Replication of Objects
using the Object Group Service

Huib van den Brink

Center for Software Technology, Utrecht University
http://www.cs.uu.nl/groups/ST/

October 26, 2005
Outline

1. Limitations of distributed programming
   - Problems involved
   - Solutions and backgrounds

2. Object Group Service
   - Details, design and decisions

3. Java implementation
   - OGS Implementation
   - Example

4. Finally
   - Pros & Cons
   - Questions
   - Exercise
Distributed programming

CORBA provides means to become:

1. Heterogeneous (CORBA provides the IDL to abstract)
2. Interoperable (due to wrapper objects and ORB gateways)
3. Extensible (due to highly modular applications)
Distributed programming

CORBA provides means to become:

1. Heterogeneous (CORBA provides the IDL to abstract)
2. Interoperable (due to wrapper objects and ORB gateways)
3. Extensible (due to highly modular applications)

But CORBA doesn’t provide

1. Availability
2. Reliability

The Object Management Group chose not to (would require entailed protocol standardization)
Structural solution

Object replication

1. Dynamic replication
2. Active replication
3. Passive replication
Structural solution

Object replication
1. Dynamic replication
2. Active replication
3. Passive replication

Transparency levels
1. Plurality transparency
   (unaware of replication; illusion of singleton objects)
2. Behaviour transparency
   (unaware of policies and protocols controlling the behaviour)
3. Type transparency
   (typed method invocations)
Short corba overview

CORBA components

1. ORB (Heterogeneous method invocation)
2. Common Facilities (end-user-oriented capabilities)
3. Services (Naming, Events, Transactions, Life Cycle)
4. Domain Interfaces (vertical application domains)
Short corba overview (2)

CORBA components

- Interface Repository
- IDL Compiler
- Implementation Repository
- Client
- IDL Stubs
- ORB Interface
- Object (Servant)
- DII
- ORB Core
- GIOP/P2P
- Standard Interface
- Standard Language Mapping
- ORB-Specific Interface
- Standard Protocol

http://www.cs.uu.nl/wiki/No
## Ad hoc solutions

### Integration approach (Proprietary system)

1. Implemented the group handling within the ORB (using a group toolkit)
2. Ease of development
3. Transparent
4. Not CORBA compliant (not portable and interoperable)
Ad hoc solutions

Integration approach (Proprietary system)
1. Implemented the group handling within the ORB (using a group toolkit)
2. Ease of development
3. Transparent
4. Not CORBA compliant (not portable and interoperable)

Interception approach
1. Intercepts requests that have the IIOP protocol (keeping the ORB unaware)
2. Low level interception mechanism on both client and server
3. Processed then by a group communication toolkit
4. Embraces OS specific mechanisms (limits portability)
Service approach

Group Service

1. Component-oriented approach
2. Placed at the 'Services' component within the OMA (Object Management Architecture)
3. Portable (it exists of IDL-specified interfaces)
4. Provides sophisticated Object Replication
5. Supports transparent group invocations
Service approach

Design and components involved

1. Object Group
2. Object Group Reference
3. Object Implementation
4. Communication Subsystem
5. Client application
Service approach

Design and components involved
1. Object Group
2. Object Group Reference
3. Object Implementation
4. Communication Subsystem
5. Client application

The OGS core
1. Messaging service
2. Monitoring service
3. Consensus service
4. Group service (group multicast & group membership)
Service approach

The Group Service situated

The OGS can either be a daemon as well as a lib
Scenario: invoking a method / sending a request (1)
Scenario: invoking a method / sending a request (2)

Multicast

1. Agreement (like ordering guarantees)
2. Validity (send them to the right people)
3. Integrity (no spurious messages accepted)
Scenario: joining / leaving a Object Group (1)

1. View management: view change protocol
2. State transfer protocol
Scenario: joining / leaving a Object Group (2)

1. View management: view change protocol
2. State transfer protocol

Diagram:
- 1. JOIN-REQUEST
- 2. FLUSH-REQUEST
- 3. FLUSH-ACK
- 4. STATE_MSG
- 5. VIEW
Interaction overview

1. Client interface (GroupAccessor)
2. Member interface (GroupAdministrator)
3. Service interface (Groupable)
Decisions involved (1)

**Naming service**

1. Group names system wide unique
2. Every member has references to all the other members
3. Long-lived object references
4. No polling to the naming service

**Reliable multicasts and their replies**

1. All replies
2. Majority of replies
3. One reply
4. Zero reply
5. One way multicast
Decisions involved (2)

**Typed invocations**

1. Dynamic Skeleton Interface
2. Dynamic Invocation Interface

**Virtual Synchronization**

1. Server object negotiates and publishes its needs
2. Total ordering when necessary
3. But loose ordering for performance when not
Failure detection

1. Replication
2. State checkpoint
3. Monitoring and availability management

Piranha

1. Automatic restart
2. Notify service (passing panicing messages)
3. Takes 1 second to detect crashed Object
4. Takes about 30 seconds to detect failed machine
5. Asynchronous implies not knowing difference between failure and slow processing

Based on Electra, Integration Approach, so not portable
# OGS using Java

**OGS Java Implementation**

1. Patrick Eugster
2. Java 1.1.5
3. VisiBroker 3.2
4. CORBA 2.0
5. 1999
// IDL
#include "GroupAdmin.idl"

interface Counter : mGroupAdmin::Groupable {
    readonly attribute long count;
    void inc();
    void reset();
};
public static void main(String[] args) {

    // Obtain initial references to the ORB
    ORB orb = ORB.init(args, null);

    // Bind to group accessor factory
    GroupAccessorFactory gaf = ...;

    // Bind the interface repository
    Repository ir = RepositoryHelper.narrow(
        orb.resolve_initial_references("InterfaceRepository"));

    // Get a reference to the interface definition of the server
    InterfaceDef in_def = InterfaceDefHelper.narrow(ir.lookup("Counter"));

    // Create a typed group accessor
    Counter cnt = CounterHelper.narrow(ga.cast(in_def));

    // Invoke replicated server
    cnt.reset();
    System.out.println("Counter is " + cnt.count());
    cnt.inc();
    System.out.println("Counter is " + cnt.count());

    // Release group accessor
    gaf.release(ga);
Server (1)

```java
CORBA::Any* Counter_i::deliver(const CORBA::Any& message) {
    return new CORBA::Any();
}

void Counter_i::view_change(const mGroupAccess::GroupView& newView) {}

CORBA::Any* Counter_i::get_state() {
    // Pack the state into an any
    CORBA::Any* a = new CORBA::Any();
    *a <<= count_;
    return a;
}

void Counter_i::set_state(const CORBA::Any& a) {
    // Extract the state from an any
    a >>= count_;
}

CORBA::Long Counter_i::count() { return count_; }
void Counter_i::inc() { count_++;
void Counter_i::reset() { count_ = 0; }
```
mGroupAdmin::OperationSemanticsSeq* Counter_i::operation_semantics()
{
    // Return the semantics associated to each operation
    mGroupAdmin::OperationSemanticsSeq* oss = new OperationSemanticsSeq(3);
    oss->length(3);
    (*oss)[0].name_ = "get_count";
    (*oss)[0].ordering_ = mGroupAccess::UNRELIABLE;
    (*oss)[1].name_ = "inc";
    (*oss)[1].ordering_ = mGroupAccess::ATOMIC;
    (*oss)[2].name_ = "reset";
    (*oss)[2].ordering_ = mGroupAccess::ATOMIC;
    return oss;
}
```java
int main(int argc, char *argv[])
{
    // Obtain initial references to the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::BOA_var boa = orb->BOA_init(argc, argv);

    // Create server application objects
    Counter_var cnt = new Counter_i();

    // Register object implementation
    boa()->obj_is_ready(cnt);

    // Bind to group administrator factory
    mGroupAdmin::GroupAdministratorFactory_var gaf = ...;

    // Create a group administrator
    mGroupAdmin::GroupAdministrator_var ga =
        gaf->create("ogs://counters.epfl.ch/clock");

    // Join the group
    ga->join_group(cnt);

    // Wait for messages
    boa->impl_is_ready();
    return 0;
}
```
Related patterns (1)

Proxy pattern

1. Regulates and controls the access to objects
2. Surrogate placeholder for remote objects
3. Likewise the Object Group Reference
Coordinator/Cohort pattern

1. Provides redundant computation
2. Only coordinator performing the computation
3. Passive replication (hot-standbys)
4. One coordinator and several cohorts
Related patterns (3)

Event Channel pattern
1. Abstraction of a message bus
2. Highly available
3. Uses subscribers and objects posting events
4. May post events to systems currently down (request decoupling)
5. Failed receivers can backlog of requests after restart

Main differences
1. Member may selectively listen to events
2. Spooling of requests
3. Doesn’t exclude failed members from the group
The Pros and Cons of OGS

**Benefits**

1. Means to load-balance
2. Means to effectively divide resources (like database parts) (think of Google)
3. Fault tolerance

**Drawbacks**

1. Registration overhead
2. Total order is slowed down by the slowest host of the group
3. Not total transparent
Questions?
Consider Google (highly available and division of data to search per machine). What configuration of Object Replication would you use and why? (think of virtual synchronization, multicasts, (un)typed invocations, naming service, Service as deamon or lib, failure detection and so on)