Reactive Behavior in Object-oriented Applications: An Analysis and a Research Roadmap

Guido Salvaneschi
Mira Mezini
Software Technology Group, University of Darmstadt
Outline

- Design space of OO reactive applications
- Empirical case studies
- Advanced languages for reactive applications
- Research roadmap
Reactive Applications in OO Languages

IDENTIFYING THE DESIGN SPACE
Reactive Applications

• External/internal events trigger a reaction
  – User input
  – Network packet
  – Interrupt
  – Data from sensors

• Classic example:
  – Data change in MVC
Strategies for Reactivity: Graphs

- **Derived graph** (D): edges exceeding MIN weight
- **Add an edge** to the **Basic** graph (B)!
Strategies for Reactivity: Graphs

- On-demand recomputation
- Caching
- Tracking dependencies
- Update incrementalization
- Accumulating changes
Implementing Reactivity in OO Applications

CASE STUDIES
Applications

- **SWT Text Editor**: editor in the SWT library
- **FreeCol Game**: open-source turn-based strategy game
- **Apache JMeter**: performance assessment of servers
- **AccelerometerPlay**: example Android application
Issues of Implementing Reactivity

• Complexity
  – SWT StyledText
    • 70 mutable fields
    • Handlers/listeners = 40 methods

• Hidden Design Intent
  – AccelerometerPlay

• Redundancy

• Scattering and tangling
  – Worst case: field updated in 96 places!
Lesson Learned: Modularization

- Modularization of update code is hard in the OO style
  - Dependencies are not explicit and span over several modules
  - Updates code scattering and tangling
Lesson Learned: Consistency

• Manual update is error-prone

  – **Fail to update** all functionally dependent values.

  – Values are often **updated defensively**
    no clear knowledge of whether it is necessary.
Lesson Learned: Design

- Trade-off: **efficiency** Vs **complexity**.
  - Keep the design simple: accept the cost of on-demand recomputation and potential redundancy.
  - Optimization: highly complicates the application
    - Complex update logic
    - Object fields for intermediate caching
State of the art in reactive applications

ADVANCED LANGUAGES
The \textit{(good? old) Observer Pattern}

\textit{Decoupling the code that changes a value from the code that updates the values depending on it}

- Long story of criticism...
  - Inversion of \textit{natural} dependency order
  - Boilerplate code
  - Reactions do not compose
  - Scattering and tangling of triggering code
  - ...
Event-based Languages

• Event-based languages are *better*!
  – Language-level support for events
  – C#, Ptolemy, EScala, ...

• Features:
  – Integration with OO and imperative style:
    • Fine-grained updates
    • Object identity
  – More composable
  – Less boilerplate code

```scala
abstract class Figure {
  def moveBy(dx: Int, dy: Int) { position.move(dx, dy) }
  def resize(s: Size) { size = s }
  def setColor(col: Color) { color = col }
  def getBounds(): Rectangle
}
```
Event-based Languages

• Dependencies still encoded manually
  – Handler registration

• Updates must be implemented explicitly
  – In the handlers

• Dependents notifications is still error prone:
  – Too rarely / too often

• No support for non-functional choices
  – E.g. manual caching
  – Accumulation of changes?

```scala
class Connector(val start: Figure, val end: Figure) {
  start.changed += updateStart
  end.changed += updateEnd

  def updateStart() { ... }
  def updateEnd() { ... }
}
...
Reactive Languages

- Functional-reactive programming (FRP) -- Haskell
  - Time-changing values as dedicated language abstractions.

```javascript
var nowB = timerB(1000);
var startTm = nowB.valueNow();
var clickTmsB = $E("reset", "click").snapshotE(nowB)
                 .startsWith(startTm);
var elapsedB = nowB - clickTmsB;

insertValueB(elapsedB, "curTime", "innerHTML");
<body onload="loader()"
     <input id="reset" type="button" value="Reset"/>
     <div id="curTime"> </div>
</body>
```

[Flapjax: a programming language for Ajax applications. Meyerovich et al. OOPSLA’09]
Reactive Languages

var elapsedB = nowB - clickTmsB;

• Several issues in the case studies are solved!
  – Dependent values automatically consistent
    • Simpler applications
    • No update errors
  – The update code modularized with the entity
    • Local reasoning
    • No scattering and tangling
  – Reactive behaviors are composable
    • In contrast to callbacks, which return void
    • Design intentions are explicit
    • Declarative style
(Functional) Reactive Languages

var elapsedB = nowB - clickTmsB;

- Do not fit well with mutable objects.
  - Reactive languages recompute the dependents every time
    - E.g. require immutable data structures
  - Changes to the state of the object are not even detected

list2 = list1.filter(x>10)
Reactive Collections

- Efficient incremental updates, LiveLinq, Glazed Lists
  - “View maintenance” in the database community

```java
EventList cars = new BasicEventList();
EventList sorted = new SortedList(cars);
TextFilterList filtered = new TextFilterList(sorted);
filtered.getFilterEdit().setText("Hybrid");
cars.add("Insight");
cars.add("EV1");
cars.add("Prius Hybrid");
cars.add("Jetta TDI");
cars.add("Escape Hybrid");
cars.add("Escalade");

// should print "[Escape Hybrid, Prius Hybrid]"
System.out.println(filtered);
```
Reactive Collections

• Advantages similar to reactive languages
  – Automatic updates!

• Efficient update of various operators
  – Incremental changes /caching

• Approach restricted to a specific domain / operators
Summary

• Event-based languages
  😊 Fit into the OO design, mutable objects / fine-grained changes
  😞 Not declarative.
  😞 Drawbacks of Observer

• Reactive languages
  😊 Overcome the limitations of Observer
  😊 Declarative style
  😞 No fine grained changes / mutable objects, no integration with OO

• Reactive collections
  😊 Efficient, declarative
  😞 Single domain
RESEARCH ROADMAP
Roadmap

Reactive languages

- Integration with Event-based Programming
- Integration with Object-oriented Design
- Efficient Reactive Expressions
- Propagation Model
Roadmap

Reactive languages

• Integration with Event-based Programming
• Integration with Object-oriented Design
• Efficient Reactive Expressions
• Propagation Model
Integration of Reactive Programming and Event-based Programming

• Events and reactive abstractions are complementary

• Interface
  • [Signal → Event]    Signal.changed
  • [Event → Signal]    Event.latest
  • [(S, E) → Signal]   snapshot

• Methodology
  – When to use behaviors and when events
  – Case studies and guidelines
  – Patterns ?
Roadmap

Reactive languages

• Integration with Event-based Programming
• Integration with Object-oriented Design
• Efficient Reactive Expressions
• Propagation Model
Integration of Reactive Programming and Object-oriented Design

• Methodological/design issues
  – Encapsulation
  – Behaviors should be part of the interface?
    • Private behaviors?
  – Reassign behavior expressions?
  – Inheritance of behaviors?
  – Subtype polymorphism?

[Crossing state lines: Adapting object-oriented frameworks to functional reactive languages, Ignatoff et al, FLOPS’06]
Roadmap

Reactive languages
• Integration with Event-based Programming
• Integration with Object-oriented Design
• Efficient Reactive Expressions
• Propagation Model
Efficient Reactive Expressions: Challenge!

- Reconciling the **openness** of reactive languages with the **efficiency** of reactive collections
Efficient Reactive Expressions (1)

• Predefined operators cover the most common scenarios e.g. collections and relational operations.

• Interoperability with reactive abstractions

```
s = list1.filter(x>K).size
```

behaviors
Efficient Reactive Expressions (2)

```python
list2 = list1.filter(x>10)
```

- Domain knowledge enables incremental updates.
  - Open refinements of reactive computations!
  - Can be easy as overriding an existing method?

- Separate **creation** from **maintenance**
- **Default**: complete recomputation
- **Late binding** of *the best* refinement

- Generality of mechanism Vs domain-specific incrementalization
Roadmap

Reactive languages

• Integration with Event-based Programming
• Integration with Object-oriented Design
• Efficient Reactive Expressions
• Propagation Model
Propagation Model: Intro

- Graph model
  - Constraints among dependent values
  - Change propagation

\[
\begin{align*}
f &= 1 \\
g &= 1 \\
c &= f + g \quad \text{// } c = 2 \\
f &= 2 \quad \text{// } c = 3
\end{align*}
\]
c := f + g

b := e * 100

d = true

a := { if(d)  c  
      else  b  }
Propagation Model: Strategies

• Current languages: single point in the design space
  – Caching with invalidation?
  – Proactive recomputation?
  – Caching of intermediate values?
  – Accumulate changes?
  – Adaptive mechanism?
Propagation Model: Collapsing

- Optimizations
  - Alternative graphs can be observationally equivalent
  - Performance can guide the choice.
    - Caching Vs moving to the stack
Propagation Model: Collapsing

(time-consuming-op
  (infrequently-changing-op frequent-emitter))

• Choose carefully!
  – Code analysis / dynamic techniques?
Propagation Model: VMs

• Dedicated VM support?

• Graph:
  – Events
  – Time-changing values

• Previous work: dedicated VM support for AOP
  [Efficient control flow quantification, Bockisch et al, OOPSLA’06]
Propagation Model

• **Current languages:** expressive power
  – E.g. dynamic signal creation

• **Dataflow languages:** performance
  – Lustre, Esterel: memory and time-bound execution

• Black-or-white choice

• Leave the programmer the choice!
  – Whole range available
  – Explicit performance implications
Ongoing!

- ERC grant *PACE*, Prof. Mira Mezini
- *Programming Abstractions for Applications in Cloud Environments*
  - [http://www.stg.tu-darmstadt.de/index.en.jsp](http://www.stg.tu-darmstadt.de/index.en.jsp)

Interested? Contact us!
Ongoing: REScala

• **REScala**: R(reactive)EScala

  - Development methodology for reactive applications
  - Systematic evaluation by case studies
  - Tech report is out!
    - [http://www.stg.tu-darmstadt.de/research/escala/index.en.jsp](http://www.stg.tu-darmstadt.de/research/escala/index.en.jsp)
Thank you!

QUESTIONS?
Take Away

• Reactive applications are an important research area.
• Current solutions go only half the way:
  – Known to be flawed (Observer)
  – Promising (reactive programming)
  – Specific for a single domain (reactive collections)

• Merge of those techniques
• Integration with OO programming