Advanced Distributed Systems

Reflective and Adaptive Middleware

MSc in Advanced Computer Science
Gordon Blair (gordon@comp.lancs.ac.uk)
Overview of the Session

- Reflective middleware: principles
  - Background on middleware
  - The need for adaptation
  - Introducing reflection
  - Reflective middleware

- Reflective middleware: application
  - Examples of reflective middleware
  - Case study: GridKit
  - GridKit in action

Associated Reading: see Lancaster web site for details of reflective middleware projects:
http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/
Middleware

- **Rationale**
  - Provide a high level programming abstraction (but which one?)
  - Hide the underlying complexity associated with distributed systems (including the underlying heterogeneity)
A History of Middleware

- First generation middleware
  - Based exclusively on the *client-server model*, and approaches such as 2- or 3- tier architectures
  - Examples include the Open Group’s DCE

- Second generation middleware
  - Based on *distributed object technology*
  - Examples include CORBA and Java RMI

- Third generation middleware?
  - Based on emerging *component technology*, with more general n-tier architectures (see session 4)
  - Also web services as a ‘competing’ approach
Other Styles

- Resource discovery platforms (e.g. Jini)
- Group communication services (e.g. Jgroups)
- Search middleware
- Cloud middleware
- Distributed file systems
- Publish-subscribe systems (e.g. JMS)
- Distributed transaction services
- Distributed document-based systems
- Agent-based systems
- P2P technology
- Etc.
Discussion Point

Is middleware fit for purpose for the 21st century

???
The history of Computer Science has been dominated by the search for abstraction (quite rightly)
- In programming language design
- In database design
- In operating system design
- In distributed systems

But.....
- Is abstraction *on its own* always a good thing?
- Is there a role for a more open systems philosophy
  - e.g. based on reflection
Open Implementation

“A system with an open implementation provides (at least) two linked interfaces to its clients, a base-level interface to the system’s functionality similar to the interface of other such systems, and a meta-level interface that reveals aspects of how the base-level interface is implemented”

Rao, 1991
From Open Implementation to Reflection

\[ \text{Reflection} = \text{Open Implementation} + \text{Causally Connected Self Representation (CCSR)} \]

(cf. the art of the meta-object protocol)
Styles of Reflection

- **Structural vs behavioural**
  - Structural reflection is the representation of *(static)* structure of the system
  - Behavioural reflection is the representation of *ongoing activity*

- **Procedural vs declarative**
  - In procedural reflection, meta-programming involves *direct manipulation* of the system
  - In contrast, declarative reflection involves manipulation of a more *abstract representation*
Why Reflection?

- Support for introspection
  - The ability to inspect the structure and behaviour of the system
    - e.g. dynamic monitoring or accounting

- Support for intercession
  - Towards system families
    - e.g. build-time configuration
  - Short term dynamic re-configuration
    - e.g. changing protocol configuration
  - Longer term evolution
    - e.g. adding new multimedia service
  - Autonomic systems
    - e.g. achieving self-* properties
Reflective Languages

- Reflection and Lisp
  - 3-Lisp, ObjVlisp, Loops, Flavours, CLOS

- Reflection and concurrent languages
  - AL-1/D, RbCL, Cognac, ABCL/R family

- Reflection and C++
  - MPC+, Open C++, Iguana

- Reflection and Java
  - Java Core Reflection API, Reflective Java, Open Java, Javassist, Open JIT, Kava, Dalang, ...
A Closer Look at CLOS

What is CLOS?

- An object-oriented extension to Lisp (featuring \texttt{defclass}, \texttt{defgeneric} and \texttt{defmethod})
- Supports the standardised CLOS MOP [Kiczales]

Reflective features

- Reifies above definitions via \texttt{standard-class}, \texttt{standard-generic-function} and \texttt{standard-method}
- Also supports additional meta-level generic functions, e.g. computing the class precedence list
A Simple Example

Creates a sub-class of standard class with an extra slot for counting

(defclass counted-class (standard-class)
  ((counter : initform 0 )))

(defmethod make-instance :after((class counted-class)&key)
  (incf (slot-value class 'counter )))

(defclass counted-x (x) (:meta-class counted-class)
  Specialise make-instance to increment counter

Finally, create an instance with this meta-class

(:meta-class counted-class)
... and the ABCL/R Family

What is ABCL/R

- **ABCL/1**: the underlying concurrent language based on (single-threaded) objects communicating via message passing
- **ABCL/R**: initial reflective version whereby every object $x$ has a meta-object $\uparrow x$
- **ABCL/R2**: hybrid architecture featuring meta-objects (for structural reflection) and meta-groups (for behavioural reflection)
The ABCL/R MOP

■ Structural reflection
  ➔ Variables provided at the meta-level for:
    ♥ *Scriptset*: the set of scripts (methods)
    ♥ *State*
    ♥ *Evaluator* (the associated interpreter)
    ♥ *Queue* (of incoming messages)

■ Behavioural reflection
  ➔ Scripts provided in the meta-level for the events of
    *message arrival, receipt* into the queue, *acceptance*,
    *execution* and *end of execution*
Other Reflective Systems

- Operating systems
  - Adaptive or extensible operating systems
    - Spin, Synthetix, Exokernel/ Aegis, Spring, the Cache Kernel, \(\mu\)Choices, etc
  - Reflective operating systems
    - *Apertos* (now Aperios), Aeon (using Iguana), Merlin, Tunes

- Distributed systems
  - *CodA*: a reflective distributed object-oriented platform
  - R-OK: reflective distributed object management
  - Barga and Pu’s reflective transaction service
  - Etc.
BUT WHAT IS REFLECTIVE MIDDLEWARE

???
The Lancaster Approach: A Marriage of Three Technologies

- Reflection
  - Use reflection to access structure and behaviour of the underlying middleware platform

- Components
  - Apply component-oriented programming at base and meta levels
  - Software (e.g. middleware) built from components
    - A distributed component model would be built from components

- Component frameworks
  - Domain-specific ‘life-support environments’ for plug-in components
The OpenCOM v2 Component Model

- Central concepts:
  - component
  - capsule
  - interface
  - receptacle
  - binding

- Use of IDL interfaces to give language independence
OpenCom v2 (continued)

template
load(comp_type name);
comp_inst
instantiate(template t);
status
unload(template t);
status
destroy(comp_inst comp);
comp_inst
bind(interface i, receptacle r);
status
putprop(entity e, key k, opaque value);
Opaque
getprop(entity e, key k);

Reflective extensions
platform extensions

OpenCOM runtime

Deployment env (hardware and/or software)

Target system

Built in terms of component frameworks

Capsule API
**The Reflective Extensions**

- Reflection yields *openness* (inspection | adaptation | extension)

- Use of multiple optional orthogonal *reflective meta-models*
  - Architecture
    - Represent the topology of a composition of components within a capsule
    - Used for topological inspection, adaptation, extension
    - ‘Graph-oriented’ meta-interface
  - Interception
    - Interpose interceptors in bindings
    - Useful, e.g., for lightweight monitoring
  - Interface
    - Dynamically discover details of a component’s interfaces/receptacles
    - Dynamically invoke dynamically-discovered interfaces
Component Frameworks (CFs)

- Systems are built uniformly in terms of CFs
  - Reusable run-time “life support environments” for plug-in components in a particular area of application
  - Protocol stacking | routing | buffer management | job scheduling ...

- CFs offer
  - Coarse-grained system structuring
  - System configurability
  - Constraint on pluggability, reconfigurability
Applying our Approach (1 of 2)

- Multimedia
  - Early work in the France Telecom funded SUMO project (1998-2001) | EPSRC funding
  - Developed an RM-ODP compliant platform for distributed multimedia computing | based on the Chorus micro kernel
  - OpenCOM first emerged at this time

- Mobile computing
  - Lucent-funded ReMMoC project | Reflective Middleware for Mobile Computing
  - Addressed middleware heterogeneity in service-based mobile computing
    - RPC, pub-sub, messaging, …
    - SLP, uPnP, DISCO, …
  - Now ReMMoC II
Applying our Approach (2 of 2)

- Programmable networks
  - EPSRC NETKIT project | application to Network Processors
  - EPSRC Virtual Routers project

- Embedded systems and sensor networks
  - EU Cortex project | middleware for real-time control
    - Autonomic control of vehicles | sentient object paradigm
  - EU RUNES IP
    - Applying (delta of) OpenCOM approach in "reconfigurable ubiquitous networked embedded systems"
    - OpenCOM on sensor motes
Applying our Approach (3 of 3)

- Recent focus on highly heterogeneous ‘Grid’/ cloud environments
- Diverse infrastructures
  - Networks: clusters | LAN-based | WAN-based | infrastructure and ad-hoc wireless | sensor nets ...
  - Nodes: supercomputers | workstations | PDAs | sensors ...
- Diverse “interaction types”
  - RPC and SOAP messaging | (un)reliable multicast | media-streaming | publish-subscribe | tuple spaces | P2P resource location or sharing | workflow ...
## Introducing Gridkit

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Service discovery</th>
<th>Resource discovery</th>
<th>Resource mgmt</th>
<th>Resource monitoring</th>
<th>Security</th>
</tr>
</thead>
</table>

### Pluggable APIs (e.g. WS)

### Open Overlays framework

### Reflective component model runtime

\[ \text{OpenCOM v2} \]
Open Overlays

- Network overlays are virtual communication structures logically laid over physical network
  - Hide network heterogeneity
  - Provide a range of different networking abstractions not supported in the network
    - peer-to-peer groups, distributed hash tables, application-level multicast, etc

- Open overlays is a configurable and reconfigurable framework integrated into broader middleware architectures
  - Supports (flexible) virtualization of the network resource,
  - the co-existence of multiple (physical or) virtual networking abstractions,
  - and potentially the layering of virtual network abstractions to achieve desired network services through composition.
The Overlays Framework

- Host one or more overlay implementations
  - For each overlay: need local node implementation, e.g.,:
    - Join procedure
    - Overlay maintenance
    - Message routing

- Developing overlays for Gridkit
  - 3 components
    - Control – Implements the topology maintenance
    - Forward – Implements the routing algorithm
    - State – maintains the overlay’s state information
  - Configurability
    - Common interfaces support plugging in of overlays
  - Reconfigurability
    - Replace routing & control algorithms
Example Configuration

Interfaces for virtualisation

Overlay plug-ins
a) Layering for richer services
b) Re-use
c) Parallel deployment

Overlay Multicast
IDistributedHashTable

IMulticast
IDistributedHashTable

Overlay Multicast
DHT

TBCP
Scribe
Chord DHT

Chord KBR

TCP transport
UDP transport
AODV network

Adv. Dist. Systems
G. Blair/ F. Taiani
### Available overlay plug-ins

<table>
<thead>
<tr>
<th>Overlay Name</th>
<th>Description and configurability options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chord KBR</td>
<td>A KBR overlay based on Chord</td>
</tr>
<tr>
<td>DHT</td>
<td>Data storage overlay</td>
</tr>
<tr>
<td>Pastry KBR</td>
<td>A KBR overlay based on Past</td>
</tr>
<tr>
<td>Failure</td>
<td>Monitoring overlay; detects and disseminates node</td>
</tr>
<tr>
<td>Monitor</td>
<td>failure info</td>
</tr>
<tr>
<td>SCAMP</td>
<td>Scalable Group Membership overlay with gossip-based Forwarding</td>
</tr>
<tr>
<td>Scribe</td>
<td>Multicast used atop any KBR overlay</td>
</tr>
<tr>
<td>Spanning Tree</td>
<td>Tree overlay for fan-in routing</td>
</tr>
<tr>
<td>TBCP</td>
<td>Wide area multicast overlay</td>
</tr>
</tbody>
</table>
Gridkit on WSNs

- Gridkit + Gumstix = Gridstix

- Depth and Camera Sensors
Gridkit@Sourceforge

http://sourceforge.net/projects/gridkit
Expected Learning Outcomes

At the end of this third unit:

- You should be able to discuss the basic concepts, terminology and motivation behind reflection
- You should have an initial understanding of interpretation of reflection in languages and systems
- You should have a stronger appreciation of what is meant by reflective middleware
- You should be able to discuss the main concepts underpinning the Lancaster approach to reflective middleware architecture, including the key role of components (more in session 4)
- You should be able to discuss how reflective middleware can contribute in various application domains including Grid computing