Transparent Componentisation

Multi-Paradigm Programming for Evolving Distributed Systems

Shen Lin¹, François Taïani¹,², Gordon Blair¹, Marin Bertier², Anne-Marie Kermarrec²
Motivation

Today’s Distributed Systems

- **Heterogeneous**
  - resources, mobility, context

- **Dynamic**
  - churns, failures

- **Large and Complex**
  - SoS, P2P, cloud

Need for solutions that are *flexible, evolvable, expressive*
The rise of components

- Component successfully applied to distributed systems
  - industry: EJB, CCM, OSGi
  - research: Fractal, OpenCOM
  - middleware Frameworks: GridKit, Rapidware, Ensemble, Cactus, Open Overlays

- Clear structure, explicit dependencies

- Benefits
  - ☀ promote reuse
  - ☀ easily composable and configurable (SPL..)
  - ☀ lend themselves to runtime reconfiguration
The problem with components

Components focus on **structure**, not **behaviour**

- **Drawbacks**
  - 😞 **low intelligibility** (where is the intent?)
  - 😞 **conceptual mismatch** for developers focusing on behaviour
  - 😞 **high learning** curve for unfamiliar frameworks
High-level distributed languages

- **Spec. lang. and DSL:** High-level per node description
  - Lotos, Estelle, PLAN-P, Mace …

- **Marco-programming:** system as one entity
  - E.g. Kairos, Regiment, TinyDB, MIT-Proto
  - centralised shared-memory parallel abstraction
  - main program compiled into code for each node

- **Benefits**
  - 😊 high level of *abstraction* (in particular for macro-prog)
  - 😊 *intelligible*
  - 😊 good conceptual *match* for developers looking at behaviour
Behaviour rather than structure

Can we build a hybrid approach that combines the strengths of both?

- Drawbacks

😊 we loose the benefits of components (reuse, …)

F. Taiani
Outline

concept: transparent componentisation

application to gossip: WhispersKit

evaluation: simplicity and performance
Outline

- **concept**: transparent componentisation
- application to gossip: WhispersKit
- **evaluation**: simplicity and performance
structure + behaviour = ?

- **tangling** behaviour & structure
- ‘breaks’ **encapsulation**
- tension **granularity** vs. scattering

- **complex** composition
- tension **structural** needs vs. **programmatic** ones
structure + behaviour = ?
Transparent Componentisation

- **Separation of concern** between behaviour / structure
- **Developers** can focus on **high level logic**
- **Systems** takes care of **modularity**, reuse, and evolution

- ☺ simple
- ☺ concise
- ☺ high-level
- ☺ modular
- ☺ reusable
- ☺ (re)configurable

F. Taiani
Outline

- concept: transparent componentisation
- application to gossip: WhispersKit
- evaluation: simplicity and performance
Application to Gossip Protocols

- Gossip (aka epidemic) protocols
  - leverage **rumour-like** propagation of information
  - large applicability: aggregation, broadcast, clustering
  - often **composed** to realised higher-level services

- Conceptually simple
  - typically symmetric behaviour
  - key notions of **state**, information **flows**, and **decisions**

- But implementation can be time consuming
  - multithreading, distributed coordination, network intricacies, co-existence
The WhispersKit Architecture

GossipKit Framework

Metamodel

Abstract Model

Runtime

Configuration Description

Component Architectural Abstraction

Reconfiguration Management

Event-driven Architecture

OpenCom Component Framework

Gossip Developer

Whispers

F. Taiani
GossipKit

- A component framework for epidemic protocols
  - based on analysis of 30 gossip protocols
  - event-based
  - XML-based configuration for component composition
  - targets abstraction, modularity, reuse, evolvability

F. Taiani
GossipKit Examples

**RPS**

**Anti-Entropy**

**T-Man**

**SCAMP**

**Wireless broadcast**
Whispers

- macro-programming language for gossip protocols
  - system as one entity

- primitives

```java
protocol { .. }  // protocol block
every (time) { .. }  // periodic behaviours
wait (Event e type T) { .. }  // reactive behaviours
foreach(n in nodeSet)  // distribution
  synchronised { .. }  // pairwise data exchange
State state = new State[fields][size] ;  // state decl.
state.field ;  // get a column of data
state.add([fields])  // add
state.remove(row_ID)  // remove
i.RandomStateCompress(...)  // library call
```
Whispers Example: RPS

RPS {
    State sample = new State[Node:PeerID][Size=5];
    Node n, i;
    every (5000) { // do the following every 5000 ms
        foreach (n in AllNodes) { // for each node n
            i=n.RandomPeerSelection(n.sample)[Size=1];
            n.sample.add([n]);
            i.RandomStateCompress(i.sample,n.sample)[Size=5];
            n.RandomStateCompress(i.sample,n.sample)[Size=5];
        } // end of foreach
    } // end of every
} // end of RPS protocol block
Deployment Process

1. Programs that describe system behaviours

2. Componentisation Mechanism

- C2
- F1
- Net1
- Component Configuration

Node n's Runtime

3(a) Initialisation
- C2
- F1
- Net1
- S2

3(b) Apply reconfiguration to an existing system
- C2
- F1
- Net1
- S1
- S2

Network

F. Taiani
```
State sample = new State[Node:PeerID][Size=5];
Node n, i;
every (5000) { // do the following every 5000 ms
  foreach (n in AllNodes) { // for each node n
    i = n.RandomPeerSelection[n.sample][Size=1];
    n.RandomStateCompress[i.i.n.RandomStateCompress][i.i.n.RandomStateCompress];
  }
}
// end of foreach
// end of every
// end of RPS protocol block
```
Distributed Reconfiguration

- Developers describe new behavior in Whispers
- Platform uses component representation
  - to compute minimal set of change
  - to propagate and enact reconfiguration
Distributed Reconfiguration

- Example: RPS $\rightarrow$ T-Man(Ring) $\rightarrow$ T-Man(Grid)

- coarse grained
- fine grained

Figure 5.6: Initial random graph maintained by RPS
Figure 5.7: 5th rounds since 1st reconfiguration
Figure 5.8: Ring constructed at the 11th round

Figure 5.9: Topology at the 20th round
Figure 5.10: Grid constructed at the 23rd round

F. Taiani
Outline

- **concept:** transparent componentisation
- application to gossip: **WhispersKit**
- **evaluation:** simplicity and performance
## Evaluation: Simplicity (1)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>WHISPERS</th>
<th>Java</th>
<th>GOSSIPKIT</th>
<th>XML Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gossip1</td>
<td>14</td>
<td>277</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>Gossip2</td>
<td>14</td>
<td>279</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>Anti Entropy</td>
<td>16</td>
<td>544</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Averaging</td>
<td>14</td>
<td>466</td>
<td></td>
<td>85</td>
</tr>
<tr>
<td>Ordered Slicing</td>
<td>14</td>
<td>471</td>
<td></td>
<td>85</td>
</tr>
<tr>
<td>RPS</td>
<td>12</td>
<td>439</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>SCAMP</td>
<td>19</td>
<td>463</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>T-Man</td>
<td>20</td>
<td>491</td>
<td></td>
<td>93</td>
</tr>
<tr>
<td>Average</td>
<td>15.4</td>
<td>424</td>
<td></td>
<td>76.3</td>
</tr>
</tbody>
</table>
Evaluation: Simplicity (2)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>WHISPERS</th>
<th>Java</th>
<th>GOSSIPKIT configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cyclomatic Comp.</td>
<td>Component</td>
<td>Parameter</td>
</tr>
<tr>
<td>Gossip1</td>
<td>2</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Gossip2</td>
<td>2</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Anti Entropy</td>
<td>3</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Averaging</td>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Ordered Slicing</td>
<td>3</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>RPS</td>
<td>2</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>SCAMP</td>
<td>3</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>T-Man</td>
<td>3</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Average</td>
<td>2.6</td>
<td>11.5</td>
<td>7.3</td>
</tr>
</tbody>
</table>
Performance Overhead

![Bar chart showing performance overhead for different applications and tools. The chart compares GossipKit and JVM for tasks such as Gossip1, Gossip2, Averaging, Anti Entropy, SCAMP, Ordered Slicing, RPS, and T-MAN. The y-axis represents local process time in microseconds (µs), ranging from 0 to 2000. The x-axis lists the different tasks and applications.]
Conclusion

- **Transparent componentisation**
  - separates the behavioural concerns from structural ones
  - developers can entirely focus on their code’s logic
  - while harnessing the benefits of a modular approach

- **Whispers/GossipKit**
  - combines component framework and high-level language
  - Simple and high-level expressions
  - Macro-level description
  - Automatic mapping from programs onto components

- **Available on line:** [http://ftaiani.ouvaton.org/GossipKit/](http://ftaiani.ouvaton.org/GossipKit/)
Future Work

- **Extend** Whispers/GossipKit beyond Gossip
- **Exploit** Transparent Componentisation for
  - reasoning about interferences and synergies
  - self-adaptation
- **Link** to Model-driven Engineering
- More generally explore link between structure / behaviour / and intelligibility in dist. systems
Thank you

Questions?
Composite Example: Averaging

Averaging {
RPS rps = new RPS();
State state = new State[float:Temp][Size=1];
Node n, i;
every (5000) {
    foreach (n in AllNodes) {
        i = n.rps.RandomPeerSelection(n.rps.sample)[Size=1];
        synchronised {
            i.Average(i.state.Temp, n.state.Temp)[];
            n.Average(i.state.Temp, n.state.Temp)[];
        } // end of synchronised
    } // end of every
} // end of foreach
} // end of Averaging protocol block