Deep & wide
Distributed programming in a large-scale fast-moving world
Françoise Taïani
Introducing Lancaster
Lancaster Middleware Group

What we do …

- Facilitate development of new distributed systems
  - Abstractions?
  - Mechanisms?
  - Exotic environments: MANETs, WSN, routers, …
Next-gen middleware team

- Academics
  - Gordon Blair
  - Geoff Coulson
  - Francois Taiani

- Researchers
  - Paul Grace
  - Barry Porter
  - Nelly Bencomo

- Students
  - Shen Lin, Lei Liu, Vatsala Nundloll, Rajiv Ramdhany, Bholanath Surajbali, Nirmal Weerasinghe …

F. Taiani
Current research targets

- **A wide** computing world:
  - broad, pervasive, and distributed
  - e.g. MANETs, grids, clouds, VLN, social networks
  - Which **abstractions** for a **multifaceted** reality?

- **A deep** computing world:
  - drilling into the small: WSN, ECU, microchips, VMM
  - transformative jumps at any level
  - Which **abstraction** for a **multi-layer** reality?

- **Large** scale and **fast** moving
  - Issue for machines (physical) and **humans** (logical)
Outline

Helping to realise distributed systems in:

- a wide world
- a deep world
- built (for and) by humans
Outline

Helping to realise distributed systems in:

- a **wide** world
- a **deep** world
- built (for and) **by humans**
A wide world

Co-existing technologies, systems, & applications

- Should work together: interoperability
- Should thrive together: synergies
Interoperability

- **Multi-faceted** ‘Sys. of Systems’
  - together to meet global aims
- **Extreme heterogeneity**
  - Techno, network, sw, hw
- **Dynamic** composition
  - unknown until runtime

**Vision:** On-the-fly Interop.
- **Just-in-time middleware**
Example: Service Discovery Protocols
Interoperable Service Discovery

- Lancaster approach: deconstruction reconstruction
  - decompose set of SD protocols (SDP, UPnP, etc.)
  - generic component based architecture

[Carlos Flores 07]
Interoperable Service Discovery

- Lancaster approach: deconstruction reconstruction
  - **decompose** set of SD protocols (SDP, UPnP, etc.)
  - **generic** component based **architecture**
  - specific front-facing **instances** for each protocol
  - specific configuration for each **roles**

- **Next step:** for any kind of middleware paradigm
  - on-the-fly synthesis of connectors (from protocol spec.)
  - **Connect Project** (FP7), ongoing
A wide world

Co-existing systems, technologies, & applications

- Should work together: interoperability
- Should thrive together: synergies
Synergies

Goal: get independent distributed systems to cooperate

- Synergy = Local Behaviour, Global benefits
  - Local adaptation of each overlay’s behaviour
  - Goal: Improve overall efficiency
  - High payoff (global) at a minimum cost (local)

- Systematic analysis of synergies in overlays
  - assuming component-based architecture
  - both in space and time

- Challenge
  - conserve original properties of combined systems
# A Classification of Synergies

[Shen Lin 09]

<table>
<thead>
<tr>
<th>Facet</th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Overlay A, Overlay B</td>
<td>Overlay A, Overlay B</td>
</tr>
<tr>
<td>State</td>
<td>Overlay A, Overlay B</td>
<td>Overlay A, Overlay B</td>
</tr>
<tr>
<td>Service</td>
<td>Overlay A, Overlay B</td>
<td>Overlay A, Overlay B</td>
</tr>
</tbody>
</table>

Temporal Synergies

Dynamicity
Horizontal State Synergy

- **Potential Benefit:** message reduction by 1/3
- **Potential Risk:** failure to converge
Horizontal State Synergy

Simulation on Jist/SWANS (Cornel)
GossipKit Component framework

- Messages reduced by 35%
- Convergence maintained
Outline

Helping to realise distributed systems in:

- a wide world
- a deep world
- built (for and) by humans
The Wisebed approach (FP7)

Goal: large-scale WSN test beds
- configurable, extensible, federated

Approach
- **Interconnect** individual testbeds via the Internet
- **Mix** physicality, virtuality and simulation
- Offer value-added **software services**
  - A library of algorithms, mechanisms and protocols
  - A SDK tailored for heterogeneity and change [Lorien]
- Support **management** of data traces
The structure of Wisebed

Overlay Software running on the Portal Server

Testbed Portal Servers at each WISEBED testbed contributor

Each WISEBED contributor maintains their own physical testbed with different hardware equipment and setup

Overlay Network

Simulator cluster

Users connect to a virtual testbed using the same API that is used for single testbed access

Users connect to a single testbed directly using Web Services based API
The structure of Wisebed
The structure of Wisebed
Hard and soft heterogeneity

<table>
<thead>
<tr>
<th>Platform</th>
<th>CPU</th>
<th>OS</th>
<th>Language</th>
<th>Dyn. Mem</th>
<th>RAM</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSense</td>
<td>Jennic</td>
<td>iSense-FW</td>
<td>C++</td>
<td>128kB</td>
<td>92kB</td>
<td>32</td>
</tr>
<tr>
<td>ScatterWeb MSB</td>
<td>MSP430</td>
<td>SCW-FW</td>
<td>None</td>
<td>48kB</td>
<td>10kB</td>
<td>16</td>
</tr>
<tr>
<td>ScatterWeb ESB</td>
<td>MSP430</td>
<td>SCW-FW</td>
<td>None</td>
<td>60kB</td>
<td>2kB</td>
<td>16</td>
</tr>
<tr>
<td>Tmote Sky</td>
<td>MSP430</td>
<td>C</td>
<td>Physical</td>
<td>48kB</td>
<td>10kB</td>
<td>16</td>
</tr>
<tr>
<td>MicaZ</td>
<td>ATMega128L</td>
<td>C</td>
<td>Physical</td>
<td>128kB</td>
<td>4kB</td>
<td>8</td>
</tr>
<tr>
<td>TNode</td>
<td>ATMega128L</td>
<td>TinyOS</td>
<td>nesC</td>
<td>Physical</td>
<td>128kB</td>
<td>4kB</td>
</tr>
<tr>
<td>iMote2</td>
<td>Intel XScale</td>
<td>TinyOS</td>
<td>nesC</td>
<td>Physical</td>
<td>32MB</td>
<td>32MB</td>
</tr>
<tr>
<td>GumStix</td>
<td>Intel XScale</td>
<td>Emb. Linux</td>
<td>C</td>
<td>Virtual</td>
<td>16MB</td>
<td>64MB</td>
</tr>
<tr>
<td>Shawn</td>
<td>Desktop</td>
<td>Win/Linux/Mac</td>
<td>C++</td>
<td>Virtual</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>ns-2</td>
<td>Desktop</td>
<td>Win/Linux/Mac</td>
<td>C++/OTcl</td>
<td>Virtual</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

Different CPUs
Different Languages
Different amount of RAM
Different OSs
Different amount of ROM

Highly flexible SDK needed
A WSN-OS: Lorien

- Lorien’s is (part of) Wisebed’s WSN-SDK
  - pure component-based OS
  - each part strongly-separated / dynamically replaceable

- Built using OpenCom (C version)
  - minimalist / purist approach
  - OpenCom kernel is a component like any other
  - OpenCom is OS-less: runs on bare metal
  - ‘proto-kernel’

[Barry Porter 09]
Concrete realisation on TelosB

- GSC, configuration files & roles

[48KB program memory, 10KB RAM]

---

SCF.cfg

[Roles]
- ROMFS=linfs.so
- MassStorage=lexfs.so
- Loader=loader.so
- Runtime=opencom.so
- GSC=gsc.so

[Bindings]
- Loader.IIOFile->MassStorage
- Loader.IIOBlockNamed->ROMFS
- Runtime.IElfLoader->Loader
- GSC.IIOBlockNamed->ROMFS
- GSC.IIOFile->MassStorage
- GSC.IElfLoader->Loader

[Frameworks]
- Drivers.cfg
Concrete realisation on TelosB

- GSC, configuration files & roles

[48KB program memory, 10KB RAM]
Concrete realisation on TelosB

- GSC, configuration files & roles

[48KB program memory, 10KB RAM]
Protocol Framework

- MAC (MAC protocol)
- RANDOM (Random number generator)

Driver Framework

- TEMPSENSE (Sensor)
- CONCURRENCY (Concurrency model)
- RADIO (Radio driver)
- TIMER (Timer)

System Core Framework

- {Protocols}
- {Drivers}
- MASSSTORAGE (External flash filesystem)
- ROMFS (Prog. memory filesystem)
- GSC (Global System Configuration)
- RUNTIME (Component runtime)
- LOADER (Dynamic loader / linker)

Roles
- MAC=lp1.so
- Random=random.so

Bindings
- MAC.IMXRadio->Radio
- MAC.ITaskScheduler->Concurrency
- MAC.IRandom->Random
- MAC.ITimer->Timer

Frameworks
- Protocols.cfg
- [Roles]
- [Bindings]
- [Frameworks]

Applications.cfg
Applications.cfg

[Roles]
Beacon=beacon.so

[Bindings]
Beacon.IMXRadio->MAC
Beacon.ISense->TempSense
Beacon.ITimer->Timer
Dynamic Reconfiguration

- Set of **architectural conventions**
  - in particular layered acyclic architecture
  - dedicated interfaces to trigger quiescent state
  - state transfer interface (provided by implementer)
- **Primitives** MERGE, DETACH, EXTRACT and INJECT
  - complete system reconfiguration capabilities

---

**Applications.cfg**

**[Roles]**
Beacon=beacon.so

**[Bindings]**
Beacon.IMXRadio->MAC
Beacon.ISense->TempSense
Beacon.ITimer->Timer

**MyNewApp.cfg**

**[Roles]**
TimeSynch=timesynch.so

**[Bindings]**
TimeSynch.IMXRadio->MAC
TimeSynch.ITimer->Timer

**Applications.cfg**

**[Roles]**
Beacon=beacon.so
TimeSynch=timesynch.so

**[Bindings]**
Beacon.IMXRadio->MAC
Beacon.ISense->TempSense
Beacon.ITimer->Timer
TimeSynch.IMXRadio->MAC
TimeSynch.ITimer->Timer
# Dynamic reconfiguration

**Baseline**: TinyOS image-based update

⇒ both use Deluge (WSN reprogramming protocol)

<table>
<thead>
<tr>
<th>Update description</th>
<th>Data size (TinyOS) [bytes]</th>
<th>Data size (Lorien) [bytes]</th>
<th>Update time (TinyOS) [ms]</th>
<th>Update time (Lorien) [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition of application B</td>
<td>37606</td>
<td>1964</td>
<td>5816</td>
<td>992</td>
</tr>
<tr>
<td>Addition of applications B and C</td>
<td>37858</td>
<td>3327</td>
<td>5989</td>
<td>1504</td>
</tr>
<tr>
<td>Removal of application B (post B/C addition)</td>
<td>37606</td>
<td>165</td>
<td>5816</td>
<td>592</td>
</tr>
<tr>
<td>New version of random no. generator</td>
<td>37274</td>
<td>1122</td>
<td>5653</td>
<td>1584</td>
</tr>
<tr>
<td>New version of temperature sensor</td>
<td>37274</td>
<td>1443</td>
<td>5653</td>
<td>1565</td>
</tr>
<tr>
<td>New version of Deluge</td>
<td>37274</td>
<td>4387</td>
<td>5653</td>
<td>1968</td>
</tr>
</tbody>
</table>
Outline

Helping to realise distributed systems in:

- a wide world
- a deep world
- built (for and) by humans
Every hammer has two ends

We need to understand what we build
Example: recovering models

ORBacus Request Processing
one request, 2065 individual invocations, over 50 C-functions and 140 C++ classes.

How to observe and analyse this?

(with JC Fabre / MO Killijian)

F. Tai
Approach: “interferometry”

- **Limit** observation costs: *more for less*
  - capture stack traces (more)
  - limit observation points to layer boundaries (less)

- **Filter** out unneeded data: “interferometry”
  - graph manipulation script language
  - reusable filters

---

'cheap' but post-processing needed vs. no turbulence but very costly
Graph manipulation filters

Provides **variables** to store intermediary results

A number of graph **operators**

- **selection**
  
  put ::pthread_create*  G  CREATE

- **recursive extension**
  
  forward  CREATE  G

- **boolean algebra**
  
  remove ::pthread_create*  CREATE

- **temporal operations**
  
  fuse ::pthread_create*  ::pthread_start_thread*  G
Application to Orbacus

2065 individual invocations, over 50 C-functions and 140 C++ classes.

F. Taiani
Deterministic replication

- link OS level ↔ app level
- use reflective approach
- realise meta-model

Result
Outline

Helping to realise distributed systems in:

- a wide world
- a deep world
- built (for and) by humans

conclusion
What I’ve left out

A wide world: Systems of Systems

- Should evolve into another: **paradigm morphing**
  - combine **macroprogramming & components** (S. Lin)
- Supported by models: **models@runtime** (N. Bencomo)
  - model-driven engineering meets reflection

A deep world

- Flood monitoring using reconfigurable overlay framework

For humans

- anomaly comprehension [SoftViz 2010]
Conclusion

- Many (most) of the works presented are on-going
- Trend: The end of distributed programming (*)?
  - declarative (no behaviour)
  - compositional (no programming)
  - macro-programming (no distribution)
- Let’s not forget developers are humans
  - every tool has two ends
  - and so do programming technologies
  - How do we analyse the fit of a technology?

*(as we know it?)*
(Some) References

- **Interoperability**


- **Synergies**
(Some) References

- **WSN-software and embedded component-based proto-OS**
(Some) References

- **Human comprehension**


Thank you

Questions?