Support for Feedback and Change in Self-adaptive Systems

Dharini Balasubramaniam, Ron Morrison, Kath Mickan, Graham Kirby
University of St Andrews, UK

Brian Warboys, Ian Robertson, Bob Snowdon, R Mark Greenwood, Wykeen Seet
University of Manchester, UK

EPSRC GR/M88938 - Compliant Systems Architecture
IST-2001-32360 - ArchWare: Architecting Evolvable Software
The Known Universe
Evolving Systems

Decompose

Evolve

Compose

server

client

State
For WOSS

• What aspects of self-management are being addressed?
  - Language mechanisms for feedback and change
• What aspect are not being dealt with?
  - Architectural specification & change policy
• What domains, properties, or applications are being targeted?
  - Evolution in continuous running systems
• What are the new technical ideas?
  - Structuring for self-adaptation
  - Probe insertion in a type-safe environment
  - What would we do if we knew what to do?
Language Support

- **Computationally complete sugared π-calculus with behaviours (components) and connections (interactions)**
  - Static and dynamic (assignment & higher-order evaluation)
- **Composition and decomposition operators (for dynamic evolution)**
  - Partial decomposition into constituent parts at their reduction limit
- **Structural reflection (for dynamic evolution)**
  - Reification (into hyper-code), change, recompilation and rebinding
- **Hyper-code (representation of active components and connectors)**
  - Execution graph provides single view of software throughout its life cycle
  - Hyperlinks preserve state and shared data through evolution
  - Visual and programmable interface
value server_abs = abstraction()
{ replicate{
    via in_int receive i;
    via out_int send 2*i
}
};
value server1 = server_abs();
value client1 = client_abs(25);
value client2 = client_abs(50);
compose{ server1 and client1 and client2 }
Elements of Feedback

• Probes
  - Observe and quantify significant events in execution (What are significant events?)
  - Send feedback via event distribution network
  - Modelled as functions using published interface of observable values (locations)

• Gauges
  - Filter & store feedback using knowledge of constraints

• Feedback source
  - Any component which is an origin of feedback

• Feedback sink
  - Any component which is a target for feedback

• Event distribution network
  - Used to send feedback from source to to sink
  - Modelled as connections
Feedback Models

- **Feedback models**
  - Define how elements are structured to form feedback mechanism

- **Criteria**
  - Separation of concerns (functionality and evolution)
  - Uniform/different feedback mechanism for different types of sources
  - All/specialised feedback from sources

- **Examples**
  - *Gauge in sink*
    - Sink is notified of all significant events by source
  - *Gauge in source*
    - Sink is only notified of events of interest
Feedback - An Example

Observable Interface

Probe connection

Feedback connection

Probe

Feedback sink

Feedback source

Location for observation

Feedback sink

Feedback source

Feedback connection

Feedback connection

Feedback sink

Feedback source

Feedback connection

Feedback connection

Feedback sink

Feedback source

Feedback

WOSS 2004

9 - Newport Beach, CA
Development Methodology

- P2E - process for process evolution
- Producers and evolvers

Diagram:

- Component
- Evolver
- Producer
- Raw
- Environment
- Widgets

Connections:
- Probe, feedback and change connections
Composition of P2E Elements
**P2E for Self-adaptation**

- For each level of abstraction
  - Source as producer
  - Sink as evolver
  - Probe, feedback and change connections between source and sink (event distribution network)
  - Source publishes observable interfaces
    » Values for probes
    » Components for change
  - Feedback from source to sink
  - Change guidance from sink to source
Example

Feedback sink (evolver) → Feedback source (producer) → Change component → Feedback sink (evolver)
Example - contd.
What are the new technical ideas?

• Done
  - The technologies
  - Structuring for self-adaptation
  - Probe insertion in a type-safe environment

• Outstanding issues
  - Conflict in parallel evolutions (long-running transactions, sagas …)
  - Evolution patterns (to partially automate the process)
  - How do we transfer the technology to well-used platforms
Example Code - Types

recursive type feedback_interface_type is view [  
observation_name : string,  
observeration_val : any,  
next : location[feedback_interface_type] ]

recursive type component_interface_type is view [  
component_name : string,  
component_val : location[behaviour],  
next : location[component_interface_type] ]
Example Code - System

```plaintext
value feedback_conn = connection(integer )
value change_conn = connection(abstraction[] )

value system_component = compose{
    producer as a_source()
    and evolver as a_sink()
}
```
Example Code - Source (Producer)

value a_source = abstraction()
{
  value source_observable_interface = generate_observable_interface("a_source")
  value source_component_interface = generate_component_interface("a_source")
  value count_1 = location(0) ! updated by component_1
  publish_observation(source_observable_interface, "count_1", any(count_1))
  value count_1_probe = abstraction() ! used by component_1
  { via feedback_conn send count_1 }
  value composite = compose {
    comp1 as component_1
    and comp2 as component_2
    and comp3 as component_3
  }
  value composite_loc = location(composite)
  publish_observation(source_component_interface, "composite_loc", composite_loc)

  replicate{
    via change_conn receive change_component
    change_component() }
}
Example Code - Sink (Evolver)

value a_sink = abstraction()
{
  replicate{
    via feedback_conn receive source_count
    if fails_constraint( source_count ) do
      value make_change = abstraction()
      {
        value composite = 'composite_loc
        value components = decompose composite
        value new_component_1 = evolve_component( components::1)
        composite_loc := compose {
          comp1_1 as new_component_1
          and comp_2 as components::2
          and comp_3 as components::3
        }
      }
    via change_conn send make_change
  }
}