Object-Oriented Distributed Technology

- Objects
- Objects in Distributed Systems
- Requirements of Multi-User Applications

Reading:
- Couloris: Distributed Systems, Chapter 5

Object-Oriented Languages

- Object Identity
  - "object identifiers" (OIDs)
  - OIDs as first class values
- Actions
  - Initiated by sending message to object requesting method invocation
  - State in object may change
  - cascaded invocations of methods
- Dynamic Binding
  - The method executed is chosen according to the class of the recipient of the message.
- Garbage Collection
  - Dynamically allocated instances may be explicitly deleted or space is freed implicitly by garbage collector.
  - GC in distributed systems?
Objects in Distributed Systems

- Object Identity in a Distributed System
  - Remote object identifiers (ROIDs)
  - Ex. Java: ROI = endpoint (Java vm) + identifier (ObjID)
  - ROIDs as first-class values
  - Service for comparing remote object identifiers
    - e.g. Java: RemoteObject::equals()

- Actions in a Distributed Object System
  - Remote Method Invocation

- The Role of Proxies for Transparent RMI
  - Local proxy for each remote object that can be invoked by local object.
  - Local proxy behaves like local object, but, instead of executing message, forwards it to the remote object. (client stubs)
  - Remote object has skeleton object with server stub procedures

Proxies and Skeletons

Diagram showing the interaction between proxy and skeleton objects in a distributed system.
Proxies and Skeletons (cont)

- **Proxies:**
  - Need proxies to invoke remote objects.
  - Proxies are created when needed whenever ROID arrives in Reply message.
  - **ROID module** manages proxies and ROIDs.

- **Dispatchers and Skeletons:**
  - Not necessary for systems with **reflection** capabilities.
  - e.g. class **Method** in Java 1.2 reflection package:
    - method **invoke** can be called on instance of **Method**.
    - Dispatcher now **generic** and skeleton **unnecessary**.

Arguments and Results in RMI

- Semantics of passing arguments for RMI in object-oriented languages needs to be defined. Why?
- **Argument and Result passing in Java RMI:**
  - When type of parameter is defined as remote interface, argument or result is passed as ROID.
  - Other non-remote objects may be passed **by value** if they are **serializable**.
- **Which objects can be accessed by RMI?**
  - Any object can be accessed by RMI
  - Distinguish between remote objects and local objects. (e.g. keywords or classes with interface compiler)
  - Use **interface definition language** (IDL)
- **Problem:** migration/replication
Dynamic Binding

- Dynamic method binding should also apply to RMI.

- Smalltalk: Allow any message to be sent to any object, and raise exception if method is not supported.
  - Distributed Smalltalk: general-purpose proxies.

- Java RMI:
  - dynamic binding as a natural extension of local case
  - Example:
    ```java
    Shape aShape = (Shape) stack.pop();
    float f = aShape.perimeter();
    ```

Garbage Collection

- Some languages (Java, Smalltalk) support garbage collection.
- Explicit memory management difficult/impossible in distributed environment.
- Distributed garbage collection typically realized in ROID modules. Each ROID module:
  - keeps track how many sites hold remote ROIDs for each local object
    (maintains holders table)
  - informs other ROID modules about generation/deletion of ROIDs for their local objects (through the use of addRef() and removeRef())
- Local garbage collector collects objects with no local or remote references.
- Reference counting (addROID()/removeROID()) over unreliable networks?