Scaling-Up Behavioral Programming: Steps from Basic Principles to Application Architectures

David Harel and Guy Katz

Weizmann Institute of Science
Overview

- The Behavioral Programming (BP) paradigm
  - Scenario-based programming
- Previous work: BP is incremental & natural
- But does it scale up?

- Attempt to apply BP to a large case-study (a webserver)
- Do BP’s desirable traits carry over to large systems?
  - Conclusion: yes, but…
  - With some extensions to BP
Agenda

- Introduction to Behavioral Programming
- Our proposed extensions
- Case-study: a web server
Agenda

- Introduction to Behavioral Programming
- Our proposed extensions
- Case-study: a web server
Behavioral Programming (BP)

- A scenario-based paradigm for programming reactive systems
- Program by specifying scenarios
  - Desirable scenarios
  - Undesirable scenarios
- All scenarios are consulted at runtime
  - Producing cohesive system behavior

Behavioral Programming (cnt’d)

- A program has events and threads
- At synchronization points threads pause and declare
  1. Requested events
  2. Waited-for events
  3. Blocked events
- Event selection at synchronization points:
  1. Trigger an event requested by some thread and blocked by none
  2. Inform threads that requested/wait-for the event
The Execution Cycle
The Execution Cycle
The Execution Cycle

- **Request**
- **Behavior Threads**
- **Wait**
- **Block**
AddHotFiveTimes() {
    for i=1 to 5 {
        bSync(request=addHot, wait-for=∅, block=∅);
    }
}

AddColdFiveTimes() {
    for i=1 to 5 {
        bSync(request=addCold, wait-for=∅, block=∅);
    }
}

Interleave() {
    forever {
        bSync(request=∅, wait-for=addHot, block=addCold);
        bSync(request=∅, wait-for=addCold, block=addHot);
    }
}
Motivation for BP

- Incremental, non-intrusive development
  - New requirement? Add a thread
  - Program repair
- Threads aligned with the specification
- Natural / easy to learn
- Fosters abstract programming
BP and the Actor Model

- **Similarities:**
  - Actors / Behavior Threads: narrow view of the system
  - Event passing between threads

- **Differences:**
  - Synchronization is global
  - Undesired behaviors/the **blocking** idiom

- We regard Actors and BP as complementary
Agenda

- Introduction to Behavioral Programming
- Our proposed extensions
- Case-study: a web server
Time in BP

- Traditional BP assumed zero-time actions
  - Threads re-synchronize immediately
- Threads with multiple time scales?
- Partial solutions exist (Harel et al, AGERE! 2011)
- But, no way to reason about time
Example: Railway Crossing

- Upon `trainComing`, lower the gate
- The gate must remain down for 30 seconds

Thread `LowerGate`

```latex
while ( true )
  bSync(request=\emptyset, wait-for=\textit{trainComing}, block=\emptyset)
  bSync(request=\textit{lowerGate}, wait-for=\emptyset, block=\emptyset)
```

Thread `PreventRaise`

```latex
while ( true )
  bSync(request=\emptyset, wait-for=\textit{lowerGate}, block=\emptyset)
  bSync(request=\emptyset, wait-for=\emptyset, block=\textit{raiseGate})
```
Extension: A Timeout Idiom

- Extend synchronization calls with a timeout parameter
  \[
  \text{bSync}( \text{request}, \text{wait-for}, \text{block}, \text{timeout} )
  \]
- Threads synchronize, and an enabled event is triggered
- No enabled events? Wait for nearest timeout value
- Wake up the thread that timed-out
  - That thread may change the \text{requested}/\text{blocked} events

```
Thread PreventRaise
  while ( true )
    \text{bSync}(\emptyset, \text{LowerGate}, \emptyset, \infty)
    \text{bSync}(\emptyset, \emptyset, \text{raiseGate}, 30)
```
Strategies

- Often, multiple events requested and not blocked
  - Which is triggered?

- Traditional solutions:
  - Arbitrary
  - Event / thread priorities
  - Round robin

- Our extension: selection strategy a part of the program
  - Tailor event selection to programmer’s needs
Dynamic Thread Creation

- Previously, threads exist throughout the run
- Difficult to handle requirements that change throughout the run
  - E.g., user action creates a thread
- Our extension: dynamic thread creation
  - Threads spawn other threads, in response to events
Parameterized Events

- Previous programs dealt finitely many events
- Explicitly name all possible events…
- Our extension: allow events with parameters
Agenda

- Introduction to Behavioral Programming
- Our proposed extensions
- Case-study: a web server
The Project

- Large scope: a TCP stack and a HTTP stack
  - Together, they form a webservice
- Various programming tasks: timeouts, string manipulation, file access, checksums, multiple inputs, mandatory and forbidden behavior, etc.

- Goal: find out whether this is feasible using BP
  - Answer: yes, with the aforementioned extensions
- Sub-goals:
  - Program incrementally
  - Align threads with the specifications
The Need for the Extensions

- **Timeouts:**
  - Every TCP segment needs to be acknowledged
  - Otherwise, resend it

```
Thread ResendSegment
  do {
    bSync(sendSegment, ∅, ∅, ∞)
    bSync(∅, ack, ∅, 2)
  } while (timeoutInLastSync() )
```
The Need for the Extensions

- Spawn Threads: new thread per connection
- Strategies: answer urgent segments first
- Parameterized events: carry a segment’s payload
Conclusions & Future Work

- We’ve developed a large behavioral application
- In the process, extended BP with:
  - Timeouts
  - Dynamic Thread Creation
  - Strategies
  - Parameterized events
- In the future: extend our case study
  - May reveal additional idioms worth adding to BP
- Extend program analysis tools (model-checking, repair, etc) to the new variant of BP
Thank You!

Questions