Ways to react: Comparing Reactive Languages and Complex Event Processing

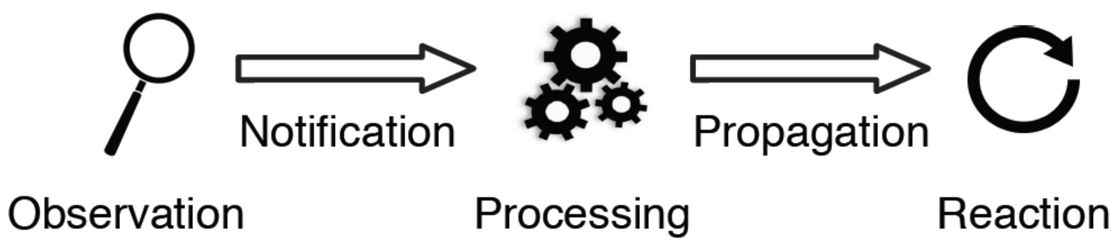
Alessandro Margara
High Performance Distributed Computing Group
Vrije Universiteit Amsterdam
Guido Salvaneschi
Software Technology Group,
University of Darmstadt

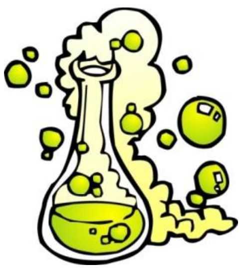
Ways to react:
Comparing Reactive Languages and Complex Event Processing

BACKGROUND

Reactive Applications

- **Continuous** external/internal → reactions
  - User input, network packet, interrupts, sensors, …





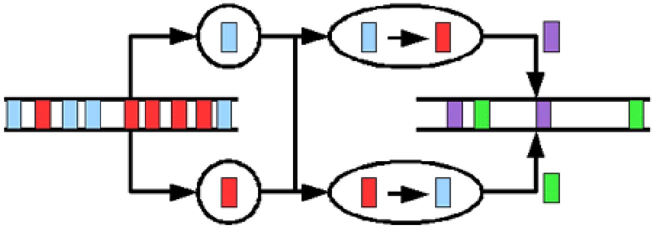
Complex Event Processing

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

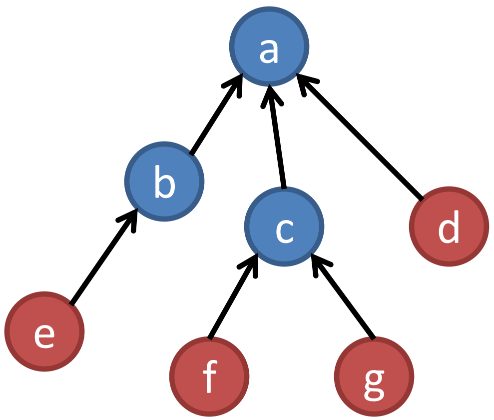
Rule R
define Fire(area: string)
from Smoke(area=$a) **and**
Avg(Temp(area=$a).value
within 5 min. from Smoke) > 45 **and**
not Rain(area=$a) **within** 10 min. **from** Smoke
where area=Smoke.area

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, …

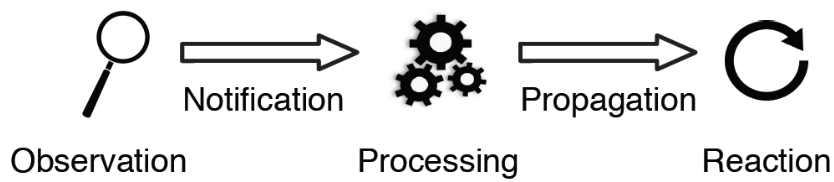


So far…

- Separate communities
  - OOPSLA, ECOOP, ICFP, …
  - DEBS, MIDDLEWARE, …
- Different analysis models

CEP & RLs

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

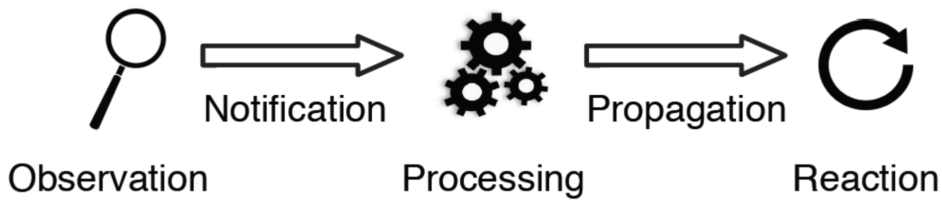


  - Common analysis framework ?
  - Synergies ? Differences ?
  - Integration ?

Ways to react:
Comparing Reactive Languages and Complex Event Processing

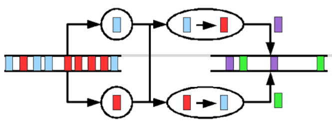
COMPARING CEP AND RLs

Back to Reactive Applications

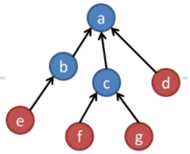


	CEP	RL
OBSERVATION	Generic Events	Value Changes
NOTIFICATION	Explicit – Push	Implicit – Push or Pull
PROCESSING	Rules (primitive E → composite E)	Expressions (signals → signals)
PROPAGATION	Explicit – Multicast – Push	Implicit – Multicast – Push or Pull
REACTION	Generic Procedures – User-Defined	Value Changes

Language Expressiveness



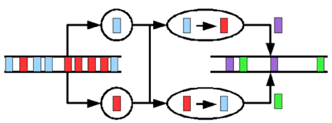
Declarative language

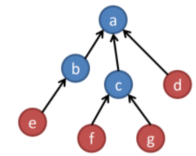


Input: history of event occurrences	Input: signals that hold a value at any point in time and change
Output: time-annotated history of composite events	Output: time changing value

- Time and history are central in CEP
  - However:
    - `Signal.last(n)`
    - `Signal.delay(5)`

Composability

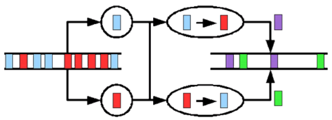


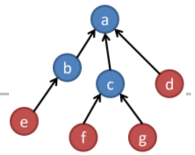


Similar!

Hierarchies of events	Hierarchies of signals
Rules can compose complex events	Event expressions compose signals
Usually no difference between rules on primitive events and rules on composite events	Observable values (Vars) and Signals can appear in signal expressions

Consistency





Users lack control
on the order of evaluation

Primitive events processed in timestamp order

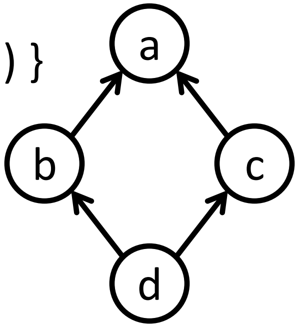
Typically guarantee glitch freedom

Composite events are generated and propagated in timestamp order

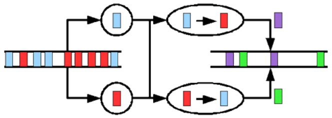
Enforce correct propagation order in the nodes of the dependency graph

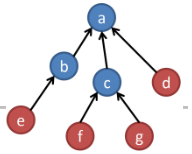
No guarantees when the computation
escapes the controlled propagation

a = Signal{ b() + c() }



Performance





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

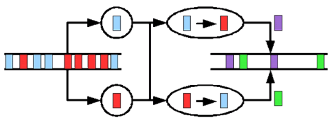
Less attention
Primary focus on language abstractions

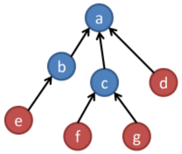
Optimizations:
- Rule rewriting techniques
- Sharing operators among rules
- Algorithms for parallel hardware

Optimizations:
- Lowering
- GADTs for dynamic optimization
- Incrementalization

- Kimberley Burchett, Gregory H. Cooper, and Shriram Krishnamurthi. 2007. Lowering: a static optimization technique for transparent functional reactivity, PEPM '07.
- Henrik Nilsson, Dynamic optimization for functional reactive programming using generalized algebraic data types, ICFP '05.

Distribution





CEP server collects and distributes events to the clients

Load over multiple machines

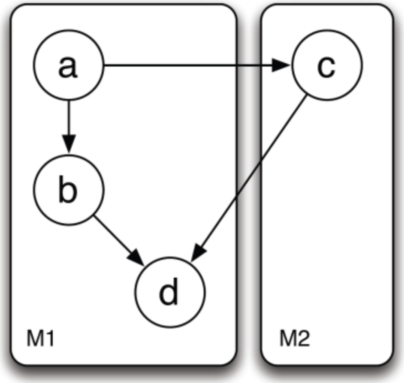
Not much considered yet

AmbientTalk/R

Ongoing: Distributed + glitch free

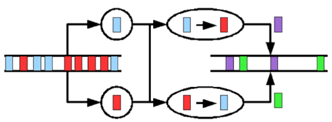
Optimizations:
- Rule rewriting
- Selections close to the sources

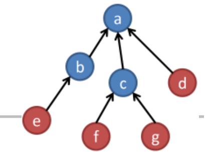
Minimize exchanged messages



• A. Lombide Carreton, S. Mostinckx, T. Cutsem, and W. Meuter. Loosely-coupled distributed reactive programming in mobile ad hoc networks. In J. Vitek, editor, Objects, Models, Components, Patterns, 2010.

Safety





Type casts at the boundaries

- Stream processing:
  table-like approach

- Event-based systems:
  attribute-value pairs

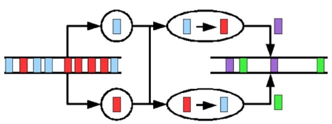
Reactive abstractions
checked by the compiler

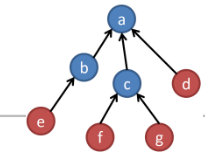
Advanced use of types:
- Execution in bounded space
- Liveness guarantees

Embedding into high-level languages
- EventJava

- Neelakantan R. Krishnaswami, Nick Benton, and Jan Hoffmann. 2012. Higher-order functional reactive programming in bounded space, POPL'12.
- Alan Jeffrey. 2013. Functional reactive programming with liveness guarantees, ICFP '13.

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

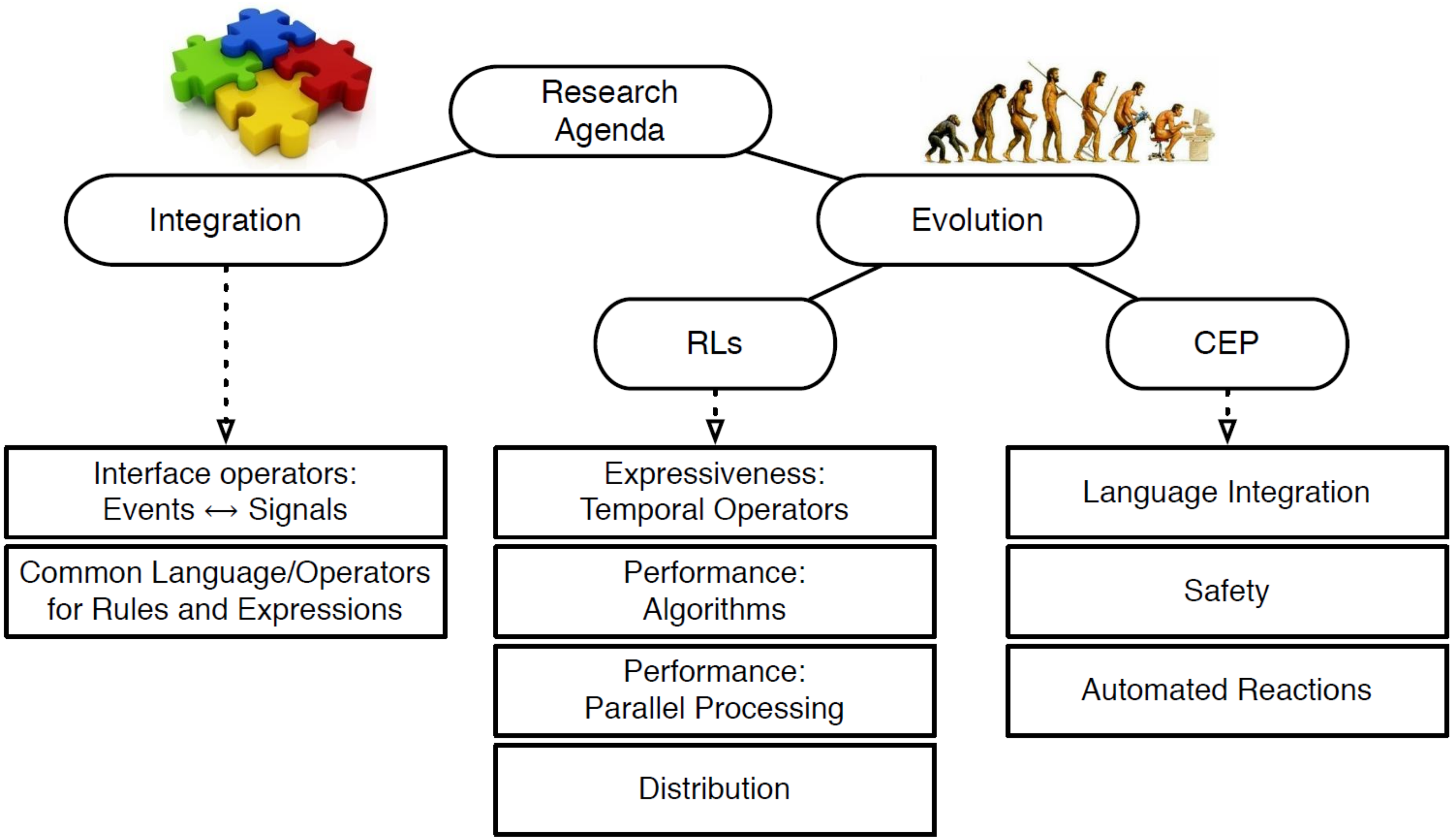
FrTime, REScala, …
Signals as objects fields
Abstract signals
Late binding
…

• 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

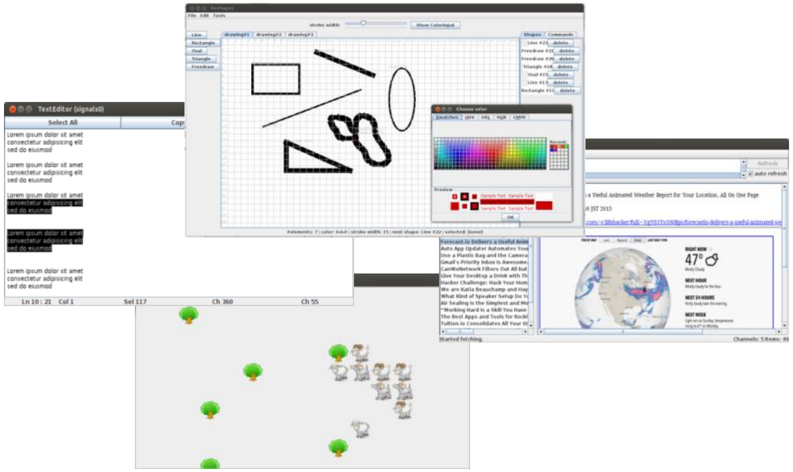
RESEARCH AGENDA

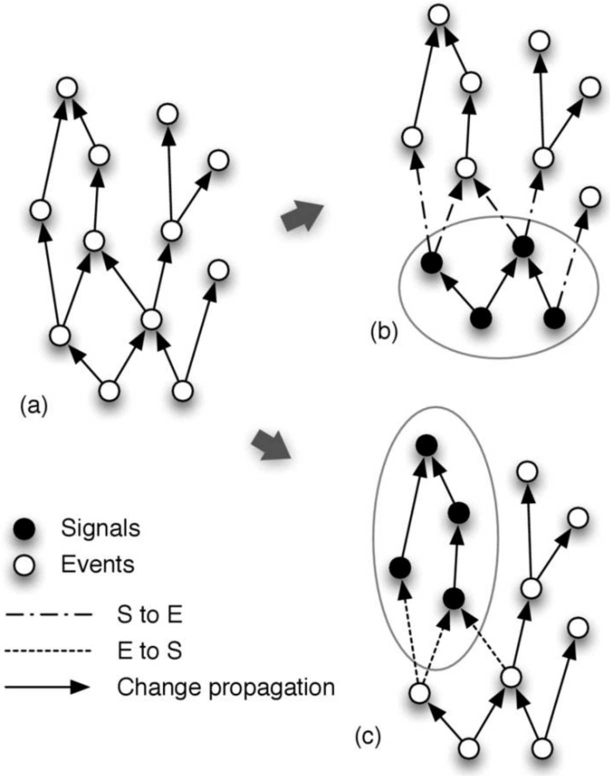
Research Agenda



Integration with OO

- OO + signals ? Reactive objects ?





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

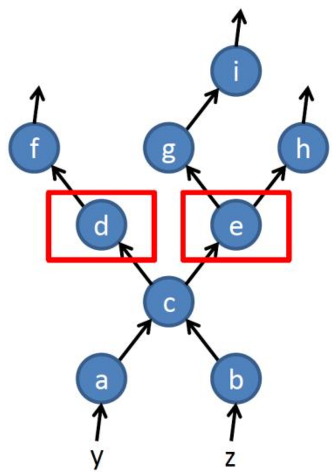
• Guido Salvaneschi, Gerold Hintz and Mira Mezini, REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications, MODULARITY AOSD 2014 (Accepted)

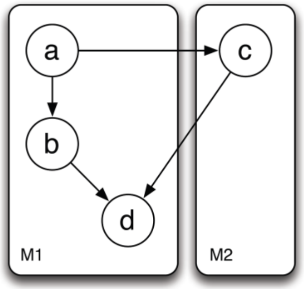
Integration of RLs and CEP

- 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”*

```
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

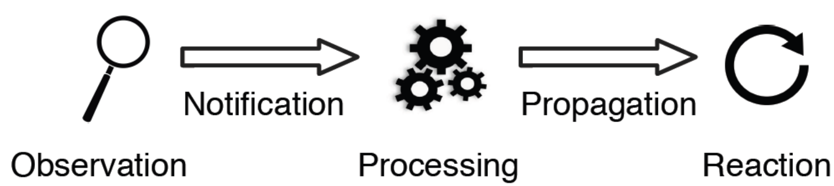
- **CEP** and **RLs** both apply to **reactive applications**

- 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