Architecture of the CORBA Component Model

POA Interfaces

POA Features

- Provide unique programming interface for servant development across implementations and languages
- Provide support for transparent activation of objects
- Allow a single servant to support multiple object identities simultaneously
- Allow multiple distinct instances of the POA to exist in one server
- Provide support for transient objects with minimal programming effort and overhead
- Provide support for implicit activation of servants with POA-allocated object ids
- Allow object implementations to be maximally responsible for an objects behaviour.
- Provide an extensible mechanism for associating policy information with objects implemented in a POA.
- Allow programmers to construct object implementations that inherit from static skeleton classes, generated by IDL compilers, or a DSI implementation

POA Creation

- Standard policies
  - Thread policy: ORB_CTRL_MODEL
  - Lifespan policy: TRANSIENT
  - Object Id Uniqueness Policy: UNIQUE_ID
  - Id Assignment Policy: SYSTEM_ID
  - Servant Retention Policy: RETAIN
  - Request Processing Policy: USE_ACTIVE_OBJECT_MAP_ONLY
- Used for RootPOA, and as the default for new POAs

Reference Creation

- Create reference not associated with a servant:
  - create_reference, create_reference_with_id
- Associate a servant with an object reference
  - activate_object, activate_object_with_id
  - Use id_to_reference, servant_to_reference to obtain object reference
- Perform implicit activation
  - according to language mapping
  - Use servant_to_reference

Object Activation

- Reference may be associated with a servant (active) or not (inactive)
- RETAIN policy: activated objects are added to active object map
  - Objects get explicitly activated through activate_object[_with_id]
  - Objects get automatically activated through servant manager added by set_servant_manager
  - USE_DEFAULT_SERVANT policy: Objects get automatically associated with the default servant
- NON_RETAIN
  - Objects active only during the request
  - Activation occurs through the servant manager, or with the default servant
- If no object can be activated for a request: OBJECT_NOT_EXIST
  - If there should be a servant manager but is none: OBJ_ADAPTER

Implicit Activation

- IMPLICIT_ACTIVATION policy requires SYSTEM_ID and RETAIN policies
- Interface of servant is determined from skeleton, or _primary_interface of DynamicImplementation
- Implicit activation happens through servant_to_reference, servant_to_id, or _this (C++, Java)
- UNIQUE_ID: only inactive servants are activated
  - Otherwise, the active object is returned
- MULTIPLE_ID: implicit activation always creates a new reference
  - Language-mapping specific: _this returns „current" object if invoked in the context of an operation implementation

Multi-Threading

- **Explicit main loop: ORB operations**
  - work_pending, perform_work, run, shutdown
- **Threading models:**
  - Single-threaded: POA is thread-unaware
  - ORB-controlled: ORB creates and terminates threads at will
  - Main thread: All POAs with that policy have their events processed in the same (main) thread

- All interfaces are defined in PortableServer
  - CORBA 2.6: All interfaces are local
- POA
- POAManager
- ServantManager
- ServantActivator
- ServantLocator
- AdapterActivator
- Current

- Policy interfaces:
  - ThreadPolicy
  - LifespanPolicy
  - IdUniquenessPolicy
  - IdAssignmentPolicy
  - ImplicitActivationPolicy
  - ServantRetentionPolicy
  - RequestProcessingPolicy
- PortableServer::Servant is a native type

POAManager

```
local interface POAManager {
    exception AdapterInactive{};
    enum State {HOLDING, ACTIVE, DISCARDING, INACTIVE};
    void activate()
                    raises(AdapterInactive);
    void hold_requests(
            in boolean wait_for_completion)
                    raises(AdapterInactive);
    void discard_requests(
            in boolean wait_for_completion)
                    raises(AdapterInactive);
    void deactivate(
            in boolean etherealize_objects,
            in boolean wait_for_completion)
                    raises(AdapterInactive);
    State get_state();
};
```

AdapterActivator

- Implemented by application
- Used to activate unknown adapters
- Associated with POAs

```
local interface AdapterActivator {
  boolean unknown_adapter(
      in POA parent,
      in string name);
};
```

Servant Managers

- Implemented by application
- Associated with POAs of appropriate policy
- Activate objects on demand
- Managers can raise **`ForwardRequest`** exception
- Two kinds
  - Activators: activate objects which get put into AOM
    - Used with RETAIN
    - Typically dispose etherealized servants
  - Locators: activate objects for the period of a single call
    - Used with NON_RETAIN
    - Typically cache servants across multiple invocations
- Base interface: ServantManager

```
local interface ServantManager{ };
```

Servant Activators

```
local interface ServantActivator : ServantManager {
   Servant incarnate (
       in ObjectId oid,
       in POA adapter)
               raises (ForwardRequest);
   void etherealize (
       in ObjectId oid,
       in POA adapter,
       in Servant serv,
       in boolean cleanup_in_progress,
       in boolean remaining_activations);
};
```

Servant Activators (2)

- Invocations to incarnate and etherealize are serialized and mutually exclusive

- Incarnations cannot overlap

- Etherealization may take time until all requests complete

- Invoking new request on an object that is being etherealizeds:
    - Requests are blocked or rejected

Servant Locators

```
local interface ServantLocator : ServantManager {
   native Cookie;
   Servant preinvoke(
       in ObjectId oid,
       in POA adapter,
       in CORBA::Identifier operation,
       out Cookie the_cookie)
               raises (ForwardRequest);
   void postinvoke(
       in ObjectId oid,
       in POA adapter,
       in CORBA::Identifier operation,
       in Cookie the_cookie,
       in Servant the_servant);
};
```

Servant Locators (2)

- One pair of preinvoke/postinvoke per request
- No serialization:
    - Locator can use Cookie to match preinvoke and postinvoke

POA Policies

- Policy objects: represent configuration information
- Policy type, policy value
  - Generic ORB operation to create policy objects
  - POA-specific operations to create POA policies
- Example: Thread policies

```
const CORBA::PolicyType THREAD_POLICY_ID = 16;
enum ThreadPolicyValue {
    ORB_CTRL_MODEL, SINGLE_THREAD_MODEL, MAIN_THREAD_MODEL
};
local interface ThreadPolicy : CORBA::Policy {
    readonly attribute ThreadPolicyValue value;
};
interface POA { // ...
    ThreadPolicy create_thread_policy(
            in ThreadPolicyValue value);
}
```

Lifespan Policy

- TRANSIENT: Objects cannot outlive the POA
  - Requests received after POAManager is deactivated receive OBJECT_NOT_EXIST

- PERSISTENT: Objects exist independent from POA
  - Typically combined with USER_ID policy, and perhaps servant manager
  - For SYSTEM_ID POAs, proprietary mechanisms might be used

IdUniquenessPolicy

- UNIQUE_ID: active servants support only one object id
- MULTIPLE_ID: a servant may be associated with more than one object id
  - Meaningless in combination with NON_RETAIN

IdAssignmentPolicy

- USER_ID: object Ids created by application
- SYSTEM_ID: object Ids created by POA

ServantRetentionPolicy

- RETAIN: activated servants are put into AOM
- NON_RETAIN: objects are etherealized at the end of the request.
  - Requires either USE_DEFAULT_SERVANT or USE_SERVANT_MANAGER

RequestProcessingPolicy

- USE_ACTIVE_OBJECT_MAP_ONLY: objects not found in the AOM don't exist
- USE_DEFAULT_SERVANT: Objects not found in the AOM are associated with the default servant
  - Need to invoke set_servant
  - Requires MULTIPLE_ID policy
- USE_SERVANT_MANAGER:
  - NON_RETAIN: Need to set servant locator
  - RETAIN: Need to set servant activator

ImplicitActivationPolicy

- IMPLICIT_ACTIVATION: support implicit activation
  - Requires SYSTEM_ID and RETAIN
- NO_IMPLICIT_ACTIVATION: implicit activation is not supported

POA Interface: Exceptions

local interface POA {

    exception AdapterAlreadyExists {};

    exception AdapterNonExistent {};

    exception InvalidPolicy {unsigned short index;};

    exception NoServant {};

    exception ObjectAlreadyActive {};

    exception ObjectNotActive {};

    exception ServantAlreadyActive {};

    exception ServantNotActive {};

    exception WrongAdapter {};

    exception WrongPolicy {};

POA Interface: POA Creation and Destruction

POA create_POA(
    in string adapter_name,
    in POAManager a_POAManager,
    in CORBA::PolicyList policies)
        raises (AdapterAlreadyExists, InvalidPolicy);
POA find_POA(
    in string adapter_name,
    in boolean activate_it)
        raises (AdapterNonExistent);
void destroy(
    in boolean etherealize_objects,
    in boolean wait_for_completion);

POA Interface: Policy Creation

ThreadPolicy create_thread_policy(in ThreadPolicyValue value);
LifespanPolicy create_lifespan_policy(in LifespanPolicyValue value);
IdUniquenessPolicy create_id_uniqueness_policy(
    in IdUniquenessPolicyValue value);
IdAssignmentPolicy create_id_assignment_policy(
    in IdAssignmentPolicyValue value);
ImplicitActivationPolicy create_implicit_activation_policy(
    in ImplicitActivationPolicyValue value);
ServantRetentionPolicy create_servant_retention_policy(
    in ServantRetentionPolicyValue value);
RequestProcessingPolicy create_request_processing_policy(
    in RequestProcessingPolicyValue value);

POA Interface: Attributes

readonly attribute string the_name;

readonly attribute POA the_parent;

readonly attribute POAList the_children;

readonly attribute POAManager the_POAManager;

attribute AdapterActivator the_activator;

POA Interface: Servant Managers

ServantManager get_servant_manager()
                raises (WrongPolicy);

void set_servant_manager(
      in ServantManager imgr)
                raises (WrongPolicy);

POA Interface: Default Servants

Servant get_servant()
      raises (NoServant, WrongPolicy);

void set_servant(in Servant p_servant)
      raises (WrongPolicy);

POA Interface: Activation and Deactivation

ObjectId activate_object(

    in Servant p_servant)

            raises (ServantAlreadyActive, WrongPolicy);

void activate_object_with_id(

    in ObjectId id,

    in Servant p_servant)

            raises (ServantAlreadyActive,

                  ObjectAlreadyActive, WrongPolicy);

void deactivate_object(

    in ObjectId oid)

            raises (ObjectNotActive, WrongPolicy);

POA Interface: Reference Creation

Object create_reference (
    in CORBA::RepositoryId intf)
        raises (WrongPolicy);
Object create_reference_with_id (
    in ObjectId oid,
    in CORBA::RepositoryId intf
);

POA Interface: Identity Mapping

ObjectId servant_to_id(in Servant p_servant)
> raises (ServantNotActive, WrongPolicy);

Object servant_to_reference(in Servant p_servant)
> raises (ServantNotActive, WrongPolicy);

Servant reference_to_servant(in Object reference)
> raises(ObjectNotActive, WrongAdapter, WrongPolicy);

ObjectId reference_to_id(in Object reference)
> raises (WrongAdapter, WrongPolicy);

Servant id_to_servant(in ObjectId oid)
> raises (ObjectNotActive, WrongPolicy);

Object id_to_reference(in ObjectId oid)
> raises (ObjectNotActive, WrongPolicy);

readonly attribute CORBA::OctetSeq id;

POACurrent

- Current objects: Thread-local
- Initial reference: "POACurrent"
- Determines object reference of current operation

```
local interface Current : CORBA::Current {
    exception NoContext { };
    POA         get_POA()          raises (NoContext);
    ObjectId    get_object_id()    raises (NoContext);
    Object      get_reference()    raises(NoContext);
    Servant     get_servant()      raises(NoContext);
};
```