Locating Distributed Objects

Motivation

- Avoid using physical locations for locating components!
- Naming:
  - Locating components by external names
  - Similar to white pages
- Trading:
  - Locating components by service characteristics
  - Similar to yellow pages

Outline

- Object Naming
  - Principles
  - CORBA Naming Service
  - COM Monikers
  - Java/RMI Registry
- Object Trading
  - Principles
  - CORBA Trading Service
Naming

Common Principles

- Object-oriented middleware uses object references to address server objects
- We need to find a way to get hold of these object references without assuming physical locations
- A name is a sequence of character strings that can be bound to an object reference
- A name binding can be resolved to obtain the object reference

Common Principles

- There may be many server objects in a distributed object system
- Server objects may have several names
- Leads to large number of name bindings
- Name space has to be arranged in a hierarchy to avoid
  - Name clashes
  - Poor performance when binding/resolving names
- Hierarchy achieved by naming contexts
Common Principles: Naming Contexts

Common Principles: Composite Names

- Names are composed of possibly more than one component
- Used to describe traversals across several naming contexts
- Example:
  - ("UEFA","England","Premier","Chelsea").

Common Principles: Name Server Behaviour

- Name bindings are managed by name servers
- Not every server object needs a name
- Server objects may have several names (aliases)
- Name servers must store bindings persistently
- Name servers should be tuned towards efficiently resolving name bindings
- Name server may itself be distributed
**CORBA Naming Service**

- Supports bindings of names to CORBA object references.
- Names are scoped in naming contexts.
- Multiple names can be defined for object references.
- Not all object references need names.

**CORBA Names**

- Names are composed of simple names.
- Simple names are value-kind pairs.
- Value attribute is used for resolving names.
- Kind attribute is used to provide information about the role of the object.

**IDL Types for Names**

```idl
module CosNaming {
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence <NameComponent> Name;
    ...
};
```
The IDL Interfaces

- Naming Service is specified by two IDL interfaces:
  - NamingContext defines operations to bind objects to names and resolve name bindings.
  - BindingIterator defines operations to iterate over a set of names defined in a naming context.

Excerpt of NamingContext Interface

```idl
interface NamingContext {
    void bind(in Name n, in Object obj)
        raises (NotFound, ...);
    Object resolve(in Name n)
        raises (NotFound, CannotProceed,...);
    void unbind (in Name n)
        raises (NotFound, CannotProceed...);
    NamingContext new_context();
    NamingContext bind_new_context(in Name n)
        raises (NotFound, ...);
    void list(in unsigned long how_many,
              out BindingList bl,
              out BindingIterator bi);
}
```

Excerpt of BindingIterator Interface

```idl
interface BindingIterator {
    boolean next_one(out Binding b);
    boolean next_n(in unsigned long how_many,
                    out BindingList bl);
    void destroy();
}
```
Naming Scenario: Binding

Promote Bielefeld to German '1. Liga' and relegate Frankfurt

```c
1L=resolve("UEFA","Germany","1. Liga")
bind("Arm. Bielefeld", bielefeld)
unbind("Eintr. Frankfurt")
```

Bootstrapping Naming Service

- **How to get the Root Naming Context?**
- **ORB Interface provides for initialization:**

```c
module CORBA {
    interface ORB {
        typedef string ObjectId;
        typedef sequence <ObjectId> ObjectIdList;
        exception InvalidName{
        ObjectIdList list_initial_services();
        Object resolve_initial_references
            in ObjectId identifier) raises(InvalidName);
    }
}
```

Naming Scenario: Resolving

Print squad of Borussia Dortmund

```c
doresolve("UEFA","Germany")
doresolve("1. Liga","BVB")
print()
```
Naming Scenario: Iterating

- Print all team names of German 1. Liga

```
ctx = Team(0, bl, bi)

while (true):
    t = next_one().value
    print(t.name())
```

Naming in COM: Monikers

- **Oxford English Dictionary:**
  - *moniker* (‘monikar’). slang. A name, a nickname.

- In COM used for ‘de-coupling clients from the algorithms and the information that is needed for finding server objects’ [Box98]

- Supports binding and resolving of names

- Name space can be hierarchically structured

Naming in COM: IMoniker Interface

```c
interface IMoniker : IPersistStream {
    HRESULT BindToObject([in] IBindCtx *pbc,
                         [in, unique] IMoniker *pmkToLeft,
                         [in] REFIID riid,
                         [out, iid_is(riid)] void **ppv);
    ...`
```
Creation of Monitors: IParseDisplayName

- To be nameable, server objects implement IParseDisplayName interface
- Creates a monitor object by parsing an external (textual) name that is called display name

```c
interface IParseDisplayName {
    HRESULT MkParseDisplayName([in] IBindCtx *pbc,
                               [in,string] const OLECHAR *pwszName,
                               [out] ULONG *ppchEaten,
                               [out] IMoniker **ppmk);
}
```

Assessment

- COM has both internal names (monikers) and external names (display names), which complicates naming
- COM Naming is intertwined with other parts of the COM specification (containers)
- COM Naming is not transparent to server object designers - they have to implement IParseDisplayName

Naming in RMI: The RMI Registry

- Simplified version of CORBA Naming
- No composite names
- Security Restriction: Name bindings cannot be created from remote hosts
- There has to be a registry on each host
- Different registries have to be integrated into a federated name space
Naming in RMI: The RMI Registry

```java
package java.rmi.registry;
public interface Registry extends java.rmi.Remote {
    public static final int REGISTRY_PORT = 1099;
    public java.rmi.Remote lookup(String name)
        throws java.rmi.RemoteException,
            java.rmi.NotBoundException,
            java.rmi.AccessException;
    public void bind(String name, java.rmi.Remote obj)
        throws java.rmi.RemoteException,
            java.rmi.AlreadyBoundException,
            java.rmi.AccessException;
    public void rebind(String name, java.rmi.Remote obj)
        throws java.rmi.RemoteException,
            java.rmi.AccessException;
    public void unbind(String name)
        throws java.rmi.RemoteException,
            java.rmi.NotBoundException,
            java.rmi.AccessException;
    public String[] list()
        throws java.rmi.RemoteException,
            java.rmi.AccessException;
}
```

Using the RMI Registry

```
c: Client
  L: Locate
  E: Enumerate

root: Registry

E = lookup("UEFA")
root = getRegistry("ns.fifa.org")

E:
  1L:
    BVB:
      D = lookup("Germany")
      1L = lookup("1. Liga")
      BVB = lookup("BVB")
```

Assessment

- Lack of hierarchical names increases number of remote operations needed to resolve a name binding
- Finding the root registry is not necessarily location transparent
- Security restriction breaks name location transparency
Limitations of Naming

- Limitation of Naming in all approaches: Client always has to identify the server by name.
- Inappropriate if client just wants to use a service at a certain quality but does not know from who:
  - Automatic cinema ticketing,
  - Video on demand,
  - Electronic commerce.

Trading

Motivation

- Locating objects in location transparent way
- Naming simple but may not be suitable when
  - clients do not know server
  - there are multiple servers to choose from
- Trading supports locating servers based on service functionality and quality
- Naming ↔ White pages
- Trading ↔ Yellow Pages
Trading Characteristics

- Trader operates as broker between client and server.
- Enables client to change perspective from 'who?' to 'what?'
- Similar ideas in:
  - mortgage broker
  - insurance broker
  - stock brokerage

Trading Characteristics

- Common language between client and server:
  - Service types
  - Qualities of service
- Server registers service with trader.
- Server defines assured quality of service:
  - Static QoS definition
  - Dynamic QoS definition.

Trading Characteristics

- Clients ask trader for
  - a service of a certain type
  - at a certain level of quality
- Trader supports
  - service matching
  - service shopping
**Example**

- **Hongkong Telecom video-on-demand server:**
  - Server
    - MGM
    - Warner
    - Independent
  - Video-on-demand provider
  - User

**The Trading Process**

- **Example: Video-on-demand server**
  - Client
  - Trader
  - MGM:VoDS
  - Warner:VoDS
  - query()
  -export()
  - download()
  - modify()

**Service Type Definition**

- **Service types define**
  - Functionality provided by a service and
  - Qualities of Service (QoS) provision.
- **Functionality defined by object type**
- **QoS defined based on properties, i.e.**
  - property name
  - property type
  - property value
  - property mode
    - mandatory/optional
    - readonly/modifiable
Service Type Example

typedef enum {VGA, SVGA, XGA} Resolution;

service video_on_demand {
  interface VideoServer;
  readonly mandatory property float fee;
  readonly mandatory property Resolution res;
  modifiable optional property float bandwidth;
}

Service Type Hierarchy

- An object type might have several implementations with different QoS.
- Same object type might be used in different service types.
- Service type S is subtype of service S' iff
  - object type of S is identical or subtype of object type of S'
  - S has at least all properties defined for S'
- Subtype relationship can be exploited by trader for service matching purposes

Constraint Definition

- Importer defines the desired qualities of service as part of the query:
- Example:
  fee<10 AND res >=SGA AND bandwidth>=256
- In a query, trader matches only those offers that meet the constraint
Trading Policies

- Depending on constraint and available services, a large set of offers might be returned by a query.
- Trading policies are used to restrict the size of the matched offers:
  - Specification of an upper limit
  - Restriction on service replacements
  - Restriction on modifiable properties (these might change between match making and service requests)

Federated Traders

- Scalability demands federation of traders
- A trader participating in a federation:
  - Offers the services it knows about to other traders
  - Forwards queries it cannot satisfy to other traders
- Problems:
  - Non-termination of import
  - Duplication of matched offers

Trading Graph

[Diagram showing a trading graph with nodes labeled T1, T2, T3, and T4, with various attributes such as query.hop_count and max_hop_count.]
**Defining Quality of Service**

typedef Istring PropertyName;
typedef sequence<PropertyName> PropertyNameSeq;
typedef any PropertyValue;
struct Property {
    PropertyName name;
    PropertyValue value;
};
typedef sequence<Property> PropertySeq;
enum HowManyProps {none, some, all}
union SpecifiedProps switch (HowManyProps) {
    case some : PropertyNameSeq prop_names;
};
**Trader Interface for Exporters**

```java
interface Register {
    OfferId export
        (in Object reference,
         in ServiceTypeName type,
         in PropertySeq properties)
        raises(...);
    OfferId withdraw
        (in OfferId id)
        raises(...);
    void modify
        (in OfferId id,
         in PropertyNameSeq del_list,
         in PropertySeq modify_list)
        raises(...);
}
```

**Trader Interface for Importers**

```java
interface Lookup {
    void query
        (in ServiceTypeName type,
         in Constraint const,
         in Preference pref,
         in PolicySeq policies,
         in SpecifiedProps desired_props,
         in unsigned long how_many,
         out OfferSeq offers,
         out OfferIterator offer_itr,
         out PolicyNameSeq Limits_applied)
        raises(...);
}
```

**Key Points**

- Distributed objects can be located by naming and trading
- Naming binds externally known names to an object reference and supports name resolution to reveal the object reference
- Trading supports locating objects based on the functionality that they offer and the quality of service that they guarantee