

Adaptive Middleware (mainly: at Lancaster University)

Geoff Coulson

Who am I?

- ◆ Prof. Geoff Coulson
 - email: geoff@comp.lancs.ac.uk
 - home page and publications:
<http://www.comp.lancs.ac.uk/department/staff.php?name=geoff>
- ◆ I work with Gordon Blair and Francois Taiani in Lancaster's "middleware group" (part of our "Networked and distributed systems theme")
 - <http://www.comp.lancs.ac.uk/computing/research/mpgreflection/>
- ◆ The group currently comprises around 10 researchers and PhD students

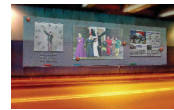
Lancaster University's Computing Dept



- ◆ Overview
 - Research led department with a strong international reputation
 - Strong practical focus with excellent links to industry
 - Over £10m currently held in research grants
- ◆ Current size
 - 30 academic staff
 - 60 research associates
 - Over 100 registered PhD students

Our 4 research themes (for context)


- ◆ Networked and distributed systems
 - Blair, Coulson, Hutchison, Pink, et al
- ◆ Software systems engineering
 - Sommerville, Sawyer, Rashid, et al
- ◆ Mobile and ubiquitous systems
 - Davies, Gellersen et al
- ◆ Cooperative and interactive systems
 - Dix et al



Structure of the morning (9:30-12:30)


- ◆ Part I: Reflection and reflective middleware
- ◆ Part II: The Lancaster adaptive middleware approach
- ◆ Part III: Applying the approach

Part I: Reflection and reflective middleware

LANCASTER UNIVERSITY 


Overview

- ◆ Some definitions
- ◆ Introduction to computational reflection
- ◆ Reflection and object-oriented computing
- ◆ Reflective languages and systems
- ◆ Reflective middleware
- ◆ Case study: OpenORB



InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk


LANCASTER UNIVERSITY 

Open implementation -> 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


Rao, 1991

cf. ‘black-box’ philosophy of software development




InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY 

Open implementation -> reflection




Reflection = *Open Implementation*
+ *Causally Connected Self Representation (CCSR)*

Reflection is the capability of a system to reason about and act upon itself. A reflective system contains a representation of its own behaviour, amenable to examination and change, and which is causally connected to the behaviour it describes. “Causally-connected” means that changes made to the system’s self-representation are immediately reflected in its actual state and behavior, and vice-versa

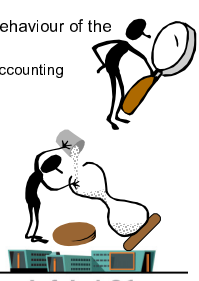
InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY 


Why reflection?

- ◆ Support for introspection
 - The ability to inspect the structure and behaviour of the system
 - e.g. dynamic monitoring, debugging or accounting
- ◆ Support for adaptation
 - Short term dynamic re-configuration
 - e.g. changing protocol configuration
 - Longer term evolution
 - e.g. adding a new transactions service



InfoLab21
Computing Department


InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY 

Some key terminology


- ◆ A reflective system or language has a *base-level* and a *meta-level* (cf *meta-space*), the latter supporting *meta-level programming*
- ◆ Going to the meta-level is called *reification*
- ◆ Going back down to the base level is called *absorption*
- ◆ The interface to the meta-level (means of reaching the CCSR) is referred to as the *meta-object protocol*

Also, infinite towers of reflection



InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk


LANCASTER UNIVERSITY 

Styles of reflection

- ◆ Structural vs behavioural
 - Structural reflection is the representation of the *structure* of the system (static)
 - 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*

InfoLab21
Computing Department


InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk


LANCASTER UNIVERSITY 


Reflection and object-orientation

“Reflective techniques make it possible to *open up* a language’s implementation without revealing unnecessary implementation details or compromising portability; **and** object-oriented techniques allow the resulting model of the language’s implementation and behaviour to be locally and incrementally adjusted”

Kiczales, 1991






InfoLab21
Computing Department


LANCASTER UNIVERSITY 

Reflection and OO (cntd)

- ◆ Options for scope
 - The *meta-object* approach
 - inspection and adaptation on a **per object** basis
 - The *meta-class* approach
 - inspection and adaptation on a **per class** basis
 - The *meta-group* approach
 - inspection and adaptation on a **per group** basis
- ◆ Granularity
 - *Fine-grained* or *coarse-grained* adaptation?






InfoLab21
Computing Department


LANCASTER UNIVERSITY 

Two pragmatic issues for reflective systems

- ◆ Integrity
 - Does the reflective language or system remain in a ‘consistent’ state if run-time changes are made at the meta-level?
- ◆ Performance
 - Can a reflective language or system deliver performance that is comparable to a ‘standard’ system or language?






InfoLab21
Computing Department

LANCASTER UNIVERSITY 

Some example 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, ...



InfoLab21
Computing Department


LANCASTER UNIVERSITY 

Example: the ABCL/R family

- ◆ What is ABCL/R
 - *ABCL/1*: a concurrent language based on (single-threaded) objects communicating via message passing
 - *ABCL/R*: a reflective version in which every object x has a meta-object $\uparrow x$
 - *ABCL/R2*: a hybrid architecture featuring meta-objects (for structural reflection) and meta-groups (for behavioural reflection)





InfoLab21
Computing Department

LANCASTER UNIVERSITY 

The ABCL/R2 MOP

- ◆ Structural reflection
 - These variables are provided at the meta-level:
 - *Scriptset* (the set of scripts (i.e. methods))
 - *State* (application specific state of the object)
 - *Evaluator* (the associated interpreter)
 - *Queue* (of incoming messages)
- ◆ Behavioural reflection (on meta-groups)
 - Scripts can be injected at the meta-level to handle the events of: *message arrival*, *receipt into the queue*, *message acceptance*, *execution* and *end of execution*


InfoLab21
Computing Department

Reflective operating systems

- ◆ Adaptive or extensible operating systems
 - Spin, Synthetix, Exokernel/ Aegis, Spring, the Cache Kernel, μ Choices, etc
 - Adaptive but not truly reflective
- ◆ Reflective operating systems
 - Apertos/Aperios, Aeon (using Iguana), Merlin, Tunes

What makes an operating system reflective?



Example: Apertos

- ◆ Reflection in Apertos
 - *Objects* in Apertos offer a base-level service
 - *Meta-spaces* offer an execution environment; contain a number of *meta-objects* (cf. ABCL/R2 meta-groups)
- Meta-space adaptation in Apertos
 - Objects can *migrate* to a new meta-space (coarse-grain)
 - E.g. a meta-space offering atomicity, debugging facilities, etc
 - Changes can be made to meta-space (fine-grain)
 - E.g. change a scheduling policy

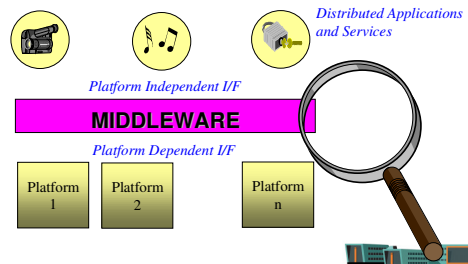


Reflection and distributed systems

- ◆ A number of reflective distributed systems platforms have been developed, such as
 - *CodA*: a reflective distributed object-oriented platform
 - R-OK: reflective distributed object management
 - Barga and Pu's reflective transaction service
 - Etc.
- ◆ Later work has been carried out under the banner of *reflective middleware*



What is reflective middleware?



Definitions

- ◆ The term **Middleware** refers to a set of services that reside between the application and the operating system and aim to facilitate the development of distributed applications
- ◆ The term **Reflective Middleware** therefore describes the application of reflection to the engineering of middleware systems in order to achieve openness, configurability and reconfigurability/ adaptivity
- ◆ See
 - http://www.computer.org/portal/site/dsonline/menutitem_9ed3d9924aeb0dcd82ccc6716bbe36e0/index.jsp?&pName=ds_o_level1&path=dsonline/topics/middleware&file=RMarticle1.xml&xsl=article.xsl&
 - Proc. of Reflective Middleware workshop series: <http://www.ics.uci.edu/~arn/>



Analysis of mainstream middleware

- ◆ Largely follows a 'black-box' philosophy
 - Lack of configurability and re-configurability
 - Increasingly seen as a major problem
- ◆ Some responses:
 - OMG: minimal and real-time CORBA, portable interceptors, the POA, policies, ...
 - Microsoft: COM support for evolution, DCOM custom marshalling, COM+ interceptors, .NET introspection, ...
 - SUN: Java Core Reflection, ...

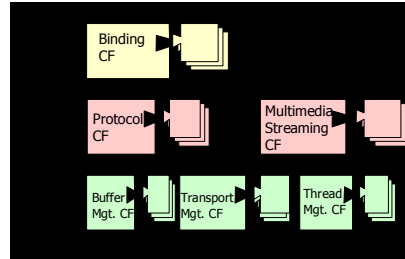


Reflective middleware example: OpenORB

- ◆ Lancaster's 1st generation reflective middleware
- ◆ A combination of three elements
 - reflection
 - use reflection to access structure and behaviour of the underlying middleware platform
 - software components
 - apply component-oriented programming at base and meta levels
 - *see more later*
 - component frameworks
 - domain-specific 'life-support environments' for plug-in components
 - *see more later*

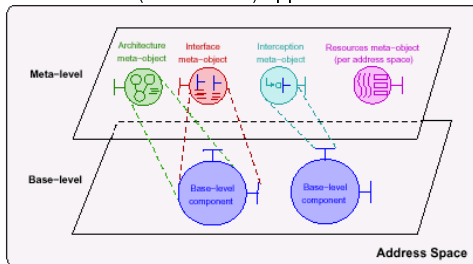


OpenORB's structural architecture



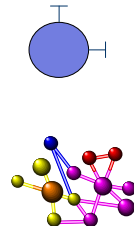
OpenORB's reflection architecture

- ◆ A multi-model (multi-CCSR) approach



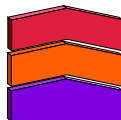
Structural reflection in OpenORB

- ◆ The interface meta-model
 - Supports *introspection* on the external representation of a component
- ◆ The architectural meta-model
 - Supports *introspection* and *adaptation* of the underlying software architecture
 - Software architecture = component graph + architectural constraints



Behavioural reflection in OpenORB

- ◆ The interception meta-model
 - Supports the dynamic insertion of interceptors
 - Pre- and post- method call
- ◆ The resources meta-model
 - Supports introspection and adaptation of the allocation of resources and related resource management
 - Sophisticated CCSR based on the concepts of *tasks* and *resources*



Example APIs

Interface MOP (introspection only)	
Operation signature	Description
<code>get_interfaces()</code>	Returns the identifiers of all the interfaces supported by a component.
<code>get_style(identifier)</code>	Returns the style of the identified interface (i.e. either operational, stream or signal).
<code>get_operations(identifier)</code>	Returns a list with the names and types (supersides) of all attributes in the identified interface.
<code>get_operations(identifier, name)</code>	Returns a list with the names of all intercessors (either operations, flows or signals) in the identified interface.
<code>get_full_description(identifier)</code>	Returns the full description of the named intercession (operation, flow or signal) of the identified interface.
<code>get_value(identifier, name)</code>	Returns the actual value of the named attribute in the current instance of the identified interface.
<code>set_value(identifier, name, value)</code>	Set the value of the named attribute in the current instance of the identified interface.
<code>enable_operation(identifier, name)</code>	Enables a dynamic call to be made to an operation of the identified interface. (Does not work for stream and signal interfaces.)

Example APIs (cntd)

Architectural MOP	
Operation signature	Description
Operations for introspection	
<code>ObjectGraph get_obj_graph();</code>	Returns the complete representation of the component graph that describes the structure of the base-level configuration.
<code>ISeq get_introspect_components();</code>	Returns a list with the identifiers of the components that constitute the base-level configuration.
<code>BoundComponentSeq get_bound_components(in ID comp_id);</code>	Returns a list with information (component id and interface names) of all the components bound to the one identified as the argument.
<code>ISeq get_introspect_bindings();</code>	Returns a list with the ids of all binding objects that are part of the base-level composition.
<code>ArchStyle get_arch_style();</code>	Returns the architecture style of a composite component.
<code>ISeq get_style_rules();</code>	Returns the sequence of rules of the composite architecture.
<code>boolean check_rule_consistency(in Rule rule);</code>	Checks if a given rule is conformant with the architecture style rules in use by an architecture.
<code>SeqOfSeq get_dependency_constraints();</code>	Gets the sequence of dependency constraints associated with the architecture, i.e. constraints between two or more components (such as if one is replaced another must also be replaced).

Example APIs (cntd)

Architectural MOP	
Operation signature	Description
Operations for adaptation	
<code>in ID connect(in ID id1, in ID id2);</code>	Establish a local binding between the two identified interfaces.
<code>in ID disconnect(in ID id);</code>	Break the local binding between the two interfaces.
<code>in ID create(in ID id, in ID id1, in ID id2);</code>	Create and insert a new component into the base-level composition, with the given name and in the specified location (given by one or more interfaces to which the new component should be bound, if any interfaces are given, the new component is left unbound).
<code>in ID remove(in ID id);</code>	Remove the component from the composition, or binding the adjacent interfaces of neighbouring components, if appropriate and according to the given sequence of constraints in the abstract.
<code>in ID replace(in ID id, in ID id1, in ID id2);</code>	Replace an existing component with a new component of the given type (the old component is deleted).
<code>in ID modify(in ID id, in ID id1, in ID id2);</code>	Modify the interface of an internal component as a new instance of the composite component.
<code>in ID modify_rule(in ID id, in ID id1, in ID id2);</code>	Change the boundary for a new set of modifications that will be introduced in the configuration graph (over the interaction).
<code>in ID complete();</code>	Complete the transaction.
<code>in ID rollback();</code>	Roll back the transaction.
<code>in ID get_arch_style();</code>	Get the architecture style which contains the configuration.
<code>in ID get_style_rules();</code>	Get the style rules in the architecture.
<code>in ID get_rule_constraints();</code>	Retrieve a rule constraint from the rule constraints.
<code>in ID add_rule_constraint(in ID id, in ID id1, in ID id2);</code>	Retrieve the specified rule and add the new rule.
<code>in ID del_rule_constraint(in ID id, in ID id1, in ID id2);</code>	Del a dependency constraint for a given architecture.

Example APIs (cntd)

Interception MOP	
Operation signature	Description
<code>void add_pre_interceptor(in InterceptorDescriptor interceptor, in Name interceptor_name);</code>	Add an interceptor to the base-level interface, in order to trap incoming messages and introduce additional behaviour to be executed <i>before</i> the interaction is actually processed.
<code>void add_post_interceptor(in InterceptorDescriptor interceptor, in Name interceptor_name);</code>	Add an interceptor to the base-level interface, in order to trap incoming messages and introduce additional behaviour to be executed <i>after</i> the interaction is actually processed.
<code>void del_interceptor(in Name interceptor_name);</code>	Remove the named interceptor from the base-level interface.

Example APIs (cntd)

Resources MOP	
Operation signature	Description
Operations on Factory	
<code>AbstractResource createResource(in Size size, in Policy adminPolicy, in Param adminParam);</code>	Create an abstract resource of a given size and associates a management policy with it; scheduling parameters are passed in case of the creation of processing resources.
Operations on Scheduler	
<code>in ID suspend(in ResourceID abstractResourceID);</code>	Suspend an abstract processing resource.
<code>in ID resume(in ResourceID abstractResourceID);</code>	Resume an abstract processing resource.
Operations on Abstract Resource	
<code>Resource getID();</code>	Get lower level resources.
<code>in ID setLevel(in Resource level);</code>	Set lower level resources.
<code>in ID getLevel(in Resource level);</code>	Get higher level resources.
<code>in ID setHighLevel(in Resource level);</code>	Set higher level resources.
<code>in ID getHighLevel(in Resource level);</code>	Get the manager of this resource.
<code>Manager getManager();</code>	Set the manager of this resource.
<code>in ID setManager(in Manager manager, in Param adminParam);</code>	Get the factory of this resource.
<code>Factory getFactory();</code>	Set the factory of this resource.
<code>in ID setFactory(in Factory factory);</code>	

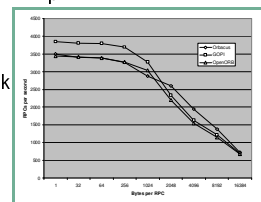
Three generations of Lancaster's reflective middleware

- ◆ Generation 0
 - Python based rapid prototype(s)
- ◆ Generation 1
 - OpenORB: based on a lightweight and reflective component model (OpenCOM v1)
 - Construction of a CORBA compliant platform
- ◆ Generation 2 (*see more later*)
 - OpenCOM v2
 - Applied in series of contrasting application domains



Integrity and performance

- ◆ For integrity management we have employed plug-in 'validator components'
- ◆ Performance of OpenORB
- ◆ More sophisticated approaches in more recent work (*see later*)



Expected learning outcomes

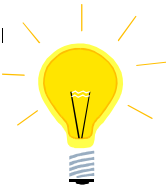
- ◆ You should be able to discuss basic reflection concepts and terminology, and understand the motivation behind reflective languages and systems
- ◆ You should have an initial understanding of reflection in middleware environments
- ◆ You should be able to discuss the main concepts underpinning the OpenORB middleware architecture, including the multi-model reflection approach

Part II: The Lancaster adaptive middleware approach

[acknowledgements are here due to our partners in the EU FP6 RUNES project—esp. Cecilia Mascolo at UCL and G.P. Picco at U Trento]

Overview

- ◆ Goals and overall approach
- ◆ The OpenCOM v2 component model
- ◆ Component frameworks, reflective extensions, and platform extensions
- ◆ A flavour of programming using OpenCOM v2

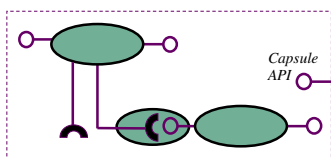


Goals and overall approach

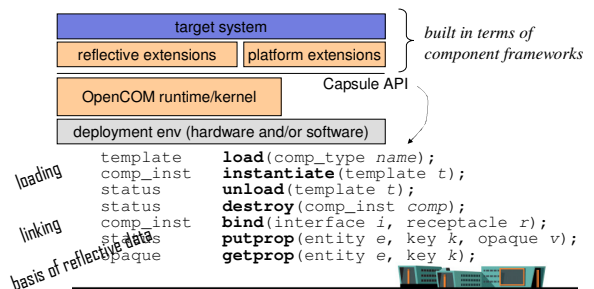
- ◆ Goal of the research
 - The design of generic middleware that can be applied in a wide range of application areas
 - Configurable | re-configurable | *self-organising*
 - 'Deep' middleware
- ◆ Overall approach
 - Based on a generic *runtime component kernel* called OpenCOM v2
 - system and language independent | low-level | minimal | efficient | uniform basis for higher layers
 - Build higher software layers in terms of
 - components | component frameworks | reflection
 - Can build any desired middleware API in these terms
 - e.g. web services based, ...

The OpenCOM v2 component model

- ◆ Central concepts: *component* | *capsule* | *interface* | *receptacle* | *binding*
- ◆ Use of IDL gives programming language independence

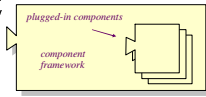


The OpenCOM v2 architecture and API



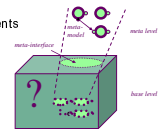
Component frameworks (CFs)

- ◆ Middleware is 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
 - A pattern for coarse-grained system structure
 - System configurability and reconfigurability
 - Constraint on pluggability, reconfigurability
- ◆ Have investigated various instantiations
 - 'Root' component with plug-in 'validator'
 - ADL | constraints
 - MDSD approach



The reflective extensions

- ◆ Reflection yields openness (inspection | adaptation)
- ◆ Use of multiple *optional* orthogonal reflective meta-models
 - Architecture
 - Represent the topology of a composition of components within a CF or capsule
 - Used for topological inspection and adaptation
 - '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



The platform extensions

- ◆ Caplets
 - 'Subscopes' within a capsule
 - Used to
 - support heterogeneous programming languages
 - isolate untrusted components
 - support heterogeneous system infrastructures
 - cf. meta groups
- ◆ Plug-in loaders
 - Perform 'special' loading-related functions | e.g., security/ safety checks before loading
 - Different loaders may be associated with different caplets
- ◆ Plug-in binders
 - For both intra- and inter-caplet binding (all within a single capsule)
 - Exploit available communication mechanisms (interrupts | special buses | shared memory ...)

Example: developing a component using the C/Linux version of OpenCOM

- ◆ In the C/Linux environment, components are represented as Linux .so files
- ◆ The developer implements the set of functions that appear in each of the interfaces supported by the component
 - each function takes 'Component' as its first argument - this refers to the component instance being invoked
- ◆ In each component, the developer provides two standard C functions:
 - `int construct(Component *comp, int argc, char argv[]);`
 - this is called by the runtime kernels *instantiate()* API
 - instantiates the Receptacles and Interfaces of the new component instance
 - `argc` and `argv` are used to pass initialization state to the new component
 - If component interfaces need to employ persistent state, this is declared in a C struct and registered with the OpenCOM runtime
 - `int destruct(Component *comp);`
 - called by the runtime's *destroy()* API
 - reclaims (using *free()*) any per-component instance state that was allocated by *construct()*

An 'adder' component (adder.h)

```

/* 1. IDL */

Interface Adder {
    int add(in int x, in int y);
    int increment(void);
};

/* 2. adder.h - Adder Component with a single Interface */
#include "opencom.h"

/* Interface struct for the Adder interface (no receptacles in this case) */
typedef struct adder {
    int (*add)(Component *comp, int x, int y);
    int (*increment)(Component *comp);
} Adder;

/* State struct for the 'adder' component */
typedef struct as {
    int current; /* for use by increment() */
} Adder_State;
    
```

An 'adder' component (adder.c)

```

/* adder.c - adder component with a single interface of type Adder */
#include "opencom.h"
#include "adder.h"

int construct(Component *comp)
{
    Adder *ai;
    Adder_State *as;

    /* STEP 1: allocate, initialize, and register the receptacle(s) (none in this component) */
    /* STEP 2: allocate, initialize, and register the interface(s) */
    VERIFY(ai = (Adder *)malloc(sizeof(Adder)), ERR, "adder.construct(): malloc");
    ai->add = add; /* assign function pointer */
    ai->increment = increment; /* assign function pointer */
    VERIFY(ocm_register_iface(comp, "Adder", ai, sizeof(Adder)), ERR, "adder.construct(): ocm_register_iface");
    VERIFY(ocm_register_state(comp, "Adder_State", as, sizeof(Adder_State)), ERR, "adder.construct(): ocm_register_state");
    return OCM;
}

int destruct(Component *comp)
{
    return OCM; /* Free any non-top-level state-related dynamic data - but there is none here */
}

int add(Component *comp, int a, int b)
{
    return a + b;
}

int increment(Component *comp)
{
    Adder_State *state;
    ocm_getstate(comp, (void **)&state);
    return ++state->current;
}
    
```

LANCASTER UNIVERSITY

A client for the adder (client.h/client.c)

```

/* 1. IDL */
interface Client {
    int run(void);
}

/* 2. client.h */
#include "opencl.h"
typedef struct client {
    Client;
    int (*run) (Component *comp);
} Client;

/* 3. client.c */
#include "client.h"
#include "adder.h"

int construct(Component *c); /* usual stuff - including: create an "Adder" receptacle for this component */
int destruct(Component *c); [...]

int run(Component *comp)
{
    Receptacle *r;
    int i, t;

    VERIFY(ocm_getreceptacle(comp, "Adder", &r), ERR, "client.run(): get_receptacle.");
    for (i=0; i<4; i++) {
        t = INVOKE_RECEPTACLE(r, adder, add, i, i);
        VERIFY("Adder: add(%d, %d) %d", i, i, t);
        t = INVOKE_RECEPTACLE(r, adder, increment);
        printf("Adder: increment() = %d\n", t);
    }
    return OK;
}

```

InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY

Putting it all together

```

main()
{
    ComponentType *adderCT, *clientCT;
    Component *adder, *client;
    Receptacle *iadder;
    Interface *iadder, *iclient;
    Component *com;

    printf("Starting...\n");
    /* initialise */
    ocm_init();

    /* load the "adder" and "client" ComponentTypes */
    VERIFY(ocm_load("adder.so", &adderCT) != ERR, ERR, "main: ocm_load adder.");
    VERIFY(ocm_load("client.so", &clientCT) != ERR, ERR, "main: ocm_load client.");

    /* instantiate adder and client components */
    VERIFY(ocm_instantiate(adderCT, &adder, 0, (char **)0) != ERR, ERR, "main: ocm_instantiate adder.");
    VERIFY(ocm_instantiate(clientCT, &client, 0, (char **)0) != ERR, ERR, "main: ocm_instantiate client.");

    /* get the Adder Receptacle from the client instance */
    VERIFY(ocm_getreceptacle(client, "Adder", &iadder), ERR, "main: get_receptacle adder.");

    /* get the Adder Interface from the adder instance */
    VERIFY(ocm_getinterface(adder, "Adder", &iadder), ERR, "main: get_interface adder.");

    /* connect them using the default Connector */
    VERIFY(ocm_connect(&iadder, iadder, "default_connector.so", &com), ERR, "main: ocm_connect.");

    /* get the Client interface from client and call the "run" method on it */
    VERIFY(ocm_getinterface(client, "Client", &iclient), ERR, "main: get_interface client.");
    INVOKE_INTERFACE(iclient, Client, run);

    /* when we get back here, client has done making calls on adder: so clean up... */
    VERIFY(ocm_destroy(adder) != ERR, ERR, "main: ocm_destroy dummy adder.");
    VERIFY(ocm_destroy(client) != ERR, ERR, "main: ocm_destroy adder.");
    VERIFY(ocm_unload(adderCT) != ERR, ERR, "main: ocm_unload adderCT.");
    VERIFY(ocm_unload(clientCT) != ERR, ERR, "main: ocm_unload clientCT.");
    printf("Done!\n");
}

```

InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

only used in main() as bootstrap (because main() isn't a component)

LANCASTER UNIVERSITY

Summary

- We apply the principle of 'components everywhere'
 - a simple component-based programming model is used uniformly throughout the 'stack' and in all deployment environments
 - reflective extensions and platform extensions are optional
- We build systems (and sub-systems) from component frameworks
 - reusable software architectures
 - give structure, tailorability and constraint
 - optionally and dynamically (un)deployable
- Build systems by deploying and configuring component frameworks and plug-ins
- Adapt systems by reconfiguring component frameworks (with the aid of the reflective extensions)

InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY

Expected learning outcomes

- You should understand the concepts underlying the Lancaster approach to adaptive middleware provision
- You should have a sense of how OpenCOM-based systems are programmed

InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY

Part III: Applying the approach


InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

LANCASTER UNIVERSITY

Overview

- Applying our approach (historical)
- The Gridkit middleware
- Sensor network-related projects
- The RUNES project



InfoLab21
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infofab21.lancs.ac.uk

Application areas (historical)

- ◆ Multimedia
 - Early work in the France Telecom funded **SUMO** project (1998-2001)
 - Developed an RM-ODP compliant platform for *distributed multimedia computing* | based on the Chorus micro kernel
 - OpenCOM first emerged at this time (based initially on MS COM)
- ◆ 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, ...
 - ReMMoC II is ongoing

Application areas (historical) (cntd)

- ◆ Programmable networks
 - **NETKIT** project | application to Network Processors
 - **Virtual Routers** project (ongoing – with Laurent Mathy)
- ◆ Embedded systems
 - EU **Cortex** project
 - middleware for real-time control
 - Autonomic control of vehicles
 - 'sentient object' paradigm
 - EU **RUNES** project (more later...)

Application to grid computing

- ◆ Focus on future heterogeneous and pervasive Grid environments
- ◆ 'Heterogeneous' in two senses
 - heterogeneous infrastructures
 - *Networks*: clusters | LAN-based | WAN-based | infrastructure and ad-hoc wireless | sensor nets ...
 - *Nodes*: supercomputers | workstations | PDAs | sensors ...
 - heterogeneous "interaction types"
 - RPC and SOAP messaging | (un)reliable multicast | media-streaming | publish-subscribe | tuple spaces | workflow



Our first grid-related project

- ◆ The **Open Overlays** project (2004-7)
 - Collaborative with Oxford Brookes University
 - Application focus on wildfire management
 - Initial development of *GridKit* framework at Lancaster
 - Overlay-based approach to handling heterogeneous networks
 - An separate CF to handle heterogeneous interaction types
 - Development of application-level software (Oxford Brookes)
 - A distributed application that supports the collaboration of controllers and fieldworkers
 - A computationally-intensive Grid application to model fire spread



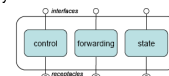
The Gridkit middleware

- ◆ Key features (as mentioned):
 - 'Overlays framework' for handling diverse network infrastructures
 - 'Interaction framework' for handling diverse interaction types

Web services API					
Interaction	Service discovery	Resource discovery	Resource mgmt	Resource monitoring	Security
Overlays framework					
OpenCOM v2 component model runtime					

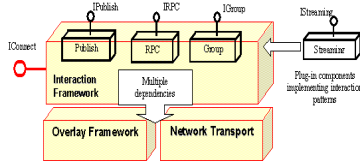
The Overlays Framework

- ◆ Hosts one or several overlay implementations (either independent or working together)
- ◆ Each overlay implementation must employ 3 per-node components
 - *Control* – maintains the topology of the overlay
 - *Forward* – implements the routing algorithm
 - *State* – maintains the overlay's state information
- ◆ Supports configurability
 - Common interfaces support 'plugging in' of overlays
 - Can stack overlays
- ◆ Supports dynamic reconfigurability
 - Can independently replace routing & control algorithms



The Interaction Framework

- ◆ Based on binding CF from OpenORB



Current state of Gridkit development (1 of 2)

- ◆ Open source project hosted at sourceforge
 - Development/ beta code available
 - First stable release was in April 2006
 - Developed as OpenCOM components in both Java and C++
 - <http://sourceforge.net/projects/gridkit>
- ◆ Available Overlay components (list is now out of date!)
 - Java
 - Tree Building Control Protocol
 - Chord Key-based Routing
 - PAST distributed hash table
 - SCRIBE (Multicast atop Chord KBR)
 - SCAMP (Scalable Gossip overlay)
 - Failure Monitoring Overlay
 - C++
 - Probabilistic Multicast

Current state of Gridkit development (2 of 2)

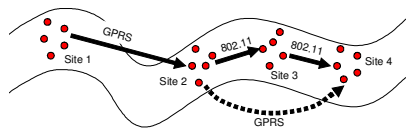
- ◆ Available Interaction Types
 - Remote Procedure Call
 - Java RMI, IIOP (C++), SOAP (C++)
 - Group communication (Java)
 - Publish-subscribe (Java)
 - Service discovery
 - Service Location Protocol v2 (Java)
- ◆ Further components under development
 - Resource management
 - Tuple-space
 - Media streaming
- ◆ Other (non-functional) cross-framework development work
 - Dependency components (middleware and overlays)
 - Security (middleware and overlays)

Sensor network-related projects

- ◆ Divergent Grid project
 - Collaborative with UCL, UK
 - Focus on MDS-based configuration of 'middleware families' to bridge diverse target domains (clusters | PDAs | sensor motes...)
- ◆ North West Grid project *
 - sensor network to monitor and predict flooding in a river valley
- ◆ RUNES project *
 - applying an OpenCOM-derived approach in "reconfigurable ubiquitous networked embedded systems"
 - 'fire in road tunnel' scenario

North West Grid project: Flood monitoring case study

- ◆ Scenario: wireless sensor network used to support flood management in a river valley in north west England
 - plus flood modelling in a back-end grid
- ◆ Motivation: to apply and exercise the OpenCOM/ Gridkit approach in a realistic 'pervasive grid'



The 'Gridstix' sensor nodes

- ◆ Based on Gumstix hardware
- ◆ Equipped with ...
 - pressure sensors, ultrasound flow sensors, video cameras | networks: Bluetooth, 802.11b, GPRS | battery plus solar panel | Linux
- ◆ Run Gridkit middleware (a very minimal configuration)
 - pub-sub and messaging interaction types
 - various types of overlays (mainly: fewest/ shortest hop spanning trees)
- ◆ Feed data into flood prediction models
 - Spatial: expensive | need to run on back-end grid
 - Point based: cheap | can run on low-power machines
- ◆ Supports local grid computation and network adaptivity ...

'Local grid computation'

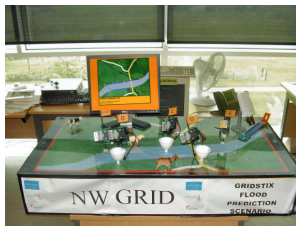
- ◆ Nodes organised into a 'lightweight grid' using an appropriate overlay
- ◆ Local 'point-based' flood prediction
 - used mainly to trigger adaptation (see next slide)
 - also used to alert local stakeholders
- ◆ Flow-rate calculation from video images of river surface
 - based on (parallel) analysis of camera images of flotsam on the river
 - video camera deployment is cheaper and easier than specialised ultrasound flow sensors
 - calculation *needs* to be done locally—too much data to send offsite
 - used to feed spatial prediction models, and also trigger adaptation

Examples of network adaptivity

- ◆ At physical network level
 - in quiescent conditions use Bluetooth
 - low power requirement | but correspondingly low reach
 - when flow-rate measurement indicates imminent flooding...; or when a critical node fails:
 - switch to 802.11b or GPRS | longer reach but more expensive
- ◆ At overlay level
 - in quiescent conditions use a *shortest hop* spanning tree overlay
 - low power requirement | but 'thin'
 - when point-based algorithm predicts local flood:
 - switch to *fewest hops* spanning tree (i.e. load and bind appropriate components) | higher power requirement but more resilient

Current status

- ◆ Currently deployed in Ribble valley | desktop demo also available



RUNES: Three middleware challenges arising from networked embedded systems

- ◆ Heterogeneity
 - from powerful PCs to tiny sensor devices
 - different hardware, OSs, and programming languages
 - multiple protocols and interaction paradigms
- ◆ Dynamicity
 - ever changing environments
 - devices fail, new devices join
 - new functionality to address dynamically-arising requirements
- ◆ Resource scarcity
 - energy, CPU speed, memory...


<i>Tmote/ Contiki/ C</i>	<i>CBlue/ RTOS/ C</i>	<i>Lippert/ Linux/ C</i>	<i>PC/ Win/ Java</i>
----------------------------------	-------------------------------	----------------------------------	------------------------------

State of the art in middleware for networked embedded systems

- ◆ Component-based systems for embedded devices
 - *Pebble, PECOS, Koala, ...*
 - programming language/hardware specific
 - aid in developing software; components are just for programming support
- ◆ Middleware and system support for wireless sensor networks
 - *MILAN, Directed Diffusion, TinyDB, ...*
 - target sensor applications only
 - difficult to reprogram in the face of changing application requirements
- ◆ Reconfigurable, component-based middleware systems for pervasive environments
 - *Gravity, P2PComp, DPRS, PCOM, ...*
 - target relatively resource-rich devices

Implications?


- ◆ No existing systems in the three identified categories of research meets all three of the identified challenges:
 - heterogeneity: nothing supports the full range of heterogeneous systems
 - dynamicity: nothing comprehensively supports dynamic reconfiguration
 - resource scarcity: nothing scales from resource-rich through resource-constrained devices
- ◆ The RUNES component model (a variant of OpenCOM v2) aims to address all three challenges

LANCASTER UNIVERSITY 


Three RUNES component model implementations

As we've seen

	Java	C/Linux	C/Contiki
Component Types	Classes	Shared obj (ELF .so)	Contiki Services
Components	Objects	C Structs	Loaded Service
Connectors	Java References / Interceptors	Function Pointers	Function Pointers
Receptacles	Objects	Function Pointers	Function Pointers
Interfaces	Java Interfaces	Function Pointers	Function Pointers


InfoLab21 
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk


LANCASTER UNIVERSITY 

Memory footprint evaluation

	Java	C/Linux	C/Contiki
Runtime Code	14.65 Kb	16 Kb	780 bytes
Runtime Data	840 bytes	4 Kb	52 bytes
Null Component mem	544 bytes	24 bytes	9 bytes
Per-Interface mem	200 bytes	40 bytes	2 bytes
Per-Receptacle mem	264 bytes	22 bytes	2 bytes

InfoLab21 
Computing Department


InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk

LANCASTER UNIVERSITY 


Memory footprint evaluation (cntd)

- Memory footprint stats for some example components from the RUNES 'fire in road tunnel' demo on the C/Contiki platform...
- Full deployment on a Tmote Sky uses 3,750 bytes

	Data Acquisition	Data Dissemination	Publish/Subscribe
Lines of Code	287 lines	181 lines	197 lines
Memory Footprint	1462 bytes	738 bytes	772 bytes

InfoLab21 
Computing Department


InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk

LANCASTER UNIVERSITY 


Expected learning outcomes

- You have an appreciation of a range of possible areas of application of adaptive middleware
- You have an idea of how adaptive middleware can be designed and deployed in such areas

Finis – et merci pour votre attention! Questions?


InfoLab21 
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk

LANCASTER UNIVERSITY 

Selected Publications

- OpenORB
 - Coulson, G., Blais, G.S., Clark, M., Pavlovic, N., 'The Design of a Configurable and Reconfigurable Middleware Platform', ACM Distributed Computing Journal, Vol 16, No 2, pp 109-126, April 02
- Open COM v2
 - Coulson, G., Blais, G.S., Green, P., Jodda, A., Lee, K., Ueyama, J., Srikaran, T., 'A Generic Component Model for Building Systems Software', to appear ACM Transactions on Computer Systems, February 2008.
- GridKit / Open Overlays
 - Green, P., Coulson, G., Blais, G.S., Porter, B., 'Deep Middleware for the Divergent Grid', Proc. IFIP/ACM/USENIX Middleware 2005, Grenoble, Nov 05
 - Green, P., Hopton, D.P., Porter, B., Blais, G.S., Coulson, G., Taheri, F., 'Experiences with Open Overlay: A Middleware Approach to Network Heterogeneity', to appear Proc. EuroSys 06, 2006.
- Sensor network middleware
 - Hughes, D., Greenwood, P., Blais, G.S., Coulson, G., Green, P., Papanicolaou, F., Smith, P., Boven, K., 'An Experiment with Reflexive Middleware to Support Grid-based Flood Monitoring', to appear Concurrency and Computer Practice and Experience, 2006.
- RUNES
 - Costa, P., Coulson, G., Gold, R., Lud, M., Mascolo, C., Motola, L., Picco, G.P., Sivaevska, T., Wessinghe, N., Zachariak, S., 'The RUNES Middleware for Networked Embedded Systems and its Application to a Disaster Management Scenario', Proc. 5th Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'07), White Plains, New York, 19-23 March 2007.

InfoLab21 
Computing Department

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk