Programming Large-Scale Distributed Systems
Some Mechanisms, Abstractions, and Tools

François Taïani
Soutenance d’HDR
17 Novembre 2011
“Middleware Engineering”

- Distributed Programming
  - fault-tolerant MW
  - overlays
  - self-adapting WSN

- Experimental Software Engineering
  - rev. engineering
  - gossip protocols
  - middleware analysis
  - reliability of AOP

Contributors:
- Barry Porter
- Shen Lin
- Nathan Weston
- Rachel Burrows
“Middleware Engineering”

- Distributed Programming
  - fault-tolerant MW
  - overlays
  - self-adapting WSN
  - gossip protocols
  - rev. engineering

- Experimental Software Engineering
  - middleware analysis
  - reliability of AOP

Contributors:
- Barry Porter
- Shen Lin
- Nathan Weston
- Rachel Burrows
A Distributed System Today ...

- *External services*
  - facebook
  - twitter
  - bit.ly
- *standards*
  - OAuth
  - JSON
- *middleware*
  - foursquare™
- *external developers*
  - Geosocial app, est. 2009
- *10M Users*
- *Middleware*
  - mongoDB
  - Flume
- *Amazon Web Services™*
Challenges

- Dynamicity & Scale
  - Google ~ 1M (?) servers
  - foursquare (geosocial network): 10M users within 2 yrs
  - Facebook: 800M active users
one RPC request,
• 2065 individual invocations
• > 50 C-functions
• > 140 C++ classes
Challenges

■ Dynamicity & Scale
  ➔ Google ~ 1M (?) servers
  ➔ foursquare (geosocial network): 10M users within 2 yrs
  ➔ Facebook: 800M active users

■ Complexity & Heterogeneity
  ➔ 🌹 functionalities
  ➔ 🌹 dependencies
  ➔ 🌹 providers
  ➔ 🌹 devices
  ➔ 🌹 inconsistencies
  ➔ 🌹 code size
How to design, program, and analyse these types of system?
Our take

Reusable programming abstractions for large-scale distributed systems

- Which abstractions?
- Supported by which tools?
Outline

- Intro (just done)
- WhisperKit: Programming Gossip-based Systems
- ProfVis: Anomaly Diagnosis in Grid Middleware
- Conclusion and Outlook
Outline

■ Intro (just done)

■ **WhisperKit:**
  Programming Gossip-based Systems

■ **ProfVis:**
  Anomaly Diagnosis in Grid Middleware

■ Conclusion and Outlook

*Joint work with: S. Lin, G. Blair, A.-M. Kermarrec, M. Bertier*
Motivation: Gossip Protocols

- Highly scalable, efficient, and robust

- Applied to wide range of services
The Problem with Gossip

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

- But implementation can be **time consuming**
Which **reusable abstractions** to facilitate Gossip programming?
Our Take: Components

- Component successfully applied to distributed systems
  - Rapidware, GridKit, Cactus, FraSCAti

- Clear structure, explicit dependencies

- Benefits
  - Smiley reusability
  - Smiley composability and configurability
  - Smiley runtime adaptation
GossipKit

- Analysis of 30 Gossip protocols

- Result: A component framework for gossip protocols
  ➔ targets abstraction, reuse
Example: Random Peer Sampling

- **Goal**: periodically returns a random set of other peers
GossipKit Examples

- **RPS**  
  [ToCS 07]

- **T-Man**  
  [Computer Networks 09]

- **Anti-Entropy**  
  [PODC 87]

- **Wireless broadcast**  
  [ToN 06]

- **SCAMP**  
  [ToC 03]
The problem with components

Recipe

Cook

Cupboard

Bowl

Form

Oven

How can best to combine **behaviour** and **structure**?

Components tend to focus on **structure**, not **behaviour**.
High-level dist. languages

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

- **Macro-programming:** system as one entity
  - E.g. Kairos, Regiment, TinyDB, MIT-Proto

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

- Drawbacks
  😞 we lose the benefits of components (reuse, adaptation, …)

```java
add(yohourt, 1)
add(milk, 2)
add(flour, 3)
add(butter, 1)
add(eggs, 2)
add(soda)
bowl.mix()
bowl.pour(form)
form.putIn(oven)
bake()
```
Can we build a hybrid approach that combines the strengths of components & high-level languages?
**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
WhisperKit = Whisper + GossipKit

- **Whispers**: inspired from macro-programming (Kairos, …)
- **WhisperKit**: compiler/deployment chain (JavaCC)
  - Built-in support for distributed reconfiguration
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
Deployment Process

1. Programs that describe system behaviours

2. Componentisation Mechanism

3(a) Initialisation
- Node n's Runtime

3(b) Apply reconfiguration to an existing system

Network

send/recv

Node 1
Runtime Execution

Node 2
Runtime Execution

Node n
Distributed Reconfiguration

- Developers describes new behaviour in Whispers
- Platform uses component representation
  → to compute minimal set of changes
  → to propagate and enact reconfiguration

![Diagram showing component mapping and reconfiguration process](image-url)
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
### Evaluation: Simplicity (1)

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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>WHISPERS</th>
<th>Java</th>
<th>GOSSIPKIT</th>
<th>configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cyclomatic Comp.</td>
<td>Component</td>
<td>Parameter</td>
<td>Connection</td>
</tr>
<tr>
<td>Gossip1</td>
<td>2</td>
<td>11</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Gossip2</td>
<td>2</td>
<td>11</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Anti Entropy</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Averaging</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Ordered Slicing</td>
<td>3</td>
<td>11</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>RPS</td>
<td>2</td>
<td>12</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>SCAMP</td>
<td>3</td>
<td>20</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>T-Man</td>
<td>3</td>
<td>11</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>Average</td>
<td>2.6</td>
<td>11.5</td>
<td>7.3</td>
<td>11.4</td>
</tr>
</tbody>
</table>

Cyclomatic Complexity [McCabe76]:
≈ Number of decision points within a program
Summary

- **GossipKit**
  - First component-based framework for gossip protocols
  - Simple and general

- **Whispers/WhisperKit** (CBSE + DSL)
  - Separates behavioural from structural concerns
  - Highly concise programs, that retain component benefits

- **Impact** of this line of research
  - One thesis
  - Collaboration links with INRIA Rennes
  - Publications at ACM SAC'11, DAIS’08, DAIS’09
  - Available on line: [http://ftaiani.ouvaton.org/GossipKit/](http://ftaiani.ouvaton.org/GossipKit/)
Outline

- Intro (just done)
- WhisperKit:
  Programming Gossip-based Systems
- ProfVis:
  Anomaly Diagnosis in Grid Middleware
- Conclusion and Outlook

Joint work with: R. Schlichting, M. Hiltunen, S. Lin, T. Ormerod, L. Ball
Studying Real-Life Reuse

- **Globus** (Argonne): ref. implementation for Grid
  - Grid Computing + Web Services

- Transition to WS stack (Version 3.9.x, 2005)
  - within a short time (a few months)
  - large, complex, collaborative (IBM, Apache,...) ≫ reuse

- But ... poor performances
  - Up to **30s** to create a simple distributed object (counter)
  - Up to **2s** for a roundtrip remote **add** operation
- **Where** does these poor performances come from?
- **What** does it tell about modern MW development?
Experience 1: Initialisation

client

clock instrumentation

create

add 3

notify 3

destroy

×5

influence of client init

container

subscribe

add 3

notify 3

destroy

×5

×5

influence of container init
Finding 1: Init is a killer

- Client
- Create
- Add
- Notify
- Destroy

Time (ms) vs. Client and Resource

- Bars represent different operations and times.
- The x5 magnification highlights significant differences in resource usage.

36
Finding 1: Init is a killer

Container init overhead (~430ms)

Client init overhead (~1.4s!)

Stabilized latency (~1.1s!)

How to analyse this?

Abstractions ➔ many levels & side effects

Reuse ➔ unfamiliar software
Exhaustive Tracing Intractable

- First attempt: tracing everything (outside the JVM libs)
  - client: 1,544,734 local method call (sic)
  - server: 6,466,652 local method calls (sic) [+time out]

- How to work around this data explosion?
Sample-based profiling

client

create
subscribe
add 3
notify 3
destroy

×5

container

Java VM

hprof

profiling data

snapshots

regular interval
Sample-based profiling

lib1.Whale .breath
lib1.Mammal.inhale
lib2.Lung .inhale
lib2.Muscle.contract
lib2.Nerve .transmit
lib3.Signal.travel

lib3.Blood .flow
lib3.Pressure.foo

lib2.Muscle.stop
lib2.Nerve .transmit
lib3.Signal.travel
Sample-based profiling

Sampling yields a set of weighted stack traces (weight reflects time spent)

Problem: **Data explosion.** On Globus:

- 55550 method invocations
- 1861 methods
- 724 classes
- 182 Java packages.
- 32 threads
How to represent the results?

Sampling yields a set of weighted stack traces (weight reflects time spent)

- **Aggregates** invocations of the same library
- **Chart w.r.t. position in call stack**

![Diagram showing stack depth and time units]
How to represent the results?

Package Activity vs. Stack Depth

Software Structure
Result on Globus

Sharp drop at length 13

Waiting related to notification management. Outside request critical path.

Layered structured for upper stack depths: architecture

Some very deep traces. Look quite regular beyond depth 28 (recursion?)
org.apache.axis predominant

Waiting related to notification management. Outside request critical path.
Findings

- **XML management issue** in apache.axis.wsdl
  - very deep recursion involving one method

- No clear culprit for overall performance
  - **Axis** 37%
  - **SOAP + XML** 44%
  - **Security (GSI, RSA)** 30%

- More generally, typical example of
  - **deep analysis**
  - **in unfamiliar software**
Interactive Visualisation

- Problem: stack depth project is **static**
  - call relationships hidden, compaction fixed

- Our take: **interactive navigation**
  - use **structural information** in dynamic data
    - e.g. `org.apache.axis.utils.ClassUtils.forName()`
  - vary ‘**local abstraction**’ level at which data is shown

- Result: collaboration with Psychology Dpt (Lancaster)
  - application of structural compaction to **dynamic data**
  - **ProfVis prototype** and explorative user study
Back to biology example
Full compaction

- Only highest level packages visible
Progressive exploration

- Different levels of compaction in different parts of graph
  ➔ including for the same package (here lib3)

Different compaction levels for same package
Demo
Evaluation

- **Goal**: explorative user study (4 users)
  - task for users: identify performance issues
  - 2 categories of programs (‘small’ and ‘large’)
  - Baseline: Textual Tree Table

- **Measures**
  - Perceived & assessed understanding
  - Interaction logs
Results: Understanding

- Perceived Understanding vs. Assessed Understanding
- Data points for different conditions:
  - Treetable small
  - Treetable large
  - Profvis small
  - Profvis large
Findings

- Disconnection perceived/assessed on large programs
  - Users **overestimate** themselves with TreeTable
  - Users **underestimate** themselves with ProfVis

- Possible cause (?):
  - TreeTable hides full scope while ProfVis does not
  - ‘false sense of mastery’
Results: Interaction

- Usage patterns seem to support this interpretation
  - users go deep w/ TreeTable, tend to hover w/ ProfVis
Summary

- High **reuse** can come with **drawbacks**

- But existing **abstractions** can help

- **Impact** of this line of research
  - Interdisciplinary links created with Psychology Dept.
  - Publications: SP&E, IEEE HPDC, ACM SoftVis
  - Talks and videos: AT&T, IBM, Cambridge, YouTube
  - Tool available on-line: [http://ftaiani.ouvaton.org/7-software/profvis.html](http://ftaiani.ouvaton.org/7-software/profvis.html)
  - Already used at Lancaster & IRISA
Outline

- **Intro** (just done)
- **WhisperKit:**
  Programming Gossip-based Systems
- **ProfVis:**
  Anomaly Diagnosis in Grid Middleware
- **Conclusion and Outlook**
Conclusion

- **Reuse and abstraction in 2 large-scale dist. systems**
  - 2 contributions: WhisperKit, Profvis
  - in 2 representative systems: gossip, grid
  - both proposal (mechanisms, abstractions) & study (tools)

- **Emerging messages**
  - feasible and beneficial (GossipKit)
  - but own challenges, that must be studied
  - by reconsidering some soft. eng. techniques (CBSE/DSL)
  - by studying existing production systems (Globus/CORBA)
Outlook: Social Networks

- Rapidly emerging
  - 800M Facebook users, 10M foursquare users
- How best to **program** fully decentralised versions?
  - Different mechanisms needed in different parts of networks
  - Different mechanisms for different features
- How to support **Adaptation / Composition / Synergies**?
The End
(Thank you)
GossipKit / Whispers


- GossipKit & Whispers, an event-oriented component framework and DSL for the development of gossip protocols
  
  http://ftaiani.ouvaton.org/GossipKit/
Middleware Analysis

- CosmOpen: dynamic reverse engineering for complex software systems [http://ftaiani.ouvaton.org/7-software/#CosmOpen](http://ftaiani.ouvaton.org/7-software/#CosmOpen)
- Profvis, an interactive visualisation tool for HPROF traces [http://ftaiani.ouvaton.org/7-software/profvis.html](http://ftaiani.ouvaton.org/7-software/profvis.html)
- Profvis Tutorial: [http://youtube.com/watch?v=luEBtRyc0F4](http://youtube.com/watch?v=luEBtRyc0F4)
- HPROF Traces: expansion and recompaction withProfvis [http://youtube.com/watch?v=G4-k1HGxA8g](http://youtube.com/watch?v=G4-k1HGxA8g).