



STACKFRAME

Distributed Objects

Gene McCulley

Distributed Objects

- What are distributed objects?
- Why would you want to use them?
- How do they work?
- How OneSAF uses them

What are Distributed Objects?

- Objects that are not tied to a single machine/JVM
- Can be referenced and manipulated over the network
- Useful for implementing distributed systems

Why Distributed Objects?

- Scalable systems
- Heterogeneous architectures
- Reliable systems

How do they work?

- Lots of different kinds of implementations
 - RMI
 - CORBA
 - DCOM
 - SOAP

RMI

- Remote Method Invocation
- Java specific abstraction
 - Depends on serialization
 - Distributed Garbage Collection
 - Can make use of distributed class definitions
 - Easy to use from Java

CORBA

- Common Object Request Broker Architecture
- Language and platform independent
- Standardized
- Many implementations
- Because no serialization, more data modeling work required upfront

SOAP

- Simple Object Access Protocol
- Language and platform independent
- Standardized
- Web/firewall friendly
- Many implementations
- Because serialization uses XML, some data modeling work required upfront



Distributed Simulation

- Simulation with more than one computer involved
- Because the simulation is distributed, lots of optimizations impossible
- Nodes in simulation can be manned modules or CGF systems
- Not necessarily real-time or HITL

OneSAF

- One Semi-Automated Forces
- Simulation system being built by the U.S. Army
- Just completed Government Acceptance Test
- 1.0 release September 30th, 2006
- 5 years in development

OneSAF SLOC

Java	C	C++	Shell
1,340,855	45,814	332,392	10,702

Big distributed exercise

- 60 nodes
 - 20 simcores (no GUI)
 - 40 workstations (human operator)
 - Dual core 2.4GHz P4, 2GB RAM
 - Mostly Linux, some Windows
- 6000 entities (tank, soldier, helicopter, etc.)
- 210,000 objects

Requirements

- Reliability
 - Low fail rate
 - Quick recovery when late joining a failed node
- Checkpoint/Restore (fast serialization)
- Long-term persistence (XML serialization)
- Data collection
- Performance (low latency)



Messaging Services

- Layer upon which distributed objects are built in OneSAF
- Zero-configuration (service/node discovery)
- UDP multicast and TCP
- Reliable multicast implementation

Object Database (ODB)

- Objects
 - Bundles of (name/value pairs)
 - Can persist beyond lifetime of creating node
 - Can reference each other
- Interactions (Events)
 - Can reference objects

Object Examples

- GroundVehicle
- Unit
- Mission
- Affiliation

Interaction Examples

- Fire
- Detonation
- Resupply
- Repair

Data Model

- Define object and interaction types as Java interface classes
- Use JavaBeans standard of getter and setter methods
- All types referenced must be Serializable, Cloneable, Encodable (XML)
- Object types can be periodic and transient

Periodic Objects

- For some object types, reliability of change events is less important
- Constantly being updated
- No point in retrying to get stale data

Transient Objects

- By default, objects persist across node failures
- Some objects associated with a node
- Marked as transient
- Objects removed from database when creating node dies
- Used to implement object migration

Proxy Objects

- ODB uses `java.lang.reflect.Proxy` to implement network transparency
- Each object gets a Proxy object implementing interface
- Calls to `get/set` methods act upon database

Lifecycle of an object

- Interface defined
- `odb.defineObject()` called with interface
 - Checks validity/defines schema
- `odb.create()` called with interface
 - object built and broadcasted to other nodes
 - Proxy built and returned to client

Client view of ODB

- `objectAdded/objectRemoved` subEvents of `ObjectEvent` sent when an object is added to/removed from database
- `java.beans.PropertyChangeEvent` sent to `java.beans.PropertyChangeListener` when an attribute is changed on an object
- Database itself implements `java.util.Collection`



Object Example

```
public interface Vehicle {  
    Point3d getLocation();  
    void setLocation(Point3d location);  
  
    public Unit getUnit();  
    void setUnit(Unit unit);  
}
```

Object Example (use)

```
odb.defineObject(Vehicle.class);
```

```
Vehicle vehicle = (Vehicle)odb.create  
(Vehicle.class);
```

```
vehicle.setLocation(location);
```

```
odb.set(vehicle, "location", location);
```

Questions?