RL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OBSERVATION</strong></td>
<td>Generic Events</td>
<td>Value Changes</td>
</tr>
<tr>
<td><strong>NOTIFICATION</strong></td>
<td>Explicit – Push</td>
<td>Implicit – Push or Pull</td>
</tr>
<tr>
<td><strong>PROCESSING</strong></td>
<td>Rules (primitive E $\rightarrow$ composite E)</td>
<td>Expressions (signals $\rightarrow$ signals)</td>
</tr>
<tr>
<td><strong>PROPAGATION</strong></td>
<td>Explicit – Multicast – Push</td>
<td>Implicit – Multicast – Push or Pull</td>
</tr>
<tr>
<td><strong>REACTION</strong></td>
<td>Generic Procedures – User-Defined</td>
<td>Value Changes</td>
</tr>
</tbody>
</table>
Language Expressiveness

Declarative language

<table>
<thead>
<tr>
<th>Input: history of event occurrences</th>
<th>Input: signals that hold a value at any point in time and change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: time-annotated history of composite events</td>
<td>Output: time changing value</td>
</tr>
</tbody>
</table>

• Time and history are central in CEP
  – However:
    • `Signal.last(n)`
    • `Signal.delay(5)`
## Composability

<table>
<thead>
<tr>
<th>Similar!</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Event hierarchy of events</th>
<th>Event hierarchy of signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules can compose complex events</td>
<td>Event expressions compose signals</td>
</tr>
<tr>
<td>Usually no difference between rules on primitive events and rules on composite events</td>
<td>Observable values (Vars) and Signals can appear in signal expressions</td>
</tr>
</tbody>
</table>
## Consistency

Users lack control on the order of evaluation

<table>
<thead>
<tr>
<th>Primitive events processed in timestamp order</th>
<th>Typically guarantee glitch freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite events are generated and propagated in timestamp order</td>
<td>Enforce correct propagation order in the nodes of the dependency graph</td>
</tr>
</tbody>
</table>

No guarantees when the computation escapes the controlled propagation

\[
a = \text{Signal}\{ \text{b()} + \text{c()} \}
\]
Performance

Primary focus!
- Rate of input events
- Number of deployed rules
- Number of event sources/receivers

Less attention
Primary focus on language abstractions

Optimizations:
- Rule rewriting techniques
- Sharing operators among rules
- Algorithms for parallel hardware

Optimizations:
- Lowering
- GADTs for dynamic optimization
- Incrementalization

• Henrik Nilsson, Dynamic optimization for functional reactive programming using generalized algebraic data types, ICFP ’05.
## Distribution

<table>
<thead>
<tr>
<th>CEP server collects and distributes events to the clients</th>
<th>Not much considered yet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load over multiple machines</td>
<td>AmbientTalk/R</td>
</tr>
<tr>
<td></td>
<td>Ongoing: Distributed + glitch free</td>
</tr>
</tbody>
</table>

### Optimizations:
- Rule rewriting
- Selections close to the sources

Minimize exchanged messages

---

Safety

Type casts at the boundaries

- Stream processing: table-like approach
- Event-based systems: attribute-value pairs

Reactive abstractions checked by the compiler

Advanced use of types:
- Execution in bounded space
- Liveness guarantees

Embedding into high-level languages
- EventJava

Interaction with OO Features

Impedance mismatch

- Serialization

- Each event → object type
  Object fields go into the event

EventJava integrates into the OO model

Towards the integration with OO programming

FrTime, REScala, ...
Signals as objects fields
Abstract signals
Late binding
...

• Patrick Eugster and K. R. Jayaram. EventJava: An Extension of Java for Event Correlation. ECOOP’09
• Gregory H. Cooper and Shriram Krishnamurthi. Embedding dynamic dataflow in a call-by-value language., ESOP’06.
Ways to react:
Comparing Reactive Languages and Complex Event Processing

RESEARCH AGENDA
Research Agenda

Integration
- Interface operators: Events → Signals
- Common Language/Operators for Rules and Expressions

Research Agenda
- Expressiveness: Temporal Operators
- Performance: Algorithms
- Performance: Parallel Processing
- Distribution

Evolution
- Rls
- CEP
- Language Integration
- Safety
- Automated Reactions
Integration with OO

• OO + signals? Reactive objects?

• E.g. REScala: Signal-event interface operators

• Guido Salvaneschi, Gerold Hintz and Mira Mezini, REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications, MODULARITY AOSD 2014 (Accepted)
Integration of RLs and CEP

• Common set of operators for rules and signals
  – What about embedding CEP rules into RL languages?

• Operators on time?
  – Windows?
  – Joins?
Evolution of RLs

- **RLs**
  - Expressiveness
    - Temporal operators
  - Performance
    - Parallel processing
    - Distribution
    - Algorithms and opt. from CEP

- i3QL: optimizations for reactive incremental computations
Evolution of CEP

• CEP
  – Language integration
  – Safety

• “Stock price of IBM falls below $15.00 after a quarterly loss”

```java
class IBMMonitor {
  event earnings(String firm1, float profit),
      stockQuote(String firm, float price)
  when (earnings < stockQuote &&
        firm == firm1 && firm == "IBM" &&
        price < 15.00 && profit < 0 ) {
    triggerAlert("IBM",price)
  }
}
```
Ways to react: Comparing Reactive Languages and Complex Event Processing

- **CEP** and **RLs** both apply to **reactive applications**

- We need...
  - ...to bridge the two communities
  - ...to develop a common analysis framework
  - ...to envision a shared research roadmap
Ways to react:
Comparing Reactive Languages and Complex Event Processing

THANK YOU!
QUESTIONS?
Ways to react: Comparing Reactive Languages and Complex Event Processing

- **CEP** and **RLs** both apply to *reactive applications*

- We need...
  - ...to bridge the two communities
  - ...to develop a common analysis framework
  - ...to envision a shared research roadmap