Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

Training

ANSAwise - Managing ANSAware Applications

Chris Mayers

Abstract

Organizations have changing requirements; this will involve changing the distributed applications that are deployed; however this should have minimal impact on the day-to-day running of the business.

This implies technical flexibility in the configuration and management of distributed applications.

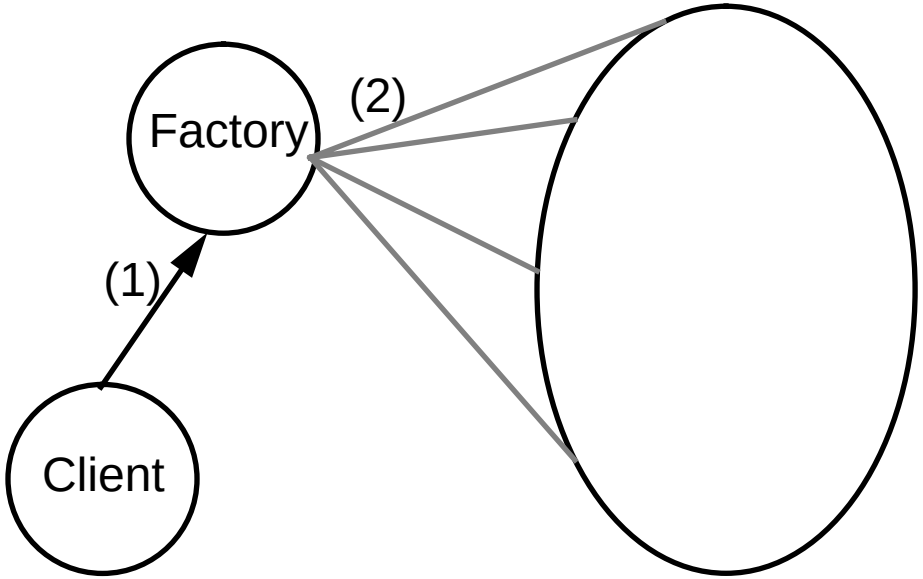
This module of the ANSAwise training programme explains the different methods of creating and activating a server. It also explains the role of the ANSAware Factory and Node Manager services. Participants then modify the Simple Bank server to be manageable via the Factory and Node Manager, and also to handle new error conditions accordingly.

APM.1588.01 **Approved** 2nd October 1995

Briefing Note

Distribution:
Supersedes:
Superseded by:

ANSAware 4.1
Managing ANSAware Applications

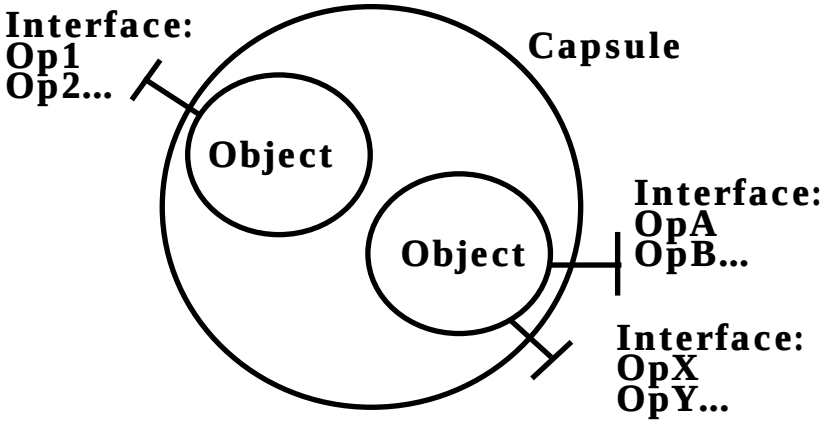


In this session

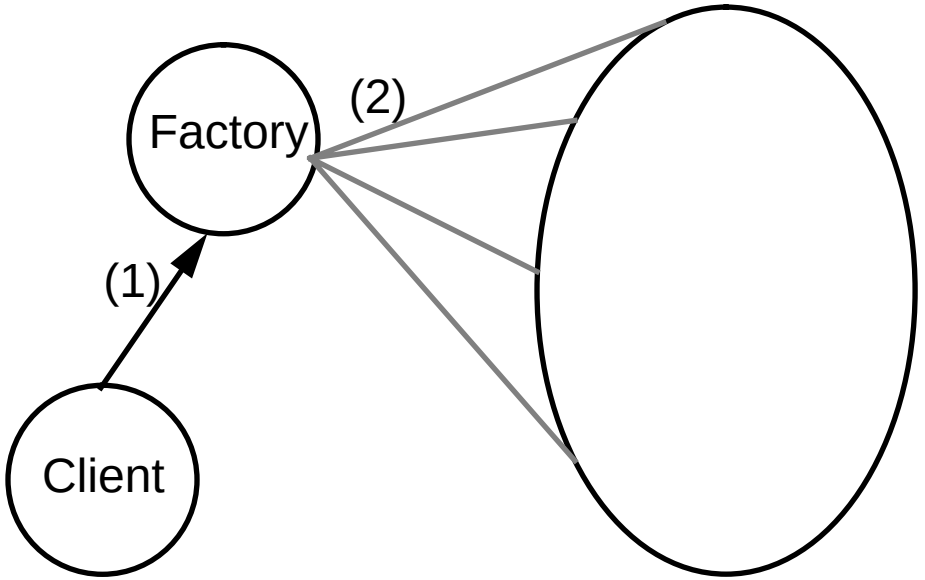
- **Explain the use of the Factory service**
- **Show how the Factory service supports the dynamic creation of capsules, and objects and interfaces within them**
- **Explain how the Node Manager service allows management of these dynamic services**

Factory services

- **Factory services (called Factories) provide a service for:**
  - **Creation/destruction of capsules**
  - **Simple monitoring of the capsules it creates**
- **Once created, capsules then provide a service for:**
  - **Creation/destruction of objects within a capsule**
  - **Creation/destruction of interfaces within an object**



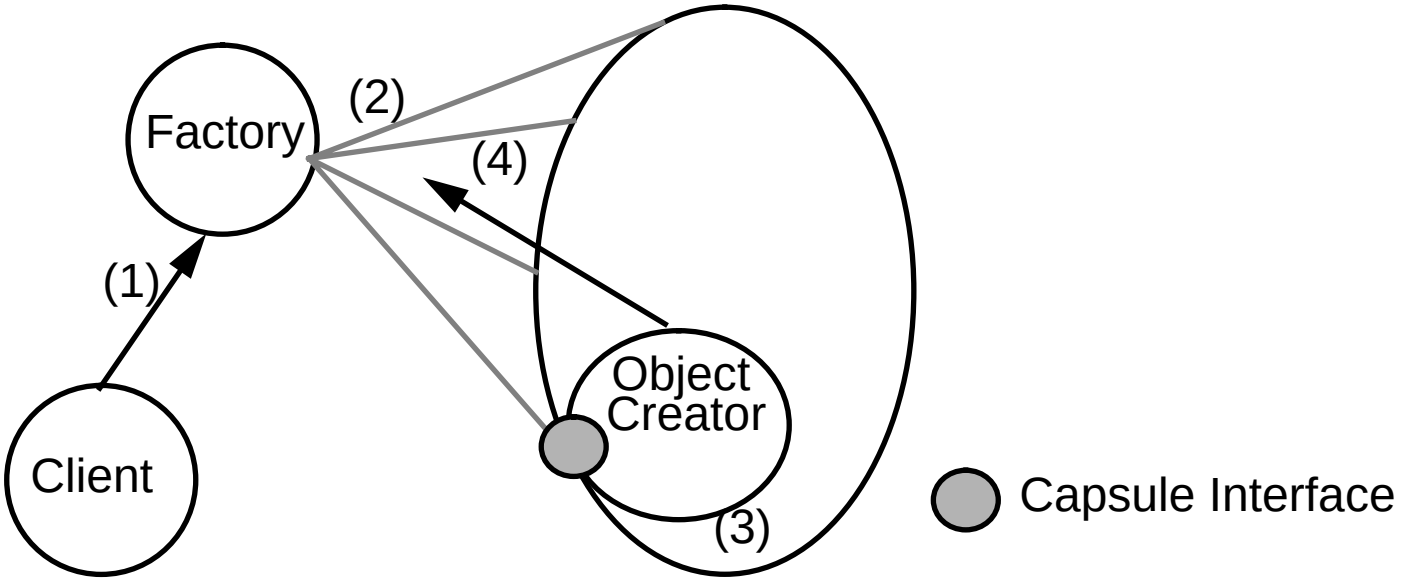
Capsule Creation



1. Client asks factory to Instantiate a capsule

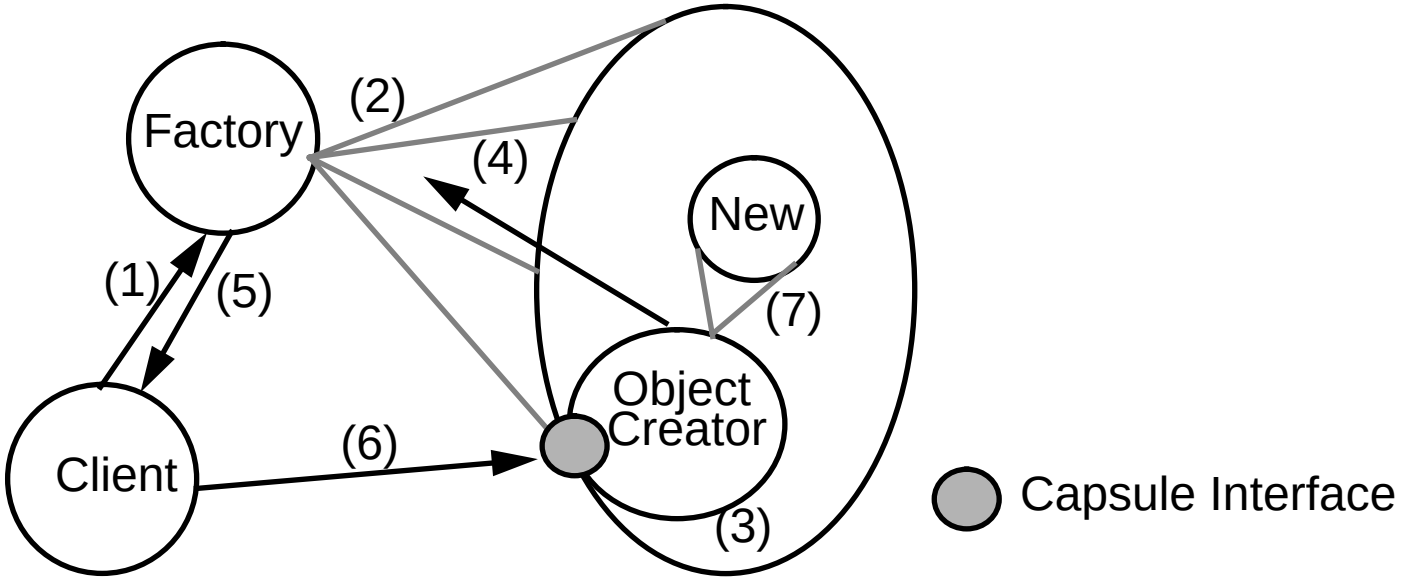
2. Factory instantiates a new capsule

Initial Object Creation



3. **New capsule creates a single object with at least a single Capsule interface. Any number of other interfaces may also be created**
4. **References to the Capsule and any other interfaces are returned to the factory**

Interface Creation



5. **These are in turn returned to the client**
6. **The client may then use the Capsule interface reference to create yet more objects**
7. **A new object is created**

Capsule structure for dynamic services

- In order for a server to be able to be started by a factory, it needs some changes
- A `! MANAGED` PREPC statement is required

  `! MANAGED ObjName`

  - the *ObjName* is a name you choose for this template
- Capsules can support multiple objects
  - effectively, they are templates for creating interfaces
  - the *ObjName* specifies the object to be used
  - this name determines the name of `Create_`... and `Destroy_`... functions for each such object

The Create and Destroy functions

- **Create_... function**

```
ansa_StatePtr Create_ObjName_Object( argc, argv, envp,
                                     results )
    int argc;
    char *argv[], *envp[];
    InstantiateResult *results;
```

- **Destroy_... function**

```
ansa_Boolean Destroy_ObjName_Object ( state )
    ansa_StatePtr state;
```

Creating an object

- The `Create_`*`ObjName`*`_Object` function must:
  - 1. use arguments and environment (if any) to decide what action is required
  - 2. instantiate any interfaces required
  - 3. allocate and initialise any object state required (explained later)
  - 4. return any state and set up results (of type `InstantiateResult`) - a sequence of interface references to instantiated interfaces

Destroying an object

- The `Destroy_`*`ObjName`*`_Object` function must:
  - 1. destroy any interfaces
  - 2. free any object state set up by `Create_`*`ObjName`*`_Object`

Simple factory-startable service/1


```
#include “ansa.h”


#include “tFoo.h”

! MANAGED FooObj


! USE Foo

! DECLARE { ifref } : Foo SERVER
```

Simple factory-startable service/2


```
   ansa_StatePtr Create_FooObj_Object( ac, av, envp, results )
   int ac;
   char *av[], *envp[];
   InstantiateResult *results;
   {
      ansa_InterfaceRef ifref;

!     {ifref} :: Foo$Create(1)
      results->length = 1;  /* Assign results - seq. of ifrefs */
      results->data = &(ifref);
      return (ansa_StatePtr)NULL; /* No obj-state in this ex. */
   }
```

Simple factory-startable service/3


```
ansa_Boolean Destroy_FooObj_Object(state)
    ansa_StatePtr state;
{
    return ansa_TRUE;
}
...
/* Functions implementing Operations of “Foo” interface,
   body() function */
...
```

Dynamic Services

- **Such servers do not require a body( ) function**
  - **but if a body( )function is provided, the service can be started either from the command-line, or from a factory...**
  - **... so a body( ) function is recommended**

Object State

- **Objects have optional state**
  - **usually based on arguments and/or environment**
  - **which can be initialised by the Create_*ObjName*_Object function**
  - **which is easily accessible from the Destroy_*ObjName*_Object function**
- **Each call of the Create_ObjName_Object function creates a new object instance**
  - **which will need its own separately allocated state...**
  - **... which must be deallocated by the Destroy_*ObjName*_Object function**

The form of object state

- **Object state can be anything; it is application-defined**

```
typedef struct objstate {.../* any struct definition here */
                        } yourStateStruct;
Create_ObjName_Object( ...)
{
   p = system_allocate( sizeof( yourStateStruct) )
   ...
   /* ...store whatever is necessary in this structure...*/
   ...
   return ( ansa_StatePtr )p;
}
```

Using object state

- **Object state is often used for keeping track of interface instances**
  - **to record whether interface instances have been exported to the Trader**
- **If so, the Destroy_...() function can Withdraw the offer from Trader before destroying the interface instances**
  - **the earlier example could have used this technique...**
  - **...the next example does**

ANSA_MANAGED_EXPORT

- ANSA_MANAGED_EXPORT is an environment variable
- By convention, it is used to indicate whether the interface should be exported to the Trader
  - this fact needs to be recorded in the object state
- The `frun` command sets this environment variable
  - this is shown later

Factory-startable server program/1


```
#include “ansa.h”
#include “tFoo.h”

#define PROPSIZE 1024
char pbuf[PROPSIZE];

! MANAGED Foo

! USE Foo
! USE Trader
! DECLARE { ir, p->ref } : Foo SERVER
```

Factory-startable server program/2


```
void body( argc, argv, envp )
int argc;
char *argv[], *envp[];
{

...Foo Operation definitions ...

typedef struct objstate {
    ansa_InterfaceRef ref;
    ansa_Boolean export;
} ObjState;
```


Factory-startable server program/3


```
ansa_StatePtr Create_String_Object(ac, av, envp, results)
int ac;
char *av[], *envp[];
InstantiateResult *results;
{
ObjState *p;
   p = (ObjState *)system_allocate(sizeof(ObjState))
!  {p->ref} :: Foo$Create(1)
```

Factory-startable server program/4


```
   if( system_getenv("ANSA_MANAGED_EXPORT", envp) != (char *)0)
   {
      p->export = ansa_TRUE;
      (void)system_init_properties( pbuf, PROPSIZE, ac,av );
!     {}<-traderRef$Export("Foo", "/ansa/testservices", \
                           pbuf, p->ref)
   }
   else
      p->export = ansa_FALSE;
   results->length = 1;
   results->data = &(p->ref);
   return (ansa_StatePtr)p;
}
```

Factory-startable server program/5


```
ansa_Boolean Destroy_Echo_Object(state)
   ansa_StatePtr state;
{
ObjState *p;
   p = (ObjState *)state;
   if (p->export == ansa_TRUE)
!     traderRef$Withdraw( p->ref )
!  {} :: Foo$Destroy( p->ref )
   system_free ((char *)p);
   return ansa_TRUE;
}
```

Factory client tools

- Two command-line tools are provided to create and destroy capsules
  `frun`
  `fkill`
- Factories can only create and destroy capsules on the same node as themselves
  - but the `frun` and `fkill` commands can access factories on other nodes
- Factories can only create capsules on nodes that support capsule creation
  - DOS does not support capsule creation
- But the `frun` and `fkill` commands can be run from DOS
  - to access a factory on another node

Creating a capsule using frun

- To create a capsule

```
frun node template object arguments environment [client args ...]
```

  - the arguments and environment are passed to the template (program) . . .

```
frun "" server ObjName "" ""
15307
```

  - ...the capsule id of the created capsule is displayed

- `frun` sets `ANSA_MANAGED_EXPORT` before instantiating capsule

  - to inform objects to export their interface instances to the Trader

Starting a client together with the capsule

- Often a service will have an associated client application
  - a user interface to manage and monitor the service
- The `frun` program allows a client application to be started up at the same time as the capsule
- If the `client` argument is given, it will be started as a sub-process with `args`... passed to it
  - when `client` terminates, `frun` will terminate the object, then terminate the capsule
- If the client argument is omitted, no such application will be started

Destroying a capsule using fkill

- **To destroy a capsule**

  ```
  fkill host cid
  ```

  - **... for example**

  ```
  fkill "" 15307
  ```

  - **cid is the capsule id as displayed by `frun`**

- **You can only destroy capsules that were created dynamically**

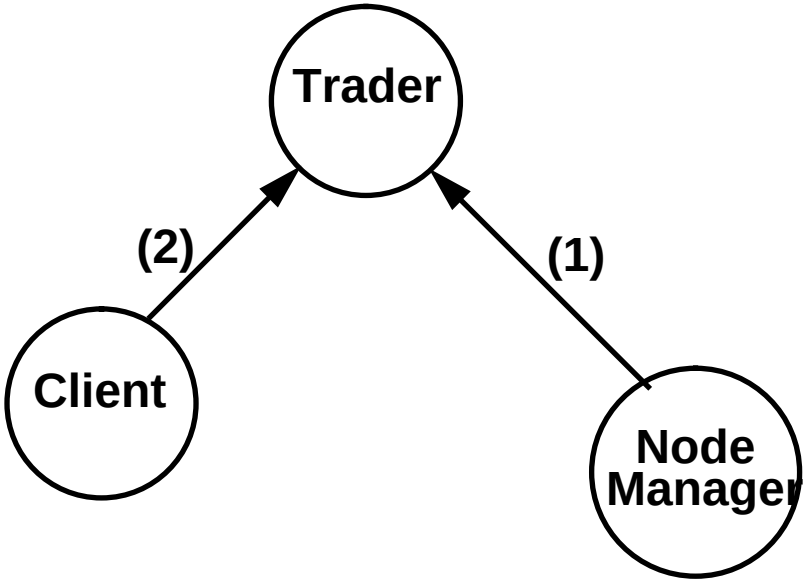
Node Manager

- **Provides an architectural interface**
  - **for creating services**
  - **for simple monitoring of services**
  - **for destroying of services**
- **Provides a database for describing services**
  - **each description is identified by an alias**

Static and Dynamic Services

- Static services are created by running aliases
  - they may be automatically restarted if they terminate
- Dynamic services are created via the federated trader interface's proxy export facility
  - if a service can be started via a Factory, it can be used with the Node Manager without change
- Service activations may be destroyed
- The Node Manager state is persistent (checkpointed)

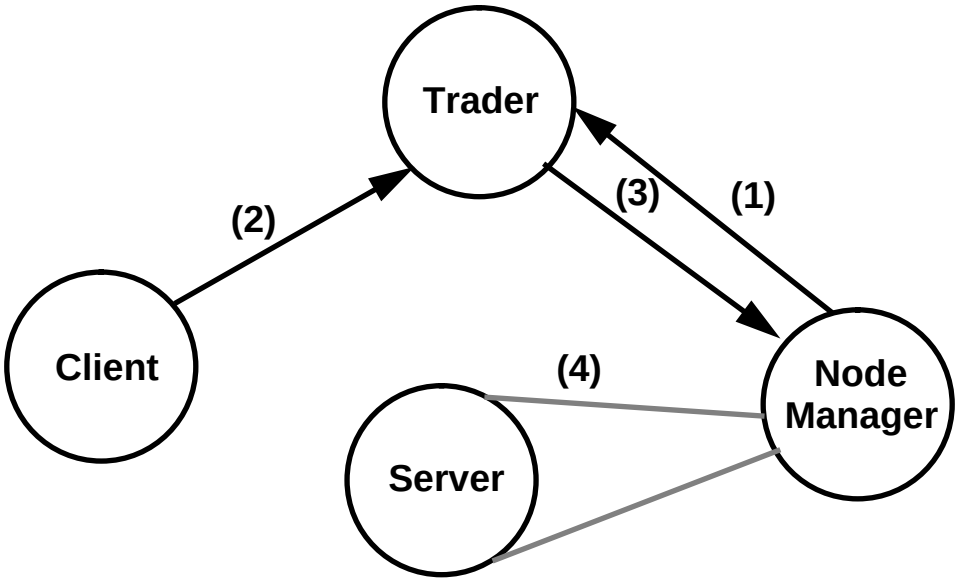
Node Manager - Registering the proxy offer



1. Node Manager registers a proxy offer with the trader

2. Client performs an import

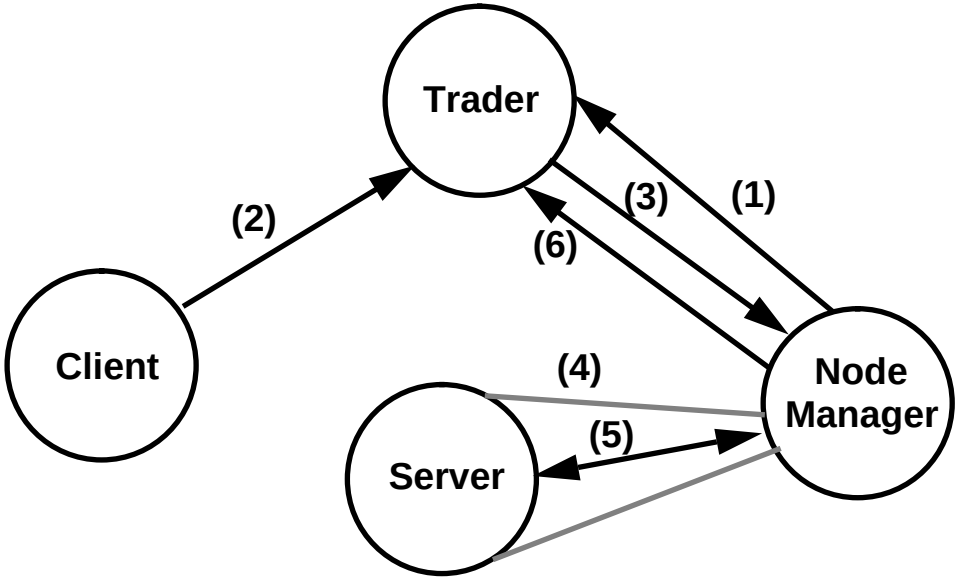
Node Manager - Creating the server capsule



3. Trader, recognising that the offer is federated, forwards the import to the Node Manager

4. Node Manager creates the server capsule (using the factory)

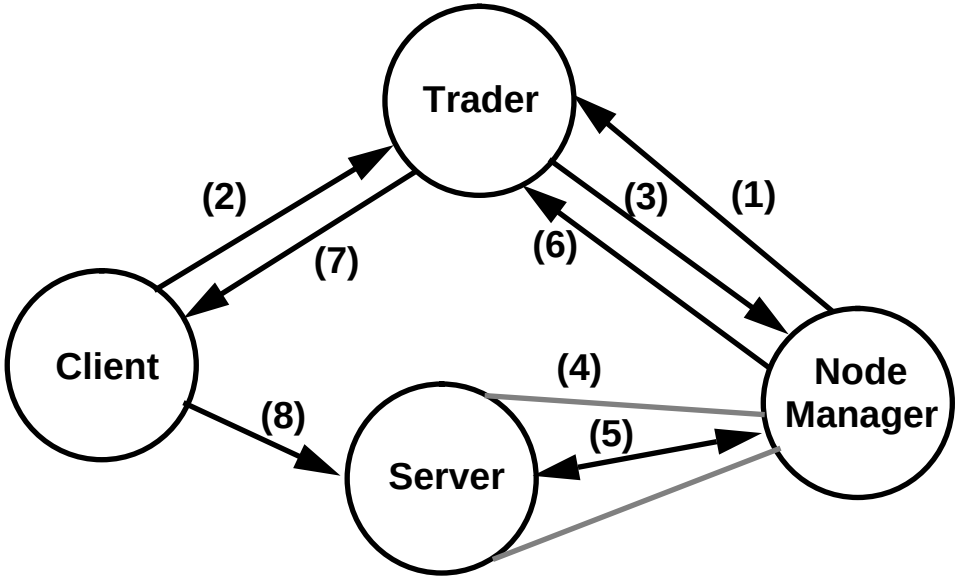
Node Manager - Instantiating the capsule



5. Node Manager invokes the Instantiate operation on the newly created capsule's Capsule interface

6. Node Manager returns the interface (result of Instantiate operation) to the trader

Node Manager - Return of interface



7. **Trader returns this interface to the original client**

8. **Client can now invoke operations on the server**

Node Manager client tool (nmclient)

- `nmclient` is a simple command-line interface to the Node Manager
- To register a proxy offer, first install an alias, then post the offer...

```
nmclient install alias max_activations
                    interface-type context properties
                    capsule object arguments environment
nmclient postproxy interface-type
```

- ...for example:

```
nmclient install “MyEcho” 1 “Echo” "/ansa/testservices” \
                    “” “server” “Echo” “” “”
nmclient postproxy Echo
```


nmclient - Listing aliases

- **To list all of the Node Manager's aliases**

```
nmclient listall
```

- **To list a single alias**

```
nmclient showalias AliasName
```


nmclient - Using constraints

- You can use Trader property constraints to select a particular Node Manager
- By default, `nmclient` uses the Node Manager on your same node
  - but you can specify constraints...

```
nmclient -p “Node == ’crippen’” listall
```

  - ....any ordinary property constraint can be specified
- To avoid confusion, it is safest to specify such a node constraint

nmclient - Activating and deactivating an alias

- To activate an alias:

```
nmclient run alias arguments environment
```

  - the activation_id for the new activation is printed out

- To deactivate an alias:

```
nmclient kill alias activation_id
```

- There may be more than one activation of an alias

  - the activation_id distinguishes between them

nmclient - Withdrawing proxy offers

- **Proxy offers posted via...**
  `nmclient postproxy Echo`
- **...can be withdrawn by:**
  `nmclient deleteproxy alias`
- **Node Manager proxy offers cannot be removed from the trader via `trclient delete`**
  - **if necessary, can use `trclient proxydelete`**
  - **note that `trclient proxydelete` can only delete proxy offers.**
- **Just as with `trclient delete`, this should not normally need to be used**
  - **the Node Manager should correctly manage the state of posted proxy offers**

nmclient - Search paths when activating

- `nmclient` will pass your path to the Node Manager
- The Node Manager will pass this path to the Factory
- The Factory then searches for the executable file (template):
  - first, in this path
  - then, if not found, in the default directory, namely:

    `<ANSAware4-path>/install/<platform>/etc/templates`
- The executable file must be in your search path or in this default directory

Exercise - 1 (Managed Service)

- **Make Simple Bank a managed service. You will need to add the following:**
  - `!MANAGED SBank`
  - `Create_SBank_Object()`
  - `Destroy_SBank_Object()`
- **Test out the service by using `frun` to make sure it works with the Factory**
  - **use the Simple Bank client in the usual way**

Exercise - 2 (Via Node Manager)

- Now, try using the managed service via the Node Manager
  - use `nmclient` to install an alias for your service with the Node Manager
  - post a proxy offer so it will be run automatically when needed
- Test new server activation by using `nmclient run Alias`
  - again, use the Simple Bank client in the usual way
- Remember that you can deactivate a service given the proxy offer activation id

```
nmclient showactive Alias
nmclient kill Alias activation-id
```

Summary

- Factories are used to create capsules
- Capsules create objects and interfaces
- Node Managers use factories to create capsules on demand
- Servers need minor change to be startable from a factory
  - mainly to handle object state
- Clients need no change - even if the server is started via the Node Manger
  - a proxy offer is same as a normal offer, as far as a client is concerned

For more information

- **Applications can use the Factory directly to instantiate capsules**
  - **via the Factory and Capsule interfaces; these are ordinary interfaces specified in ANSAware IDL**
- **Such applications are also responsible for terminating objects and capsules**
  - **using the Factory and Object interfaces**
- **This is how the `frun` and `fkill` commands work**
  - **but you can integrate these facilities into your own management applications...**
  - **...giving finer control over services**
- **For details of these interfaces, see *Application Programming in ANSAware***

nmclient options - 1


```
[-p properties] install alias max_activations interface
      context properties capsule object arguments environment

[-p properties] remove alias

[-p properties] mask alias

[-p properties] postproxy alias

[-p properties] deleteproxy alias

[-p properties] run alias args env

[-p properties] runforever alias args env

[-p properties] kill alias id
```

nmclient options - 2


```
[-p properties] showalias alias

[-p properties] showactive alias

[-p properties] showid alias

[-p properties] listall

[-p properties] listactive

[-p properties] allaliases

[-p properties] allactive
```