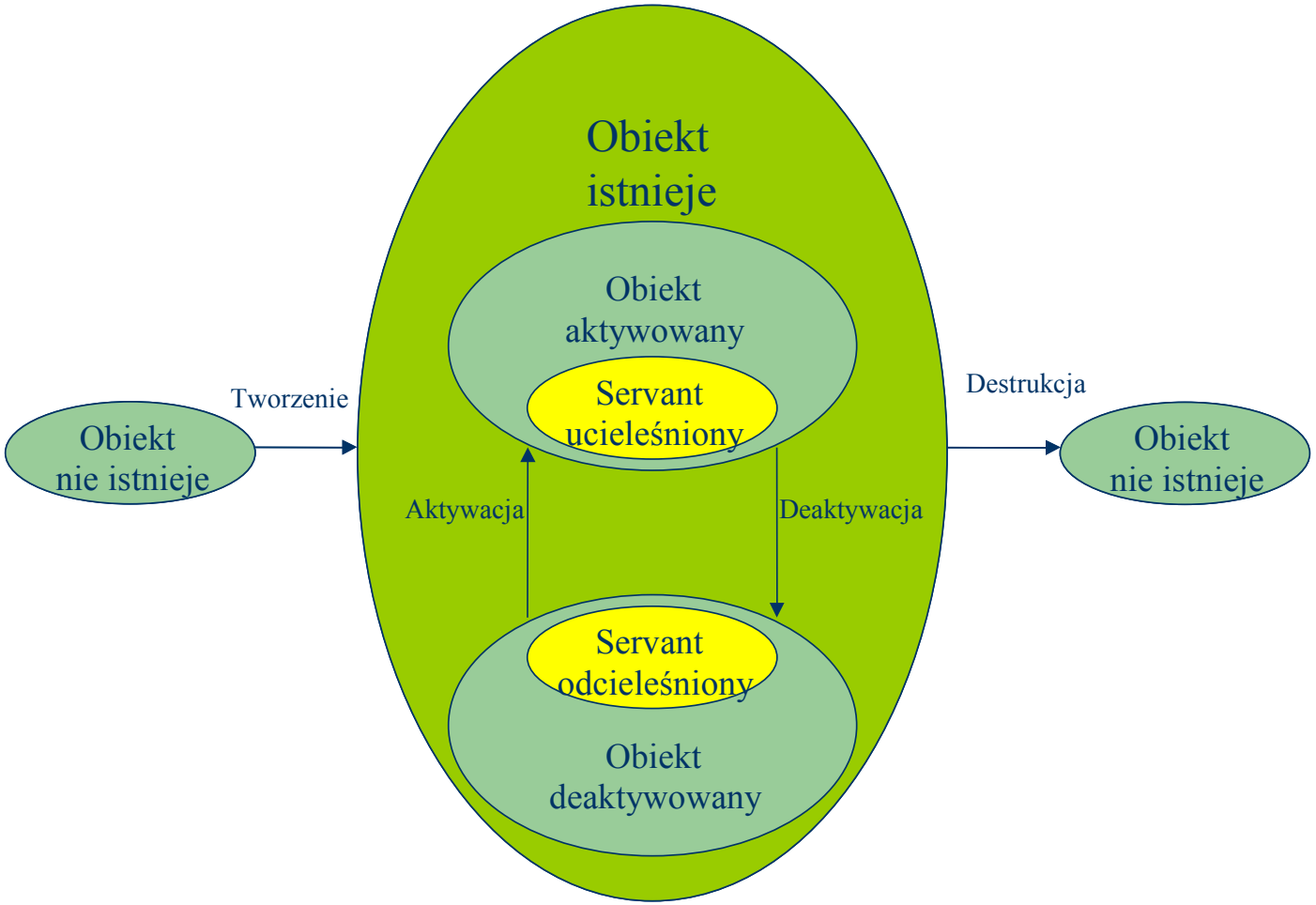
Portable Object Adapter

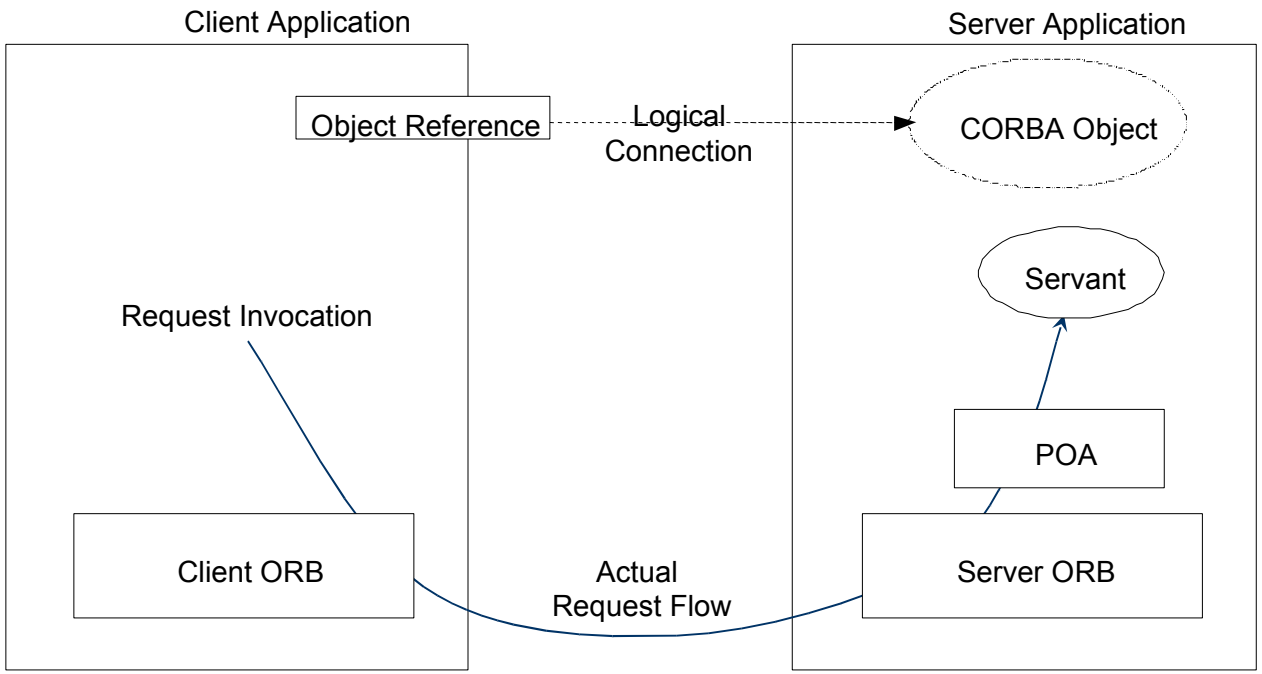
Rola POA

- Tworzenie obiektów
- Rejestracja serwantów
- Rozsyłanie zapytań do poszczególnych obiektów
- Obiekt
  - tworzenie (*create*)
  - destrukcja (*destroy*)
  - aktywacja (*activate*)
  - deaktywacja (*deactivate*)
- Serwant
  - ucieleśnienie (*incarnate*)
  - odcieleśnienie (*etherealize*)

Stany obiektu i cykl życia serwantu



Rozsyłanie zapytań przez ORB



POA Policies

- Każdy POA jest odpowiedzialny za grupę obiektów o podobnych cechach
- POA Policies służą do definiowania cech POA i obiektów za które jest odpowiedzialnych
- Są dziedziczone z interfejsu **`CORBA::Policy`**

```
module CORBA {
   typedef unsigned long PolicyType;

interface Policy{
   readonly attribute PolicyType policy_type;

   policy copy ();
   void destroy();
};
typedef sequence <Policy> PolicyList;
// …
};
```

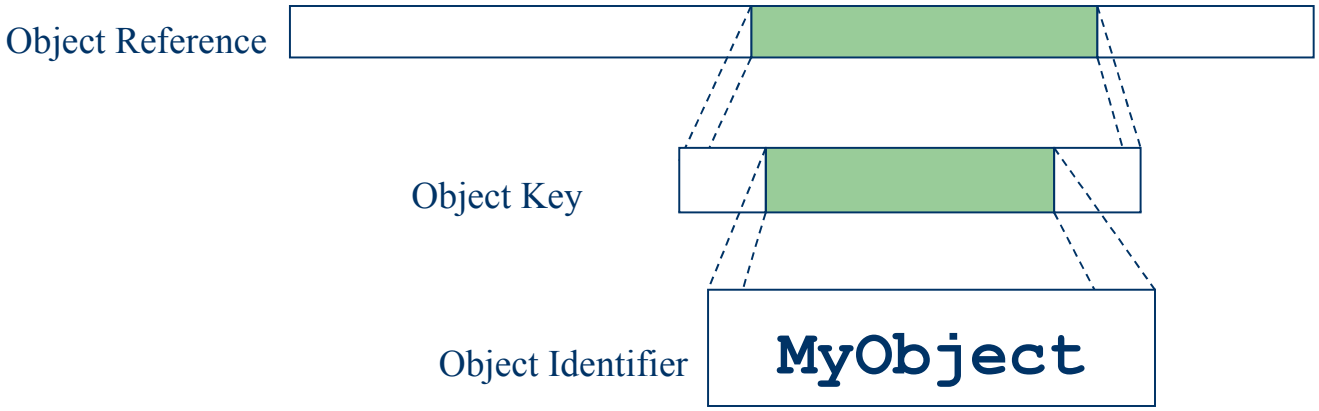
- POA Policies to obiekty o zasięgu lokalnym

POA Policies (cd.)

- Czas życia obiektu

```
module PortableServer {
  enum LifespanPolicyValue {
    TRANSIENT, PERSISTENT
    };
  interface LifespanPolicy :  CORBA::Policy {
    readonly attribute LifespanPolicyValue value;
  };
  // ...
};
```

POA Policies (cd.)



- Identyfikatory obiektu

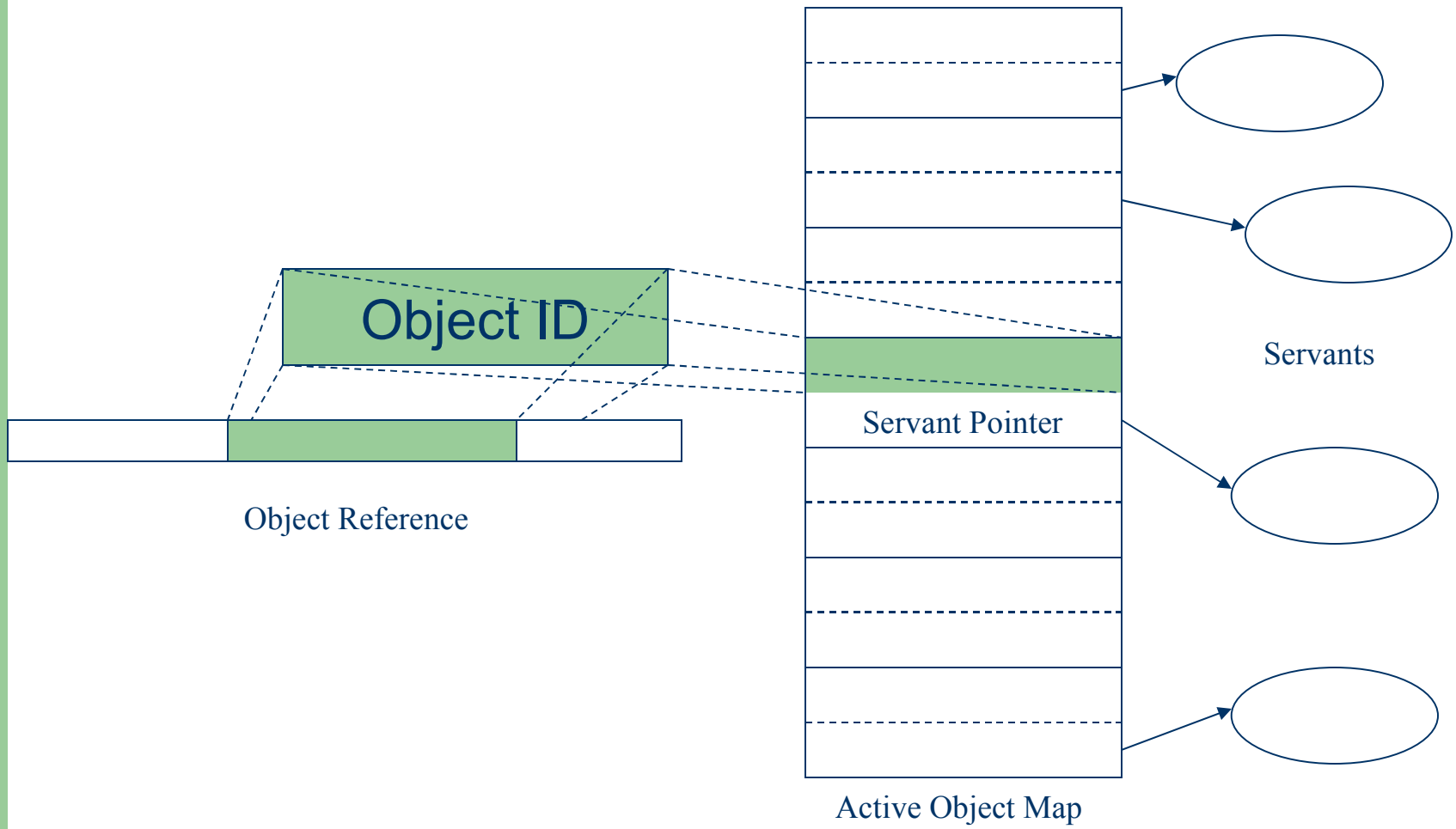
```
module PortableServer {
  enum IdAssignmentPolicyValue {
    USER_ID, SYSTEM_ID
    };
  interface IdAssignmentPolicy :  CORBA::Policy {
    readonly attribute IdAssignmentPolicyValue value;
  };
  // ...
};
```

POA Policies (cd.)

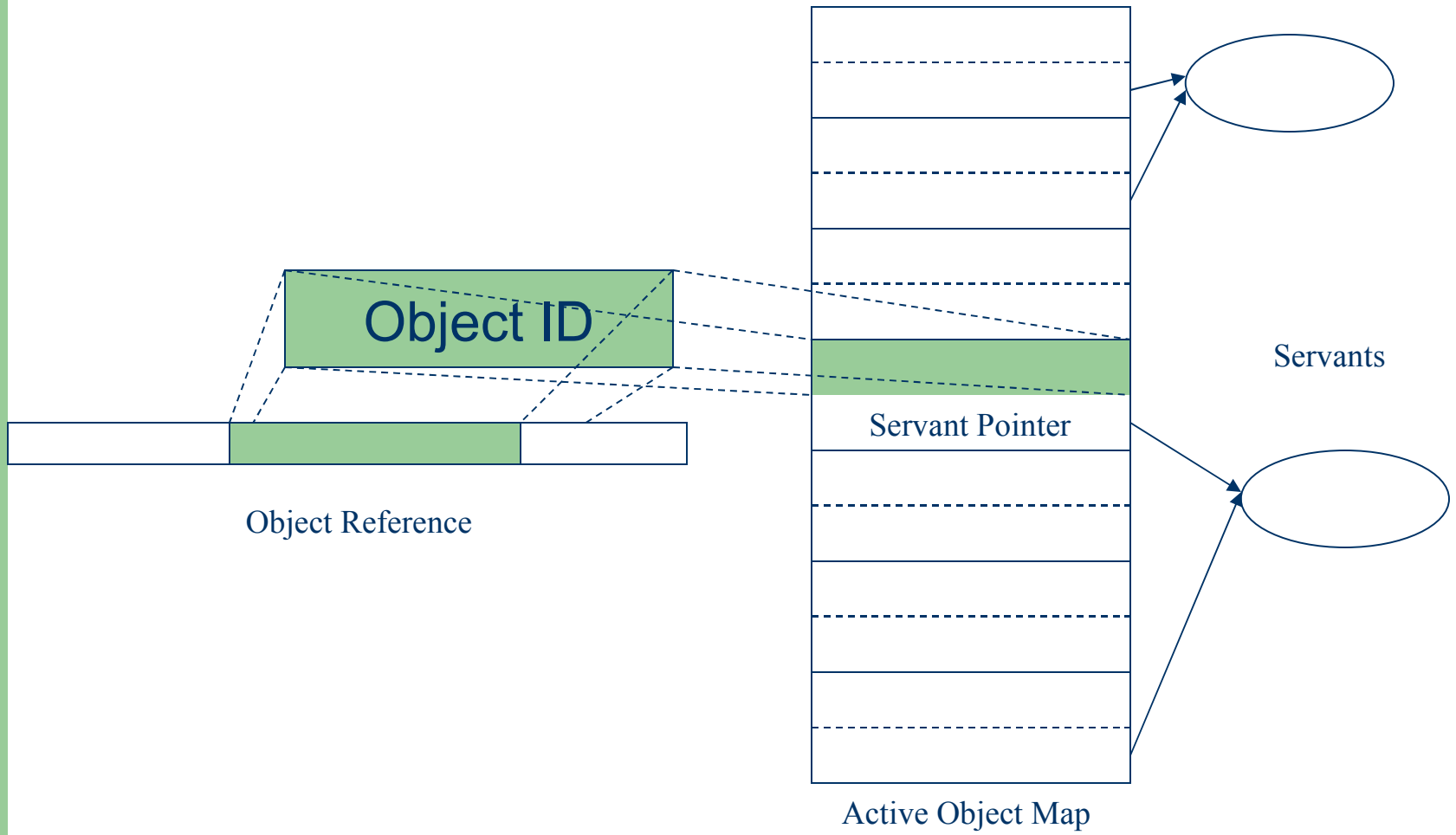
- Mapping obiektów na serwanty

```
module PortableServer {
 enum IdUniquenessPolicyValue {
    UNIQUE_ID, MULTIPLE_ID
   };
 interface IdUniquenessPolicy :  CORBA::Policy {
  readonly attribute IdUniquenessPolicyValue value;
 };
  // ...
};
```

POA Policies – UNIQUE_ID



POA Policies – `MULTIPLE_ID`



POA Policies (cd.)

- Niejawna aktywacja

```
module PortableServer {
 enum ImplicitActivationPolicyValue {
   IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION
   };
 interface ImplicitActivationPolicy :  CORBA::Policy {
    readonly attribute ImplicitActivationPolicyValue value;
  };
  // ...
};
```

POA Policies (cd.)

- Dopasowywanie zapytań do serwantów

```
module PortableServer {
 enum RequestProcessingPolicyValue {
   USE_ACTIVE_OBJECT_MAP_ONLY,
   USE_DEFAULT_SERVANT,
   USE_SERVANT_MANAGER
   };
 interface RequestProcessingPolicy :  CORBA::Policy {
   readonly attribute RequestProcessingPolicyValue value;
 };
  // ...
};
```

POA Policies (cd.)

- Kojarzenie ObjectID z serwantami

```
module PortableServer {
 enum ServantRetensionPolicyValue {
   RETAIN, NON_RETAIN
};
 interface ServantRetensionPolicy :  CORBA::Policy {
   readonly attribute ServantRetensionPolicyValue value;
 };
  // ...
};
```

POA Policies (cd.)

- Użycie wątków przy obsłudze zgłoszeń

```
module PortableServer {
 enum ThreadPolicyValue {
   ORB_CTRL_MODEL, SINGLE_THREAD_MODEL, MAIN_THREAD_MODEL
};
 interface ThreadPolicy :  CORBA::Policy {
   readonly attribute ThreadPolicyValue value;
 };
  // ...
};
```

POA Policies (cd.)

- Policy Factory

```
module PortableServer {
  interface POA {
   LifespanPolicy create_lifespan_policy
        (in LifespanPolicyValue value);
   IdAssignmentPolicy create_id_assignment_policy
        (in IdAssignmentPolicyValue value);
   // ...
   ThreadPolicy create_thread_policy
        (in ThreadPolicyValue value);
  };
};
```

- Po użyciu (stworzeniu POA) należy uzyskane obiekty usunąć za pomocą operacji **`destroy`**

Tworzenie POA

```
module PortableServer {
  interface POAManager;
interface POA {
    exception AdapterAlreadyExists {};
    exception InvalidPolicy { unsigned short index; };
    POA create_POA(
          in string adapter_name;
          in POAManager manager;
          in CORBA::PolicyList policies
       ) raises(AdapterAlreadyExists, InvalidPolicy);
    void destroy(in boolean etherealize_objects,
                 in boolean wait_for_completion);
    // ...
  };
};
```

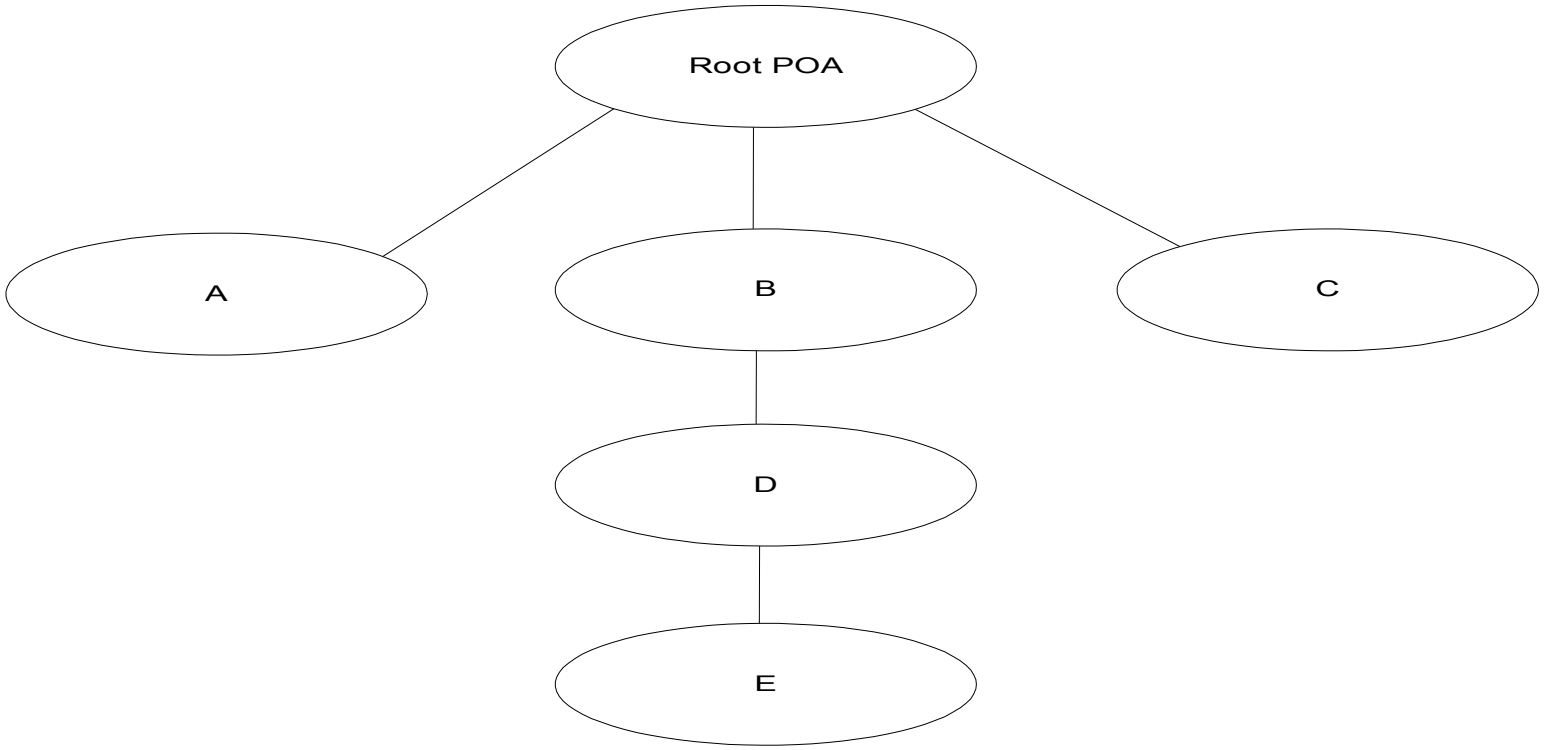
Tworzenie POA (cd.)

```
// Initialize the ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
// Get a reference to the Root POA
CORBA::Object_var obj =
  orb->resolve_initial_references("RootPOA");
PortableServer::POA_var root_poa =
  PortableServer::POA::_narrow(obj);
assert(!CORBA::is_nil(root_poa));
// Create empty PolicyList for child POA
CORBA::PolicyList policy_list;
// Invoke create_POA to create the child
root_poa->create_POA("child",
   PortableServer::POAManager::_nil(),
   policy_list);
```

Tworzenie POA (cd.)

```
// Set up a nil POAManager reference
// to pass to each create_POA call
PortableServer::POAManager_var nil_mgr =
  PortableServer::POAManager::_nil();
// Create POA A, child of the Root POA
PortableServer::POA_var poa_A =
  root_poa->create_POA("A", nil_mgr, policy_list);
// Create POA B, child of the Root POA
PortableServer::POA_var poa_B =
  root_poa->create_POA("B", nil_mgr, policy_list);
// Create POA C, child of the Root POA
PortableServer::POA_var poa_C =
  root_poa->create_POA("C", nil_mgr, policy_list);
// Create POA D, child of the POA B
PortableServer::POA_var poa_D =
  poa_B->create_POA("D", nil_mgr, policy_list);
// Create POA E, child of the POA D
PortableServer::POA_var poa_E =
  poa_D->create_POA("E", nil_mgr, policy_list);
```

Tworzenie POA (cd.)



Tworzenie POA (cd.)

```
// Create a PERSISTENT LifespanPolicy object
PortableServer::LifespanPolicy_var lifespan =
  root_poa->create_lifespan_policy(PortableServer::PERSISTENT);
// Create PolicyList
CORBA::PolicyList policy_list;
policy_list.length(1);
policy_list[0]=
  PortableServer::LifespanPolicy::_duplicate(lifespan);
// Create the child POA
PortableServer::POA_var child =
  root_poa->create_POA("child", nil_mgr, policy_list);
//Destroy our LifespanPolicy object
lifespan->destroy();
```

Typ IDL serwantu

```
module PortableServer {
   native Servant;
};
```

Interfejs serwantu

```
namespace PortableServer {
 class ServantBase {
 public:
  virtual ~ServantBase();
  virtual POA_ptr _default_POA();
  virtual CORBA::InterfaceDef_ptr _get_interface() throw(CORBA::SystemException);
  virtual CORBA::Boolean _is_a(const char * logical_type_id)
                                                  throw(CORBA::SystemException);
  virtual CORBA::Boolean _non_existent() throw(CORBA::SystemException);
  virtual void _add_ref();
  virtual void _remove_ref();
 protected:
  ServantBase();
  ServantBase(const ServantBase& base);
  ServantBase& operator=(const ServantBase & base);
 };
 typedef ServantBase * servant;
};
```

Thermometer_impl

```
Thermometer_impl::Thermometer_impl(CCS::AssetType anum) : m_anum(anum)
{
  m_ctrl->add_impl(anum); // Add self to map
};

Thermometer_impl::~Thermometer_impl()
{
  m_ctrl->remove_impl(m_anum);
};
```

Tworzenie i aktywacja obiektów

- Aplikacja może stworzyć obiekty bez tworzenia serwantów
- Aplikacja może jawnie lub niejawnie zarejestrować w POA serwant ucieleśniający dany obiekt, POA zapamięta skojarzenie serwantu z obiektem
- Aplikacja może dostarczyć obiekt typu *servant manager* który dynamicznie stworzy serwanty ucieleśniające obiekty w miarę potrzeby po przyjściu zapytania. POA może, lecz nie musi, zapamiętać skojarzenie serwantu z obiektem
- Aplikacja może dostarczyć serwant domyślny, który będzie użyty, gdy obiekt nie jest ucieleśniony przez żaden serwant

Tworzenie obiektów

```
module PortableServer{
   typedef sequence<octet> ObjectId;

   interface POA {
        Object create_reference(
               in CORBA::RepositoryId intf
               ) raises(WrongPolicy);

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

Tworzenie obiektów (cd.)

```
static CCS::Thermometer_ptr
make_dref(PortableServer::POA_ptr poa, CCS::AssetType anum)
{
    // Convert asset number to OID.
    ostringstream ostr;
    ostr << anum;
    string anum_str = ostr.str();
    PortableServer::ObjectId_var oid
        = PortableServer::string_to_ObjectId(anum_str.c_str());
    // Look at the model via the network to determine
    // the repository ID.
    char buf[32];
    if (ICP_get(anum, "model", buf, sizeof(buf)) != 0)
        abort();
    const char * rep_id = strcmp(buf, "Sens-A-Temp") == 0
                          ? "IDL:acme.com/CCS/Thermometer:1.0"
                          : "IDL:acme.com/CCS/Thermostat:1.0";
    // Make a new reference.
    CORBA::Object_var obj
        = poa->create_reference_with_id(oid, rep_id);
    return CCS::Thermometer::_narrow(obj);
}
```

Tworzenie obiektów (cd.)

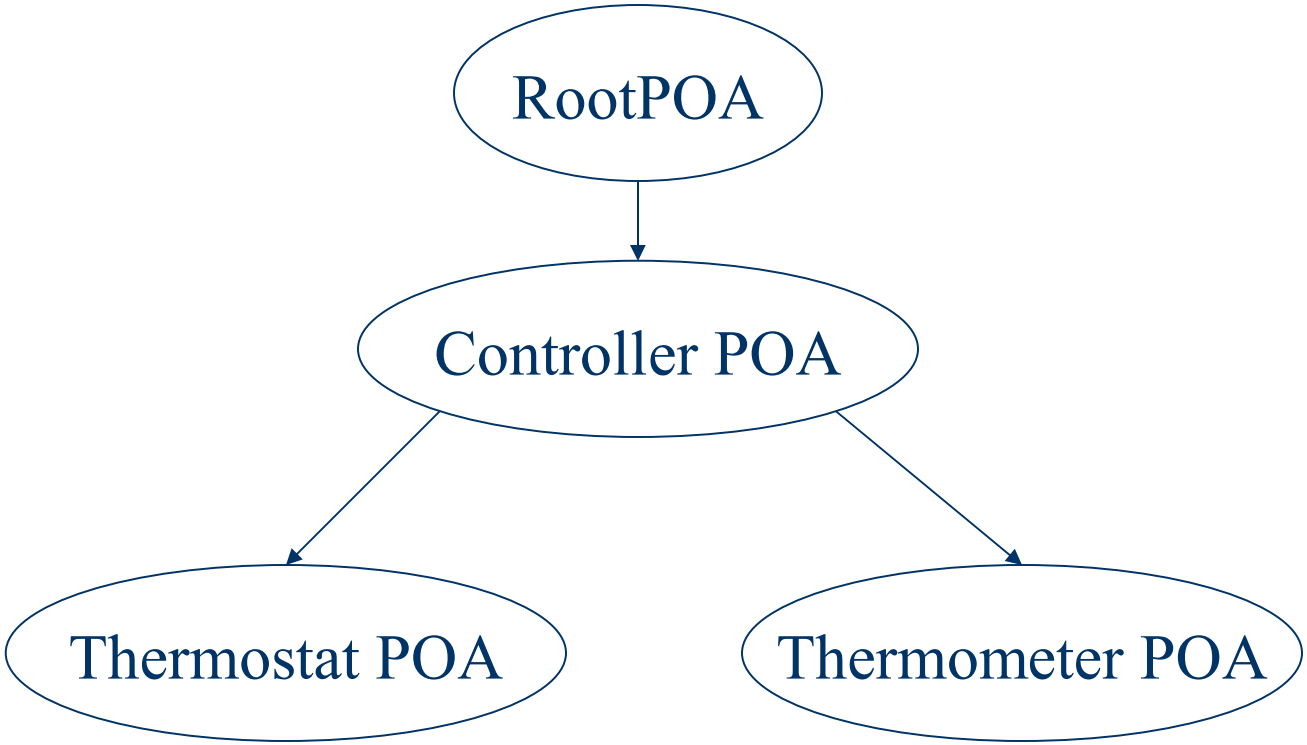
```
CCS::Controller::ThermometerSeq *
Controller_impl::
list() throw(CORBA::SystemException)
{
    // Create a new thermometer sequence. Because we know
    // the number of elements we will put onto the sequence,
    // we use the maximum constructor.
    CCS::Controller::ThermometerSeq_var listv
        = new CCS::Controller::ThermometerSeq(m_assets.size());
    listv->length(m_assets.size());
    // Loop over the m_assets set and create a
    // reference for each device.
    CORBA::ULong count = 0;
    AssetMap::iterator i;
    for (i = m_assets.begin(); i != m_assets.end(); ++i)
        listv[count++] = make_dref(m_poa, i->first);
    return listv._retn();
}
```

Narrowing

- `_narrow()` może wymagać konieczności połączenia się z serwerem i aktywacji obiektu
- Niektóre ORB próbują pewnych optymalizacji:
  - rzutowania w hierarchii klas proxy
  - porównania repository ID w referencji obiektu z repository ID typu docelowego
- Jeżeli to możliwe, najlepiej rzutować do rzeczywistego typu obiektu

```
if(strcmp(model, "Sens-A-Temp") == 0)
  return CCS::Thermometer::_narrow(obj);
else
  return CCS::Thermostat::_narrow(obj);
```

Hierarchia POA



Rejestracja serwantu

```
module PortableServer {
  interface POA {
    exception ServantAlreadyActive {};
    exception ObjectAlreadyActive {};
    exception WrongPolicy {};
    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);
  };
};
```

Serwant sterownika systemu

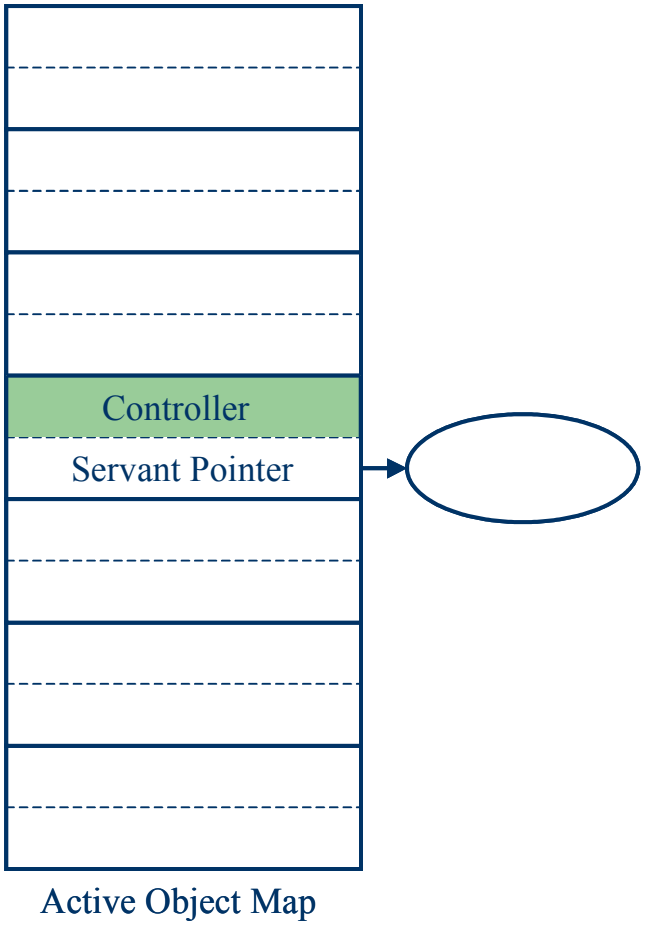
```
class Controller_impl : public virtual POA_CCS::Controller
{
public:
// IDL operations (not shown)
// Constructor and destructor
Controller_impl();
virtual ~Controller_impl();
// Helper functions to allow thermometers and
// thermostats to add themselves to the m_assets map
// and to remove themselves again
void add_impl(CCS::AssetType anum);
void remove_impl(CCS::AssetType anum);
CORBA::Boolean exists(CCS::AssetType anum) const;
private:
// Set type for storing known devices
typedef set<CCS::AssetType> AssetSet;
// Set of known devices
AssetSet m_assets;
// copy not supported
Controller_impl(const Controller_impl &);
void operator=(const Controller_impl &);
};
```

Aktywacja sterownika systemu

```
// Create our Controller servant
Controller_impl ctrl_servant;
// Create our Controller ObjectID
PortableServer::ObjectId_var oid =
   PortableServer::string_to_ObjectId("Controller");
// Activate our Controller
poa->activate_object_with_id(oid, &ctrl_servant);
```

- Dwa sposoby uzyskania object reference sterownika

```
// Activate Controller object for SYSTEM_ID POA
PortableServer::ObjectId_var oid =
   poa->activate_object(ctrl_servant);
// Obtain the object reference for the ObjectId
CORBA::Object_var object =
   poa->id_to_reference(oid);
// Narrow to the controller interface
CCS::Controller_var controller =
   CCS:Controller::_narrow(object);

// Obtain the Controller object reference
CCS::Controller_var ctrl=ctrl_servant._this();
```



Servant Managers

- Aktywacja obiektów na żądanie
- Policy **`USE_SERVANT_MANAGER`**
- Zależnie od **`ServantRetention`** Policy
  - **`RETAIN`** – **`ServantActivator`**
  - **`NON_RETAIN`** – **`ServantLocator`**

```
module PortableServer {
 exception ForwardRequest {
   Object forward_reference;
 };
 interface ServantManager {};
};
```

Servant Activators

```
module PortableServer {
 interface ServantActivator: Servant Manager {
 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 (cd.)

```
class ThermometerActivator_impl :
  public virtual POA_PortableServer::ServantActivator {
public:
  ThermometerActivator_impl(Controller_impl & ctrl);
  virtual ~ThermometerActivator_impl() {};
  virtual PortableServer::Servant
    incarnate( const PortableServer::ObjectId & oid,
               PortableServer::POA_ptr poa )
               throw(CORBA::SystemException,
                PortableServer::ForwardRequest);
  virtual void etherealize( const PortableServer::ObjectId & oid,
                            PortableServer::POA_ptr poa,
                            PortableServer::Servant serv,
                            CORBA::Boolean cleanup_in_progress,
                            CORBA::Boolean remaining_activations)
                throw(CORBA::SystemException);
private:
  Controller_impl & m_ctrl;
  //copy not supported
  ThermometerActivator_impl(const ThermometerActivator_impl & t);
  void operator=(const ThermometerActivator_impl & t);
};
```

Servant Activators (cd.)

```
CORBA::Boolean Controller_impl::exists(CCS::AssetType anum) const
{
  return m_assets.find(anum) != m_assets.end();
};

PortableServer::Servant ThermometerActivator_impl::
incarnate( const PortableServer::ObjectId & oid,
           PortableServer::POA_ptr poa )
           throw(CORBA::SystemException,
           PortableServer::ForwardRequest)
           throw(CORBA::SystemException,
                 PortableServer::ForwardRequest);
{
// Check to see if the object ID is valid
CORBA::String_var oid_string;
try {
  oid_string = PortableServer::ObjectId_to_string(oid);
} catch (const CORBA::BAD_PARAM&) {
  throw CORBA::OBJECT_NOT_EXIST();
};
```

Servant Activators (cd.)

```
// Get the asset number from the oid_string
istringstream istr(oid_string.in());
CCS::AssetType anum;
istr >> anum;
if (istr.fail()) throw CORBA::OBJECT_NOT_EXIST();
// Does the object ID denote one of our assets?
if (!m_ctrl->exists(anum)) throw CORBA::OBJECT_NOT_EXIST();
// Get the model identifier from the device
PortableServer::Servant servant = 0;
char model[32];
if(ICP_get(anum, "model", model, sizeof(model)) != 0)
  abort();
if(strcmp(model, "Sens-A-Temp") == 0)
  servant = new Thermometer_impl(anum);
else
  servant = new Thermostat_impl(anum);
return servant;
};
```

Servant Activators (cd.)

```
void ThermometerActivator_impl::
etherealize( const PortableServer::ObjectId & oid,
             PortableServer::POA_ptr poa,
             PortableServer::Servant servant,
             CORBA::Boolean cleanup_in_progress,
             CORBA::Boolean remaining_activations)
                          throw(CORBA::SystemException);
{
  if(!remaining_activations)
    delete servant;
};
```

Servant Locators

```
module PortableServer {
  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,
              out Cookie the_cookie,
              in Servant serv );
  };
};
```

Servant Locators (cd.)

```
class ThermometerLocator_impl :
  public virtual POA_PortableServer::ServantLocator
{
  public:
    ThermometerLocator_impl(Controller_impl & ctrl);
    virtual ~ThermometerLocator_impl(){};
    virtual PortableServer::Servant preinvoke (
        const PortableServer::ObjectId & oid,
        PortableServer::POA_ptr poa,
        const char * operation,
        void * & cookie) throw (
         CORBA::SystemException, PortableServer::ForwardRequest
       );
     virtual void postinvoke (
        const PortableServer::ObjectId & oid,
        PortableServer::POA_ptr poa,
        const char * operation,
        void * cookie,
        PortableServer::Servant servant) throw (CORBA::SystemException);
  private:
    Controller_impl & m_ctrl;
    // copy not supported
    ThermometerLocator_impl(const ThermometerLocator_impl & t);
    void operator=(const ThermometerLocator_impl & t);
};
```

Servant Locators (cd.)

```
PortableServer::Servant ThermometerLocator_impl preinvoke (
        const PortableServer::ObjectId & oid,
        PortableServer::POA_ptr poa,
        const char * operation,
        void * & cookie) throw (
         CORBA::SystemException, PortableServer::ForwardRequest
       )
{
 // Check to see if the object ID is valid
CORBA::String_var oid_string;
try {
  oid_string = PortableServer::ObjectId_to_string(oid);
} catch (const CORBA::BAD_PARAM&) {
  throw CORBA::OBJECT_NOT_EXIST();
};
```

Servant Locators (cd.)

```
// Get the asset number from the oid_string
istringstream istr(oid_string.in());
CCS::AssetType anum;
istr >> anum;
if (istr.fail()) throw CORBA::OBJECT_NOT_EXIST();
// Does the object ID denote one of our assets?
if (!m_ctrl->exists(anum)) throw CORBA::OBJECT_NOT_EXIST();
// Get the model identifier from the device
PortableServer::Servant servant = 0;
char model[32];
if(ICP_get(anum, "model", model, sizeof(model)) != 0)
  abort();
if(strcmp(model, "Sens-A-Temp") == 0)
  servant = new Thermometer_impl(anum);
else
  servant = new Thermostat_impl(anum);
return servant;
};
```

Servant Locators (cd.)

```
void ThermometerLocator_impl::postinvoke (
        const PortableServer::ObjectId & oid,
        PortableServer::POA_ptr poa,
        const char * operation,
        void * cookie,
        PortableServer::Servant servant) throw (CORBA::SystemException)
{
  delete servant;
};
```

Rejestracja Servant Managera

```
// Create our Controller servant
Controller_impl ctrl_servant;
// Create a ThermometerActivator servant
ThermometerActivator_impl manager_impl(ctrl_servant);
// Create a new transient servant manager object
// in the Root POA
PortableServer::ServantManager_var mgr_ref =
  manager_impl._this();
// Set the servant manager for another POA. Because we
// are registering a ServantActivator, we assume our
// POA has the RETAIN policy value
poa->set_servant_manager(mgr_ref);
```

Serwanty domyślne

```
module PortableServer {
  interface Current : CORBA::Current {
    exception NoContext {};
    POA get_POA() raises(NoContext);
    ObjectId get_object_id() raises( NoContext);
  };
  // ...
};

// Obtain a reference to the ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
// Obtain a reference to the POA Current
CORBA::Object_var obj =
  orb->resolve_initial_references("POACurrent");
// Narrow the result to the PortableServer::Current interface
PortableServer::Current_var cur =
  PortableServer::Current::_narrow(obj);
```

Domyślny serwant dla termometru

```
class Thermometer_impl : public virtual POA_CCS::Thermometer,
  public virtual PortableServer::RefCountServantBase {
public:
  Thermometer_impl(PortableServer::Current_ptr current);
  virtual ~Thermometer_impl(){};
  // Functions for the Thermometer attributes
  virtual CCS::ModelType model() throw(CORBA::SystemException);
  // ...
  static Controller_impl * m_ctrl; // My controller
protected:
  PortableServer::Current_var m_current;
  // Helper function that extracts asset number from
  // the target ObjectId
  CCS::AssetType get_target_asset_number()
        throw(CORBA::SystemException);
  // Helper functions that read data from the device
   static CCS::ModelType get_model(CCS::AssetType anum);
  // ...
private:
  // copy not supported for this class
  Thermometer_impl(const Thermometer_impl & therm);
  void operator=(const Thermometer_impl & therm);
};
```

Domyślny serwant dla termometru (cd.)

```
Thermometer_impl::Thermometer_impl(PortableServer::Current_ptr current)
  : m_current(PortableServer::Current::_duplicate(current)) {};

CCS::ModelType Thermometer_impl::model() throw (CORBA::SystemException)
{
  CCS::AssetType anum = get_target_asset_number();
  return get_model(anum);
}
CCS::AssetType Thermometer_impl::get_target_asset_number()
  throw(CORBA::SystemException)
{
  // Get the target ObjectId
  PortableServer::ObjectId_var oid = m_current->get_object_id();
  // Check to see if the object ID is valid
  CORBA::String_var asset_str;
  try {
    asset_str = PortableServer::ObjectId_to_string(oid);
  } catch(const CORBA::BAD_PARAM&) {
    throw CORBA::OBJECT_NOT_EXIST();
  };
  // Convert the ID string into an asset number
  istringstream istr(asset_str.in());
  CCS::AssetType anum;
  istr >> anum;
  if (istr.fail()) throw CORBA::OBJECT_NOT_EXIST();
  return anum;
};
```

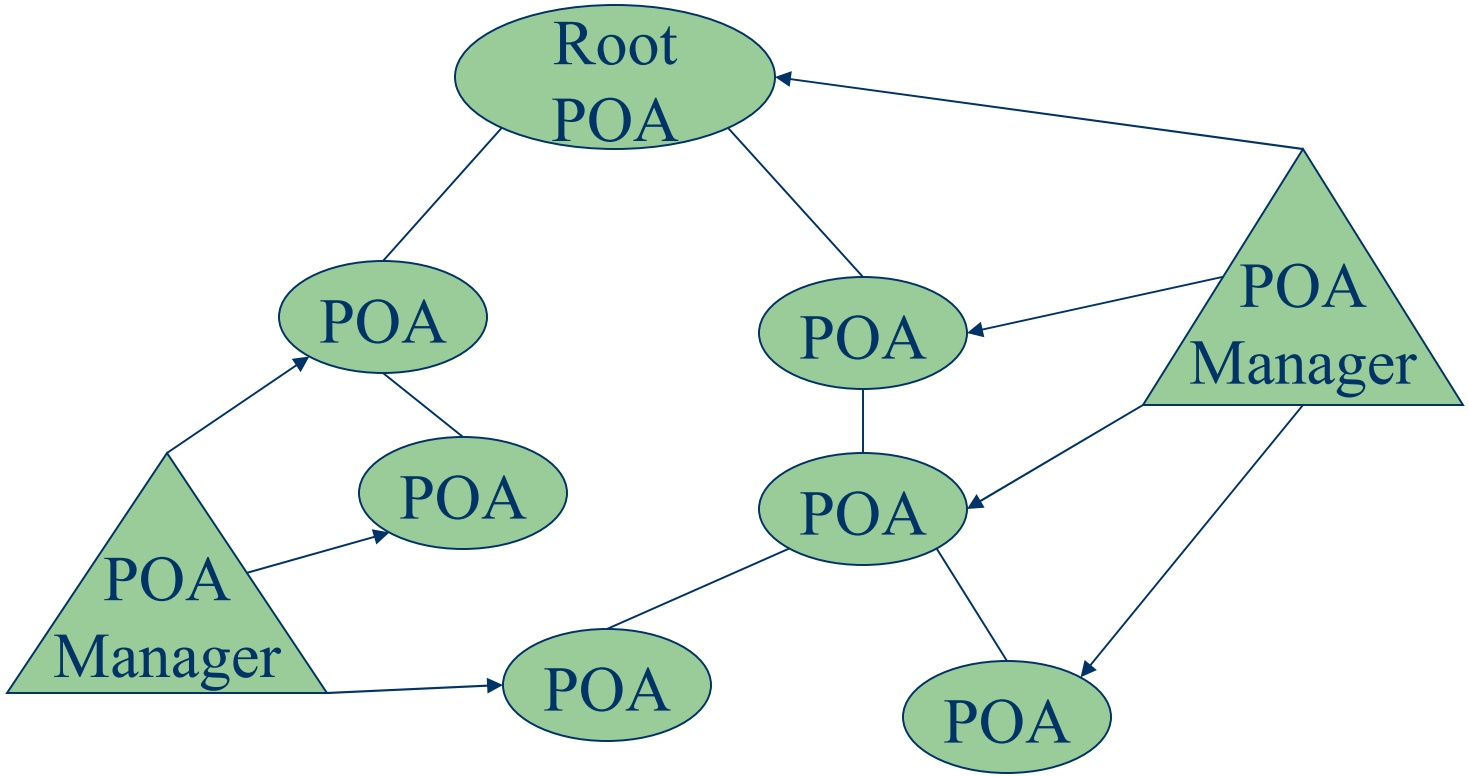
Ustawienie domyślnego serwantu

```
// Create a default servant
Thermometer_impl * dflt_servant = new Thermometer_impl(cur);
// Register it with the POA
poa->set_servant(dflt_servant);
// Because our servant inherits reference counting
// from RefCountServantBase, we call _remove_ref because
// we no longer need the servant
dflt_servant->_remove_ref();

// Use a ServantBase_var to capture the return value
// of get_servant
PortableServer::ServantBase_var servant = poa->get_servant();

// Use dynamic_cast to get back to our original
// default servant's type
Thermometer_impl * dflt_servant =
  dynamic_cast<Thermometer_impl *>(servant.in());
```

Deaktywacja obiektów

```
module PortableServer {
  interface POA {
    exception ObjectNotActive {};
    exception WrongPolicy {};
    void deactivate_object(in ObjectId oid)
         raises(ObjectNotActive, WrongPolicy);
  };
};

void SomeServant::destroy() throw (CORBA::SystemException)
{
  my_poa->deactivate_object(my_object_id);
  delete this; // looming disaster
};
```

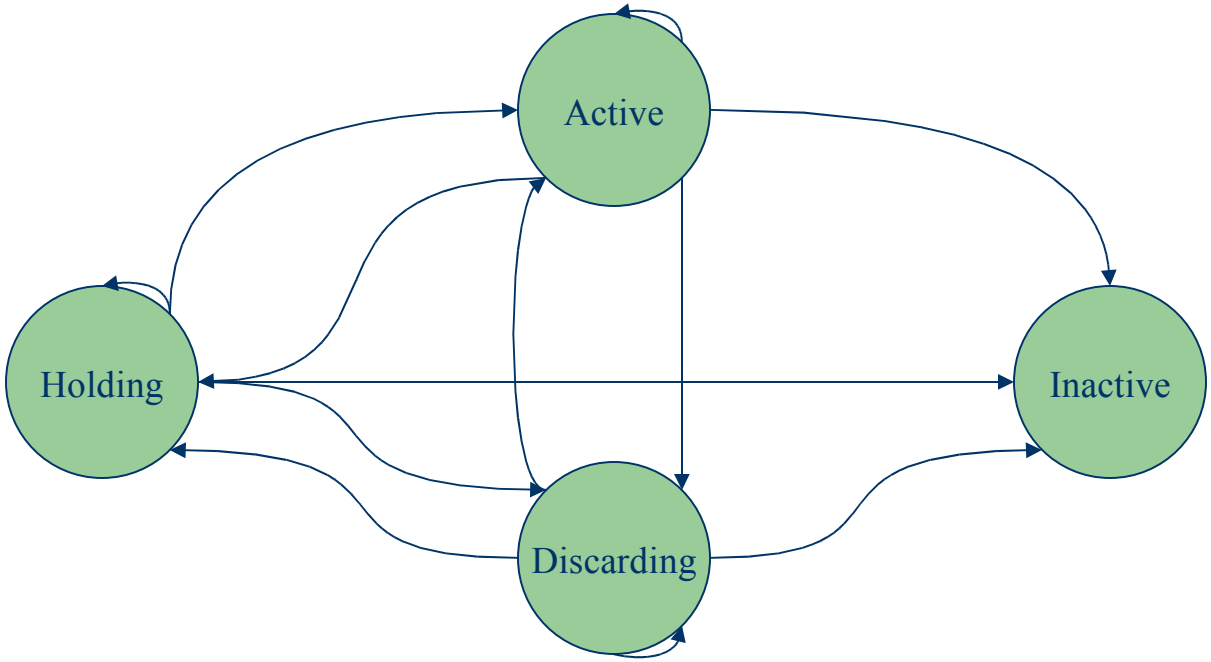
Sterowanie przepływem zapytań



Sterowanie przepływem zapytań (cd.)

```
module PortableServer {
  interface POAManager {
    exception AdapterInactive{};
    enum State { HOLDING, ACTIVE,
                 DISCARIDING, INACTIVE };
    State get_state();
    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);
  };
};
```

Sterowanie przepływem zapytań (cd.)



Obsługa zdarzeń przez ORB

```
#pragma prefix "omg.org"
module CORBA {
  interface ORB {
    void run();
    void shutdown(in boolean wait_for_completion);
    boolean work_pending();
    void perform_work();
    // ...
  };
  // ...
};
```

Obsługa zdarzeń przez ORB (cd.)

```
// The handle_gui_events function allows the user
// interface to referesh itself and handle its events
// It returns true if the user has clicked the
// "exit" button
extern bool handle_gui_events();

int main(int argc, char * argv[])
{
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
  // Initialize POAs and POAManagers and then activate
  // objects (not shown)
  // Enter event loop
  bool done = false;
  while(!done) {
    if(orb->work_pending())
      orb->perform_work();
    done = handle_gui_events();
  }
  orb->shutdown(1);
  return 0;
};
```

Zakończenie aplikacji

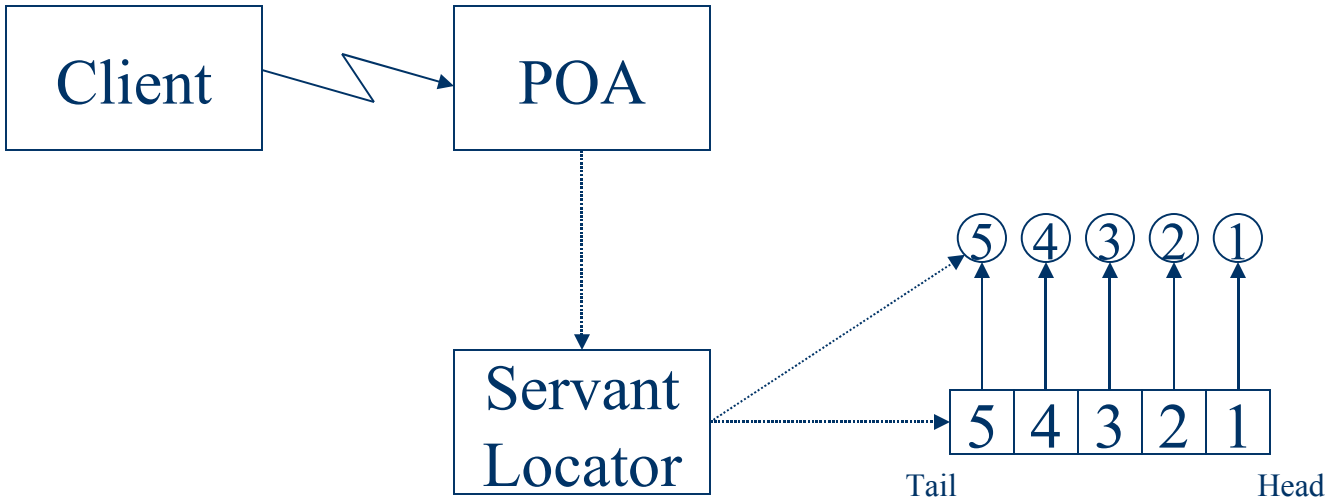
- Time-out
- Sygnał

```
// File-static ORB reference
static CORBA::ORB_var orb;
// Signal handling function
static void async_handler()
{
  if(!CORBA::is_nil(orb))
    orb->shutdown(0);
};
int main(int argc, char * argv[]) {
// First set up our asynchronous
// signal handler
TerminationHandler::set_handler
                      (async_handler);
// Initialize the ORB
orb = CORBA::ORB_init(argc, argv);
// ...
};
```

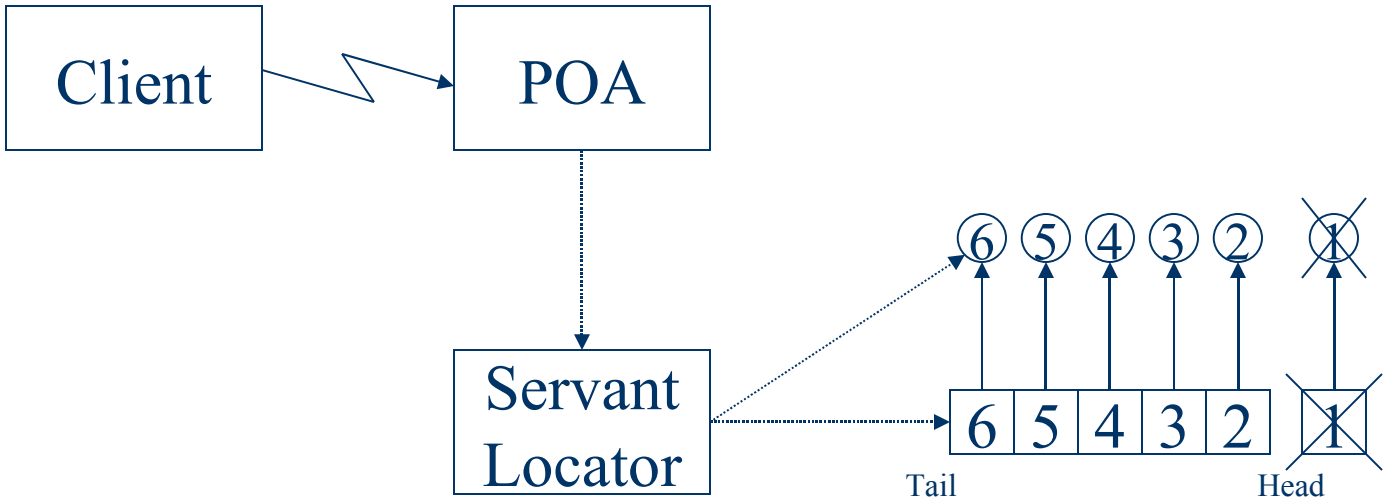
- Żądanie CORBA

```
interface ProcessTerminator {
  void shutdown();
};

void
MyProcessTerminator::shutdown()
   throw(CORBA::SystemException)
{
  orb->shutdown();
};
```

Evictor Pattern



Evictor Pattern (cd.)



Implementation Repository

Logical Server Name	POA Name	Start-Up Command	Host and Port
CCS	thermometer		bobo.acme.com:1780
CCS	thermostat		bobo.acme.com:1780
CCS	controller	rsh bobo /opt/CCS/CCS_svr	bobo.acme.com:1799
NameService	ns_poa	/opr/myorb/bin/name_svr –v	
Payroll	PR_V1		fifi.acme.com:1253
Stock	dept_1		
Stock	dept_2		