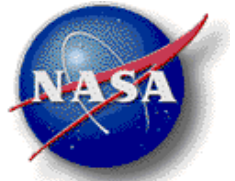


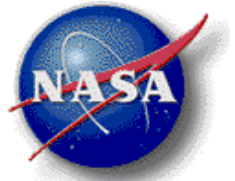
Collaborative Infrastructure Tutorial C++

Ron van Hoof
version 1.1
22 April 2008

Agenda

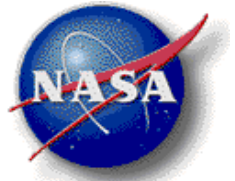


- Actor
- Actor Descriptor
- Actor Implementation
- Directory Service
- Transport Service
- Data Distribution Service
- Translation Service
- Management Service
- Time Service
- Logging Service
- Actor Hosting Environment
- Actor Service Provider
- Process Manager



- Interface to a component
- Controls component state
- Provides component status
- Access to services:
 - to publish itself and lookup other actors
 - to communicate with other actors
 - to publish data
 - to obtain time
 - to log data
- Two parts:
 - Actor Descriptor
 - Actor Implementation

Actor Descriptor

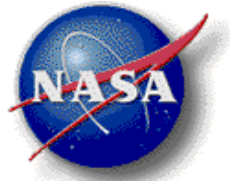


- Describes the actor, defines the actor's configuration settings and is used to load and configure the actor

```
<?xml version="1.0" encoding="UTF-8"?>  
<ACTOR>  
</ACTOR>
```

- Content consists of
 - Library
 - Credentials
 - Advertisement
 - Transport
 - Properties
- File extension: **.ciax**
- Contents available to Actor in ActorDescriptor struct

Actor Descriptor - Library



- Defines where the implementation can be found

- C++ Implementation (library file)

```
<LIBRARY type="ci.actor.library">  
  /usr/lib/libexecutive.so  
</LIBRARY>
```

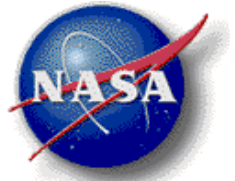
- Simple file name can be specified as well, file must then be in the LD_LIBRARY_PATH (*nix) or in the system path (Windows).

```
<LIBRARY type="ci.actor.library">  
  executive  
</LIBRARY>
```

- Java Implementation (class name)

```
<LIBRARY type="ci.actor.class">  
  gov.nasa.ci.examples.dsa.DecisionSupportAgent  
</LIBRARY>
```

- Class must be in the actor hosting environment's classpath.



- Specifies Actor's identifying information

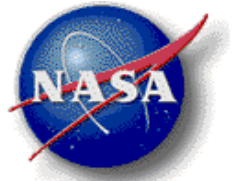
```
<CREDENTIALS>
  <SIMPLENAME>PowerExecutive</SIMPLENAME>

  <QUALIFIEDNAME>gov.nasa.executive.PowerExecutive</QUALIFIEDNAME>

  <DIRECTORYNAMES>
    <DIRECTORYNAME>gov.nasa.executive.PowerExecutive</DIRECTORYNAME>
    <DIRECTORYNAME>A40/executive/PowerExecutive</DIRECTORYNAME>
  </DIRECTORYNAMES>

  <LANGUAGES>
    <LANGUAGE>plexil</LANGUAGE>
  </LANGUAGES>

  <PROPERTIES>
    <PROPERTY name="some.property">some value</PROPERTY>
  </PROPERTIES>
</CREDENTIALS>
```



- Specifies Actor's capabilities

```
<ADVERTISEMENT>
```

```
<CAPABILITIES>
```

```
<CAPABILITY>
```

```
<NAME>Execution</NAME>
```

```
<KEYWORDS>
```

```
<KEYWORD>Execution</KEYWORD>
```

```
<KEYWORD>Power</KEYWORD>
```

```
<KEYWORD>Plan</KEYWORD>
```

```
</KEYWORDS>
```

```
<DESCRIPTION>
```

```
The Power Executive supports the execution of power related plans.
```

```
</DESCRIPTION>
```

```
</CAPABILITY>
```

```
</CAPABILITIES>
```

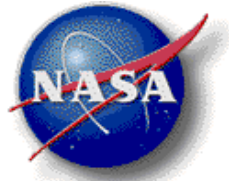
```
<PROPERTIES>
```

```
<PROPERTY name="some.property">some value</PROPERTY>
```

```
</PROPERTIES>
```

```
</ADVERTISEMENT>
```

Actor Descriptor - Transport



Automation for Operations

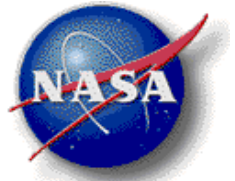
- Specifies how an actor can be reached (dedicated endpoints)
- Preferred approach is to configure only endpoints in the transport service descriptor for the actor service provider (using shared endpoints). Endpoints in the actor's descriptor are dedicated to the actor only.

```
<TRANSPORT>
  <ENDPOINTS>
    <ENDPOINT>
      <PROTOCOL>TCP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="tcp.host">planware</PROPERTY>
        <PROPERTY name="tcp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>

    <ENDPOINT>
      <PROTOCOL>UDP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="udp.host">planware</PROPERTY>
        <PROPERTY name="udp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>
  </ENDPOINTS>

  <PROPERTIES>
  </PROPERTIES>
</TRANSPORT>
```

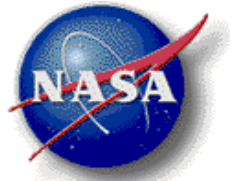
Actor Descriptor - Properties



- Used to define other Actor configuration properties

```
<PROPERTIES>  
  <PROPERTY name="ci.actor.heartbeat.interval">2500</PROPERTY>  
  <PROPERTY name="my.config.file">PowerExecutive.cfg</PROPERTY>  
</PROPERTIES>
```

Actor Descriptor IDL



- Actor Descriptor contents available to the Actor

```
struct ActorDescriptor {
    /** The Actor's Credentials as specified in the descriptor file */
    gov::nasa::ci::corba::api::Credentials ActorCredentials;

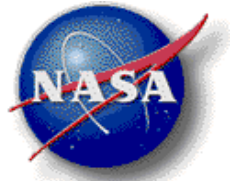
    /** The Actor's Advertisement as specified in the descriptor file */
    gov::nasa::ci::corba::api::directory::Advertisement ActorAdvertisement;

    /** The managed attributes declared for the actor in the descriptor file */
    gov::nasa::ci::corba::api::management::ManagedAttributeList ManagedAttributes;

    /** The managed operations declared for the actor in the descriptor file */
    gov::nasa::ci::corba::api::management::ManagedOperationList ManagedOperations;

    /** Other descriptor properties such as information about the
     * descriptor file itself */
    gov::nasa::ci::corba::api::util::PropertyList Properties;
}; // ActorDescriptor
```

Actor Implementation

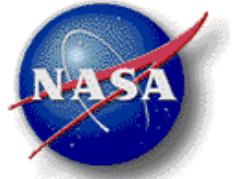


Automation for Operations

- IActor and AbstractActor
- Actor Information Methods
- Actor Service Methods
- Actor Control Methods
- Actor Status Methods
- Actor Message Processing
- Actor State Management
- Packaging
- Building
- Running the Actor

Actor Implementation

Abstract Actor



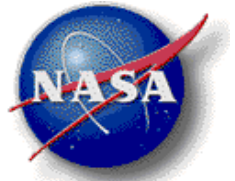
Automation for Operations

- IActor – IDL interface for all actors
- AbstractActor – provides base implementation of all methods except, control, state/status and message processing, extends POA_gov::nasa::ci::corba::api::IActor
- Provides implementation to access Actor information, services, actor CORBA reference
- C++ Implementation

```
#include <ci/api/actor/AbstractActor.h>
using gov::nasa::ci::api::actor::AbstractActor;

class PowerExecutiveActor: public virtual AbstractActor {
}; // PowerExecutiveActor
```

Actor Implementation Information Methods



- AbstractActor provides an implementation to access the actor's configuration information

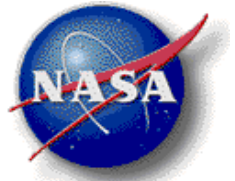
```
virtual ActorDescriptor* getActorDescriptor() throw (CORBA::SystemException);

virtual IActor_ptr getReference() throw (CORBA::SystemException, ActorException);

virtual char* getSimpleName() throw (CORBA::SystemException);
virtual char* getQualifiedName() throws (CORBA::SystemException);
virtual DirectoryNameList* getDirectoryNames() throw (CORBA::SystemException);

virtual Credentials* getCredentials() throw (CORBA::SystemException);
virtual void setCredentials(const Credentials *credentials) throw
    (CORBA::SystemException);

virtual Advertisement* getAdvertisement() throw (CORBA::SystemException);
virtual void setAdvertisement(const Advertisement *advertisement) throw
    (CORBA::SystemException);
```



- AbstractActor provides methods to access the services provided by the CI

```
virtual IDirectoryService_ptr getDirectoryService()  
    throw (CORBA::SystemException, ServiceException);
```

```
virtual ITransportService_ptr getTransportService()  
    throw (CORBA::SystemException, ServiceException);
```

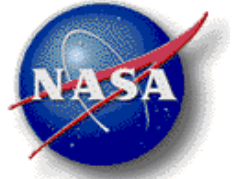
```
virtual IDataDistributionService_ptr getDataDistributionService()  
    throw (CORBA::SystemException, ServiceException);
```

```
virtual IManagementService_ptr getManagementService()  
    throw (CORBA::SystemException, ServiceException);
```

```
virtual ITimeService_ptr getTimeService()  
    throw (CORBA::SystemException, ServiceException);
```

```
virtual ILoggingService_ptr getLoggingService()  
    throw (CORBA::SystemException, ServiceException);
```

Actor Implementation Control Methods



Automation for Operations

- Used to control the state of an actor
- Invoked by the Management Service
- Actor developer responsible for the implementation

```
virtual CORBA::Boolean isInitialized() throw (CORBA::SystemException);
```

```
virtual void initialize() throw (CORBA::SystemException, ActorException);
```

```
virtual void start() throw (CORBA::SystemException, ActorException);
```

```
virtual void suspend() throw (CORBA::SystemException, ActorException);
```

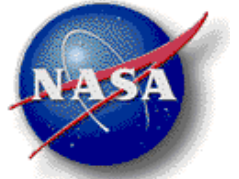
```
virtual void resume() throw (CORBA::SystemException, ActorException);
```

```
virtual void stop() throw (CORBA::SystemException, ActorException);
```

```
virtual void reset(const CheckPoint &cp)  
    throw (CORBA::SystemException, ActorException);
```

```
virtual void shutdown() throw (CORBA::SystemException);
```

Actor Implementation Status Methods



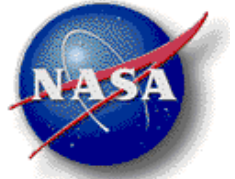
Automation for Operations

- Provides current state information about the actor
- Used by the Management Service to populate the heart beat
- Actor developer responsible for the implementation

```
virtual ActorStatus* getStatus() throw (CORBA::SystemException);
```

```
virtual ActorState getState() throw (CORBA::SystemException);
```

Actor Implementation Message Processing

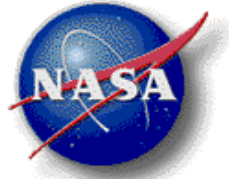


Automation for Operations

- Transport Service notifies the Actor of any messages to be handled by the Actor
- Actor developer responsible for the implementation

```
virtual void processMessage(const CommunicativeAct &message)  
    throw (CORBA::SystemException);
```

Actor Implementation State Management (1/2)



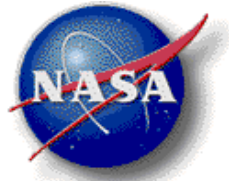
- The ActorState enumeration covers the following states:
 - created, initializing, initialized, starting, idle, waiting, executing, failing, suspending, suspended, resuming, stopping, stopped, resetting, finishing, finished, shutdown
- CI provides a State Manager
 - can optionally be used to manage the state, status and state transitions for an actor.
 - defines a state transition model
 - defines state transition and state related methods
- Method signatures

```
virtual void transitionTo(ActorState state) throw (IllegalStateException);  
virtual void transitionTo(ActorState state, CORBA::LongLong timeout)  
    throws (IllegalStateException);
```

```
virtual ActorState getState();  
virtual ActorState getPreviousState();  
virtual const ActorStatus& getStatus();  
virtual void setCurrentTask(const char* task);
```

```
virtual CORBA::Boolean isInitialized();  
virtual CORBA::Boolean inRunningState();
```

Actor Implementation State Management (2/2)



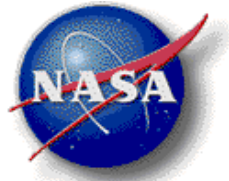
- Example use

```
void PowerExecutiveActor::start()
    throw (CORBA::SystemException, ActorException) {
    try {
        m_oStateManager.transitionTo(gov::nasa::ci::corba::api::STARTING);
    } catch (IllegalStateException &isx) {
        throw ActorException(
            "Invalid state change", IActorExceptionErrorCodes::INVALID_STATE_ERROR, "Invalid state change");
    } // end try

    // start
    try {
        // TODO: start the actor
    } catch (...) {
        try {
            m_oStateManager.transitionTo(gov::nasa::ci::corba::api::FAILED);
        } catch (IllegalStateException &isx) {
            // should never happen since any state can transition to FAILED
        } // end try
        throw ActorException("Unexpected error while starting",
            IActorExceptionErrorCodes::UNEXPECTED_ERROR, "Unexpected error while starting");
    } // end try

    // finalize start
    try {
        m_oStateManager.transitionTo(gov::nasa::ci::corba::api::IDLE);
    } catch (IllegalStateException &isx) {
        throw ActorException(
            "Invalid state change", IActorExceptionErrorCodes::INVALID_STATE_ERROR, "Invalid state change");
    } // end try
} // start
```

Actor Implementation Packaging (1/2)



Automation for Operations

- C++: Actor must be made available in a shared library/dynamically linked library
- Java: Actor's classes must be available on actor hosting environment's class path, preferably packaged as a jar file.
- C++ - PowerExecutive.h

```
#ifndef POWER_EXECUTIVE_H
#define POWER_EXECUTIVE_H

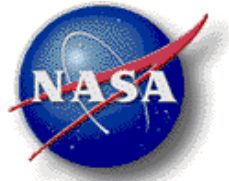
#include "PowerExecutive_export.h"
#include <ci/api/actor/AbstractActor.h>

using gov::nasa::ci::api::actor::AbstractActor;

#ifdef __cplusplus
extern "C" {
#endif

// @return rpActor a pointer to the created AbstractActor or 0 if creation failed
// @return int 0 if the creation was successful, an error code otherwise
PowerExecutive_Export int createActor(AbstractActor*& rpActor);

#ifdef __cplusplus
}
#endif
#endif
```



- C++ - PowerExecutive.cpp

```
#include "ace/ACE.h"
#include "ci/CILogMsg.h"
#include "PowerExecutive.h"
#include "PowerExecutiveActor.h"

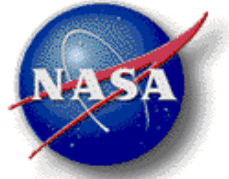
using gov::nasa::a4o::executive::PowerExecutiveActor;

int createActor(AbstractActor*& rpActor) {
    ACE_TRACE(ACE_TEXT("PowerExecutive::createActor\n"));
    ACE_DEBUG((CI_DEBUG
        ACE_TEXT("PowerExecutive::createActor - creating executive.\n")));

    rpActor = new PowerExecutiveActor();

    if (rpActor) {
        ACE_DEBUG((CI_DEBUG
            ACE_TEXT("PowerExecutive::createActor - executive created.\n")));
        return 0;
    } else {
        ACE_ERROR((CI_ERROR
            ACE_TEXT("PowerExecutive::createActor - failed to create executive.\n")));
        return 1;
    } // end if
} // createActor
```

Actor Implementation Building (1/4)



- Use generate_export_file Perl script from ACE/TAO to generate macros that can be included in the source code to allow for platform independent export and import of library classes and functions.

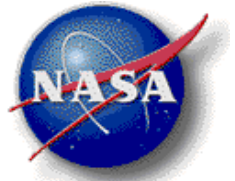
- Perl script (generateExportMacro.pl)

```
eval `(exit $?0)' && eval `exec perl -S $0 ${1+"$@"}`  
& eval `exec perl -S $0 $argv:q`  
if 0;  
$ace_root=$ENV{'ACE_ROOT'};  
system("$ace_root/bin/generate_export_file.pl",  
    "PowerExecutive", ">", "PowerExecutive_export.h");
```

- Example use (PowerExecutive.h)

```
#include "PowerExecutive_export.h"  
PowerExecutive_Export int createActor(AbstractActor*& rpActor);
```

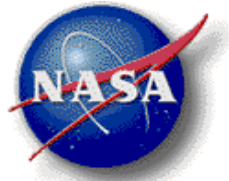
Actor Implementation Building (2/4)



Automation for Operations

- Use Make Project Creator to generate make/project files for various platforms (included with ACE/TAO)
- <name>.mwc to define a workspace
- <name>.mpc to define a project within the workspace
- PowerExecutive.mwc

```
workspace {  
  exclude {  
    bak  
    bin  
    docs  
    lib  
  } // exclude  
} // workspace
```



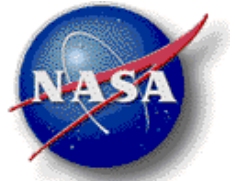
- **executive/PowerExecutive.mpc**

```
project(*IDL): taoidldefaults, anytypecode {
    idlflags += -Wb,export_macro=PowerExecutive_Export -
    Wb,export_include=PowerExecutive_export.h -I$(UE_ROOT)/executive
    IDL_Files {
        Executive.idl
    }
    custom_only = 1
} // project

project: portableserver, valuetype {
    sharedname = *
    after += *IDL
    includes += $(CI_CPP_ROOT)
    libpaths += $(CI_CPP_ROOT)/lib
    libs += CIInterface
    libout = ../lib
    dynamicflags += POWER_EXECUTIVE_BUILD_DLL

    Source_Files {
        PowerExecutive.cpp
    } // Source_Files
    Header_Files {
        PowerExecutive.h
    } // Header_Files
} // project
```

Actor Implementation Building (4/4)



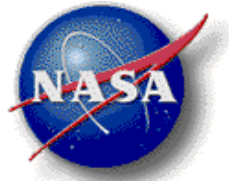
Automation for Operations

- Perl script to generate the workspace, project and make files for various platforms
- GenerateWorkspace.pl

```
eval `(exit $?0)' && eval `exec perl -S $0 ${1+"$@"}'  
  & eval `exec perl -S $0 $argv:q'  
  if 0;  
$ace_root=$ENV{'ACE_ROOT'};  
  
# GNU Makefile  
system("$ace_root/bin/mwc.pl", "-recurse", "-hierarchy");  
  
# Automake  
system("$ace_root/bin/mwc.pl", "-type", "automake", "-noreldefs");  
  
# Visual Studio 6  
system("$ace_root/bin/mwc.pl", "-type", "vc6", "-recurse", "-hierarchy");  
  
# Visual Studio 7.1  
system("$ace_root/bin/mwc.pl", "-type", "vc71", "-recurse", "-hierarchy");  
  
# Visual Studio 8  
system("$ace_root/bin/mwc.pl", "-type", "vc8", "-recurse", "-hierarchy",  
  "-name_modifier", "*_vc8");
```

Actor Implementation

Running the Actor



Automation for Operations

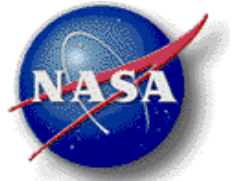
- Deploy the actor by copying its descriptor file (.ciax) into the actor hosting environment's deployment directory.
- Two possibilities to run the actor:
 1. Add the actor to the actor hosting environment's descriptor file in the <ACTORS></ACTORS> tag as:

```
<ACTOR>  
  <DESCRIPTOR>PowerExecutive.ciax</DESCRIPTOR>  
</ACTOR>
```

When the Actor Hosting Environment is started it will automatically load, initialize and start the actor.

2. Use the CI Console to request the actor hosting environment for the list of available actor descriptors. The new actor will be in that list. Request the actor hosting environment to load and initialize the actor. The actor will be listed in the CI Console at which point any of the management operations can be invoked on the actor including the start operation.

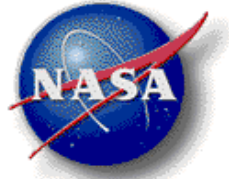
Directory Service



Automation for Operations

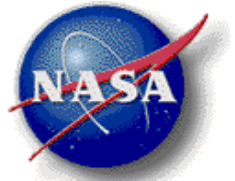
- Overview
- Configuration
- Naming
- Bind
- Rebind
- Lookup
- List Bindings
- Find (FY08)
- Unbind

Directory Service Overview



- IDirectoryService interface with interfaces to:
 - register/deregister an actor using the actor's advertisement, supporting naming hierarchies
 - actor lookup by name or by capability, returning actor advertisements, components will never deal with direct references to actors, the credentials part of the advertisement are used in the envelope of the messages
- Directory service is distributed, each actor hosting environment has its own DS which discovers other DS's on the network and replicates directory entries between the DS's. DS's are discovered using IP multicasting
- Actor lookup by capability (FY08) allows for the support of high-availability features when multiple actors provide the same capabilities, one goes down, another with the same capability can be used.

Directory Service Configuration



Automation for Operations

- Directory Service is an actor configured using a specialized actor descriptor (extension .cidx)

```
<DIRECTORY_SERVICE>  
</DIRECTORY_SERVICE>
```

- Specifies

- Credentials
- Advertisement
- Transport
- Properties

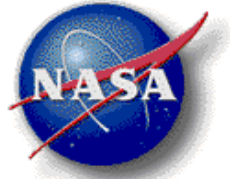
- Two directory specific properties

```
<PROPERTY name="ci.directory.advertisement.publication.interval">5000</PROPERTY>  
<PROPERTY name="ci.directory.subscription.expiration">15</PROPERTY>
```

- Reference to the descriptor is placed in the actor service provider's descriptor file

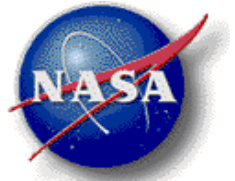
```
<DIRECTORY_SERVICE>  
  <DESCRIPTOR>HEPowerExecutive_DS.cidx</DESCRIPTOR>  
</DIRECTORY_SERVICE>
```

Directory Service Naming



- Uses a similar naming convention as CORBA name service
- Support for compound names (nested contexts)
- Name must not start with '/'
- Contexts are separated using '/'
- No support for the 'kind' attribute
- Examples:
 - gov.nasa.a4o.executive.PowerExecutive
 - nasa/a4o/executive/PowerExecutive
 - nasa/executive/gov.nasa.a4o.executive.PowerExecutive

Directory Service Bind



- Bind is used to bind an advertisement to an unbound name. If an advertisement is already bound to that name an exception is raised
- CI automatically binds actors using the directory names in the actor's descriptor

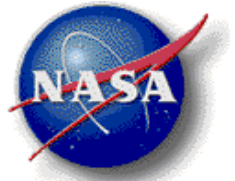
- Method signature:

```
void bind(const char* name, const Advertisement &actor)
    throw (InvalidNameException, NameAlreadyBoundException, NamingException,
          ServiceException, CORBA::SystemException)
```

- Example use:

```
try {
    IDirectoryService_var poDS = getDirectoryService();
    Advertisement_var poAdvertisement = getAdvertisement();
    poDS->bind("a4o/executive/PowerExecutive", poAdvertisement);
} catch (NameAlreadyBoundException &nabx) {
    // a4o/executive/PowerExecutive already bound, use rebind to force binding
} catch (...) {}
```

Directory Service Rebind



- Rebind is used to force the binding of an advertisement to a name. If an advertisement is already bound to that name then that binding is removed and updated with the new binding unless the binding is in a remote directory service.

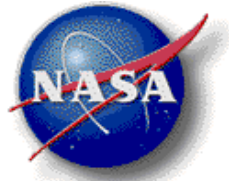
- Method signature:

```
void rebind(const char* name, const Advertisement &actor)
    throw (InvalidNameException, NameAlreadyBoundException, NamingException,
          ServiceException, CORBA::SystemException)
```

- Example use:

```
try {
    IDirectoryService_var poDS = getDirectoryService();
    Advertisement_var poAdvertisement = getAdvertisement();
    poDS->rebind("a4o/executive/PowerExecutive", poAdvertisement);
} catch (NameAlreadyBoundException &nabx) {
    // a4o/executive/PowerExecutive bound in a remote directory
} catch (...) {}
```

Directory Service Lookup



- Lookup is used to retrieve an Actor's advertisement using the directory name of an actor. Lookup performs lookup in both the local directory service and proxies of the discovered remote directory services.

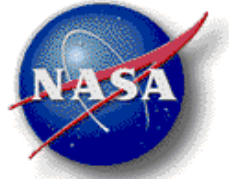
- Method signature:

```
Advertisement* lookup(const char* name)
    throw (InvalidNameException, NameNotFoundException, NamingException,
          ServiceException, CORBA::SystemException)
```

- Example use:

```
try {
    IDirectoryService_var poDS = getDirectoryService();
    Advertisement_var poAdvertisement =
        poDS->lookup("a4o/executive/PowerExecutive");
} catch (NameNotFoundException &nnfx) {
    // a4o/executive/PowerExecutive not bound
} catch (...) {}
```

Directory Service List Bindings



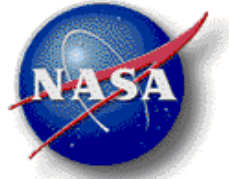
- List Bindings is used to retrieve a list of Actor advertisement bound in a naming context. Lookup performs lookup in both the local directory service and proxies of the discovered remote directory services.
- Method signature:

```
AdvertisementList* listBindings(const char* name)
    throw (InvalidNameException, NameNotFoundException, NamingException,
          ServiceException, CORBA::SystemException)
```

- Example use:

```
try {
    IDirectoryService_var poDS = getDirectoryService();
    AdvertisementList_var poAdvertisements =
        poDS->listBindings("a4o/executive");
    CORBA::String_var psSimpleName = getSimpleName();
    ACE_DEBUG((CI_DEBUG
              ACE_TEXT("%s::listBindings - found %d bindings.\n"),
              psSimpleName, poAdvertisements->length()));
} catch (NameNotFoundException &nnfx) {
    // no bindings listed in the naming context a4o/executive
} catch (...) {}
```

Directory Service Unbind



- Unbind is used to unbind an advertisement from a bound name. If no advertisement is bound to that name the method silently succeeds
- CI automatically unbinds actors bound using the directory names in the actor's descriptor

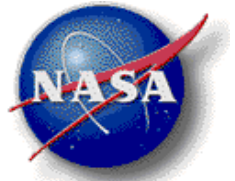
- Method signature:

```
void unbind(const char* name)
    throw (InvalidNameException, NamingException, ServiceException,
          CORBA::SystemException)
```

- Example use:

```
try {
    IDirectoryService_var poDS = getDirectoryService();
    poDS->unbind("a4o/executive/PowerExecutive");
} catch (...) {}
```

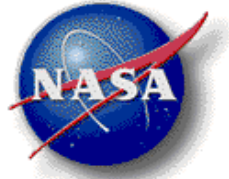
Transport Service



Automation for Operations

- Overview
- Configuration
- CommunicativeAct
- CommunicativeAct Factory
- Sending Message Asynchronously
- Sending Message Asynchronously with Response Handler
- Sending Message Synchronously
- Message Processing
- Sending Stream (FY08)
- Quality of Service Properties

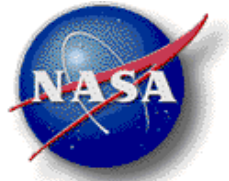
Transport Service Overview



Automation for Operations

- Transport Service provides a message based communication service to communicate with other actors
- Messages are referred to as Communicative Acts and are based on the FIPA specification (Foundation for Intelligent Physical Agents – <http://www.fipa.org>)
- Support both asynchronous and synchronous message delivery
- Hides the transport implementation supporting a pluggable architecture. Actors can publish a variety of endpoints with different protocols declared in the actor's descriptor (TCP, UDP, Multicast, SSL, HTTP, IIOP, etc.)
- Currently supports TCP, SSL, UDP and Multicast endpoints

Transport Service Configuration (1/2)

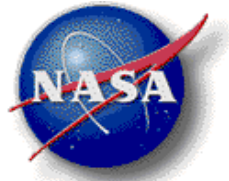


Automation for Operations

- A general transport service descriptor can be used to configure endpoints shared amongst all actors. No longer a need to configure a transport in each actor individually.
- Used for message based transport (Communicative Acts).
- Transport Service is configured using a specialized descriptor (extension .cicx)

```
<TRANSPORT_SERVICE>  
</TRANSPORT_SERVICE>
```
- Specifies
 - Endpoints
 - Properties
- Not used for data distribution service, it has a specialized transport layer and is still configured as part of the data distribution service's descriptor.

Transport Service Configuration (2/2)



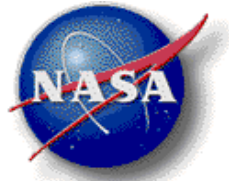
Automation for Operations

```
<TRANSPORT_SERVICE>
  <ENDPOINTS>
    <ENDPOINT>
      <PROTOCOL>TCP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="tcp.host">planware</PROPERTY>
        <PROPERTY name="tcp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>

    <ENDPOINT>
      <PROTOCOL>UDP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="udp.host">planware</PROPERTY>
        <PROPERTY name="udp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>
  </ENDPOINTS>

  <PROPERTIES>
    <!-- many transport related configuration properties
    see the service template for a full listing. -->
  </PROPERTIES>
</TRANSPORT_SERVICE>
```

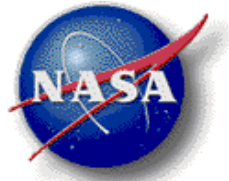
Transport Service Communicative Act (1/3)



Automation for Operations

- Messaging based on FIPA Communicative Act's (<http://www.fipa.org/specs/fipa00037/>)
- Envelope specifies to/from (agent credentials, no direct agent references), QoS parameters, Security parameters, transport hints to allow the transport service to select the most appropriate transport.
- Payload specifies:
 - Performative (inform, request, subscribe, failure, abort, ...)
 - Conversation control (conversation id, reply-to, reply-with, in-reply-to, reply-by)
 - Language (DSA_Plan, PLEXIL, PRL, ...), encoding, ontology
 - Message content (CORBA::Any) – any valid CORBA structures or CORBA primitive data types (long, long long, double, string, ...)

Transport Service Communicative Act (2/3)



Automation for Operations

- Communicative Act IDL

```
struct CommunicativeAct {
    CAEnvelope Envelope;
    CAPayload Payload;
}; // CommunicativeAct

struct CAEnvelope {
    long long CreationDate;
    PropertyList Envelope;
}; // CAEnvelope

struct CAPayload {
    PropertyList Payload;
}; // CAPayload

struct Property {
    string Name;
    any Value;
}; // Property

sequence<Property> PropertyList;
```

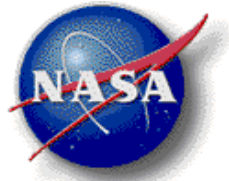
- Abstract interfaces define constants for property names

```
abstract interface ICAEnvelopeProperty {
    const string TO = "to";
};

abstract interface ICAPayloadProperty {
    const string PERFORMATIVE = "performative";
};
```

- Envelope requires **from** and **to** properties. Payload requires **performative** property. Other properties are optional.

Transport Service Communicative Act (3/3)



Automation for Operations

- CI provides interfaces and wrapper classes for the CommunicativeAct structure, envelope and payload with methods to access and set the properties using property specific methods to simplify the use of the CommunicativeAct and properly identifying the expected value types.

```
class ICommunicativeAct {           // implementation class CommunicativeActHelper
public:
    ICommunicativeAct();
    ICommunicativeAct(const CommunicativeAct &ca);
    virtual ~ICommunicativeAct();
    virtual IEnvelope& getEnvelope() = 0;
    virtual void setEnvelope(const IEnvelope &envelope) = 0;
    ...
    virtual const CommunicativeAct& getCommunicativeAct() const = 0;
}; // ICommunicativeAct

class IEnvelope {                   // implementation class EnvelopeHelper
public:
    virtual CredentialsList* getTo() const = 0;
    virtual void setTo(const CredentialsList* recipients) = 0;
    virtual void addTo(const Credentials& recipient) = 0;
    virtual void replaceTo(const Credentials& oldRecipient, const Credentials& newRecipient) = 0;
    virtual void removeTo(const Credentials& recipient) = 0;
    ...
}; // IEnvelope

class IPayload {};                 // implementation class PayloadHelper
```

Transport Service Communicative Act Factory



Automation for Operations

- Factory to simplify creation of Communicative Act's

```
class CommunicativeActFactory {
public:
    static const char* createConversationID(Credentials &creator);

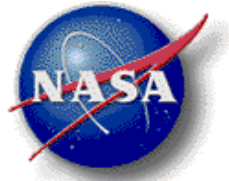
    static ICommunicativeAct* create(
        Credentials &sender, Credentials &receiver, Credentials &replyTo,
        CAPerformative p, const char* language, const char* conversationID,
        const char* action, CORBA::Any &content);
    ...

    static ICommunicativeAct* createRequest(           // performative = REQUEST
        Credentials &sender, Credentials &receiver, Credentials &replyTo,
        const char* language, const char* action, CORBA::Any &content);
    ...

    static ICommunicativeAct* createResponse(
        Credentials &sender, Credentials &receiver, CAPerformative p,
        const char* language, const char* action, CORBA::Any &content,
        ICommunicativeAct &inResponseTo);
    ...
}; // CommunicativeActFactory
```

Transport Service

Sending Message Asynchronously



- Used to send a message for which either no response is expected, to send a response to a message, or for which the response is handled by the actor's generic message handler
- Method signature

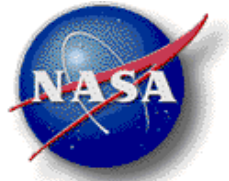
```
virtual void sendMessage(const CommunicativeAct &message)  
    throw (CORBA::SystemException, NotLocatableException, TransportFailure)
```

- Example use

```
Credentials oSender = ...;  
Credentials oReceiver = ...;  
CORBA::Any oContent;  
oContent <<= "Hello World!";  
ICommunicativeAct_var poCA = CommunicativeActFactory.create(oSender, oReceiver,  
    gov::nasa::ci::corba::api::communication::INFORM, oContent);  
try {  
    ITransportService_var poTransportService = getTransportService();  
    poTransportService->sendMessage(poCA->getCommunicativeAct());  
} catch (NotLocatableException &nlx) {  
    // receiver not reachable  
} catch (...) {  
    // transmission or CORBA failure  
} // end try
```

Transport Service

Sending Message w/ Handler (1/3)



Automation for Operations

- Used to send a message for which a response is expected and the developer expects one or more responses to the message and wishes to handle them asynchronously.
- Message requires a conversation id (createRequest method generates one automatically)
- Method signatures

```
virtual CORBA::Long sendAsynchronousMessage(  
    const CommunicativeAct &message,  
    IMessageHandler_ptr handler)  
    throw (CORBA::SystemException, NotLocatableException, TransportFailure);
```

- Example use

```
Credentials oSender = ...; // DSA  
Credentials oReceiver = ...; // Executive  
PlanStructure oPS;  
CORBA::Any oContent;  
oContent <<= oPS;  
// create the message  
ICommunicativeAct_var poCA = CommunicativeActFactory.createRequest(  
    oSender, oReceiver,  
    "PlanStructure", // content language  
    "addPlan", // action  
    oContent);  
  
// create the message handler  
MyHandler_impl oHandler();  
IMessageHandler_var poHandlerRef = oHandler._this();  
  
// send the message  
CORBA::Long nID;  
try {  
    ITransportService_var poTransportService = getTransportService();  
    nID = poTransportService->sendAsynchronousMessage(poCA->getCommunicativeAct(), poHandlerRef);  
} catch (NotLocatableException &nlx) {  
    // receiver not reachable  
} catch (...) {  
    // transmission or CORBA failure  
} // end try
```

Transport Service

Sending Message w/ Handler (2/3)



Automation for Operations

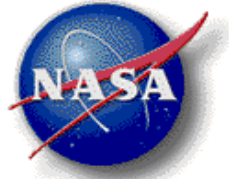
- Example use (cont'd)

```
// create the message handler
class MyHandler_impl: public virtual POA_gov::nasa::ci::corba::api::transport::IMessageHandler {
public:
    MyHandler_impl();
    virtual ~MyHandler_impl();
    virtual void processMessage(const CommunicativeAct &response,
                               const CommunicativeAct &inResponseTo)
        throw (CORBA::SystemException);
    virtual void deactivate();
}; // MyHandler_impl

void MyHandler_impl::processMessage(const CommunicativeAct &response,
                                   const CommunicativeAct &inResponseTo)
    throw (CORBA::SystemException) {
    CommunicativeActHelper oCA(response);
    CAPerformative oPerformative = oCA.getPayload().getPerformative();
    switch (oPerformative) {
    case INFORM: {
        Any oContent = oCA.getPayload().getContent();
        CORBA::Boolean bResult;
        if (oContent >>= CORBA::Any::to_boolean(bResult)) {
            if (bResult) {
                // plan successfully added by the executive
            } else {
                // plan not added by the executive
            } // end if bResult
        } else {
            // invalid response, expected boolean value
        } // end if extract boolean
        break; }
    case FAILURE: ...; break;
    } // end switch
} // processMessage
```

Transport Service

Sending Message w/ Handler (3/3)



Automation for Operations

- Example use (cont'd)

```
void MyHandler_impl::deactivate() {
    try {
        PortableServer::POA_var poPOA(this->_default_POA());
        if (!CORBA::is_nil(poPOA)) {
            PortableServer::ObjectId_var poID(poPOA->servant_to_id(this));
            poPOA->deactivate_object(poID.in());
        } // end if
    }; // MyHandler_impl
```

- Deregister the message handler when no more responses are expected

- Method signature

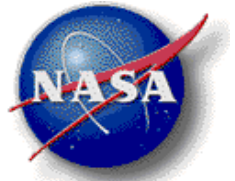
```
virtual void deregisterMessageHandler(CORBA::Long handlerID)
    throw (CORBA::SystemException);
```

- Example use

```
try {
    ITransportService_var poTransportService = getTransportService();
    poTransportService->deregisterMessageHandler(nID);
    oHandler.deactivate(); // deregister from POA
} catch (...) {
} // end try
```

Transport Service

Sending Message Synchronously (1/2)

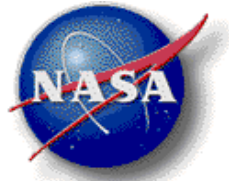


- Used to send a message for which only one response is expected and the developer wishes to block until that response is received or a time-out occurs.
- Message requires a conversation id
- The transport service generates a message handler for the caller
- Method signature

```
virtual CORBA::Long sendSynchronousMessage(  
    const CommunicativeAct &message,  
    CommunicativeAct_out response,  
    CORBA::Long timeout)  
throw (CORBA::SystemException, MessageNotRepliedToException, NotLocatableException,  
    TransportFailure);
```

Transport Service

Sending Message Synchronously (2/2)

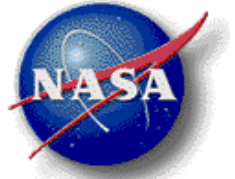


- Example use

```
Credentials oSender = ...; // DSA
Credentials oReceiver = ...; // Executive
PlanStructure oPS;
CORBA::Any oContent;
oContent <<= oPS;
// create the message
ICommunicativeAct_var poCA = CommunicativeActFactory.createRequest(
    oSender, oReceiver,
    "PlanStructure", // content language
    "addPlan", // action
    oContent);

// send the message
try {
    ITransportService_var poTransportService = getTransportService();
    CommunicativeAct_var poResponse;
    poTransportService->sendSynchronousMessage(poCA->getCommunicativeAct(), poResponse, 10000);
    // process the response
} catch (MessageNotRepliedToException &mnrtx) {
    // timed out waiting for a response
} catch (NotLocatableException &nlx) {
    // receiver not reachable
} catch (...) {
    // transmission or CORBA failure
} // end try
```

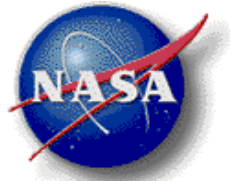
Transport Service Message Processing (1/3)



- The Transport Service sends messages to the actor's message handler if the message is not:
 - a message for the management service
 - a response for which a handler is registered
 - a response to a synchronous message
- and if
 - the qualified name of one of the recipients for the message matches that of the actor
 - the message requires no translation or the message was translated into the language required by the actor
- Used generally to handle requests
- Method signature

```
virtual void processMessage(const CommunicativeAct &message)  
    throw (CORBA::SystemException);
```

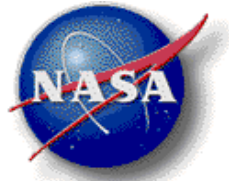
Transport Service Message Processing (2/3)



- Recommended approach is to use a message handler factory that retrieves handlers by performative and action
- Example implementation:

```
void Executive::processMessage(const CommunicativeAct &message)
    throw (CORBA::SystemException) {
    CommunicativeActHelper oCA(message);
    CommunicativeActHandlerFactory *poFactory = CommunicativeActHandlerFactory::Instance(*this);
    ICommunicativeActHandler *poHandler = poFactory->getHandler(oCA);
    if (poHandler != 0) {
        poHandler->process(oCA);
        return;
    } else {
        // make sure we did not receive a NOT_UNDERSTOOD response for one of our messages
        try {
            CPerformative oPerformative = oCA.getPayload().getPerformative();
            if (oPerformative == gov::nasa::ci::corba::api::communication::NOT_UNDERSTOOD) {
                // not-understood response received
                return;
            } // end if
        } catch (NoSuchElementException &nsx) {
            // no performative set in the payload
        } // end try
        // no handler for the type of message, send not understood response back
    }
}
```

Transport Service Message Processing (3/3)



Automation for Operations

- Example implementation (Cont'd)

```
// no handler for the type of message, send not understood response back
ICommunicativeAct_var poResponse = CommunicativeActFactory::createResponse(
    getCredentials(),
    gov::nasa::ci::corba::api::communication::NOT_UNDERSTOOD,
    message);
try {
    ITransportService_var poTransportService = getTransportService();
    poTransportService->sendMessage(poResponse->getCommunicativeAct());
} catch (...) {
} // end try
} // end if
} // processMessage
```

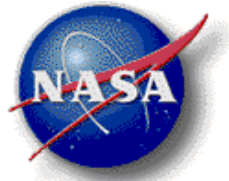
Transport Service Quality of Service Properties (1/2)



Automation for Operations

- The QoS requirements for a message can be specified in its envelope
- CI currently supports only reliability property
 - `"ci.communication.qos.reliability"` or
`gov::nasa::ci::api::communication::IQoSPropertyNames::QOS_RELIABILITY_PROPERTY`
- The reliability can have two possible values
 - “RELIABLE” – attempt to use a reliable transport to transmit the message (TCP/IP based), data delivery guaranteed provided the destination host/port can be reached.
 - “BEST EffORT” – attempt to use a best-effort transport to transmit the message (UDP based), data delivery is not guaranteed
- Default reliability for messages is: RELIABLE
- If the QoS is BEST EffORT and either the receiver publishes no such endpoint or the message delivery fails, the transport service will attempt to use a RELIABLE endpoint for delivery of the message

Transport Service Quality of Service Properties (2/2)



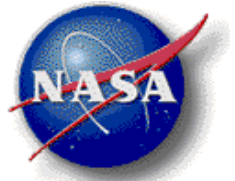
- Example use

```
Credentials_var poCredentials = getCredentials();  
ICommunicativeAct_var poMessage = CommunicativeActFactory::createRequest(  
    poCredentials.in(),  
    oReceiver,  
    "PlanStructure",  
    "addPlan",  
    oContent);
```

```
CORBA::Any oQoSReliabilityValue;  
oQoSReliabilityValue <=<= QoSReliability::BEST_EFFORT.c_str();  
poMessage->getEnvelope().setPropertyValue(  
    IQoSPropertyNames::QOS_RELIABILITY_PROPERTY.c_str(),  
    oQoSReliabilityValue);
```

```
try {  
    ITransportService_var oTransportService = getTransportService();  
    oTransportService.sendMessage(poMessage->getCommunicativeAct());  
} catch (...) {  
} // end try
```

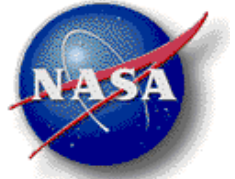
Data Distribution Service



Automation for Operations

- Overview
- Data Info Publication
- Obtain Published Data Info
- Data Subscription
- Data Object
- Data Publication
- Quality of Service Properties
- Data Pull (FY08)

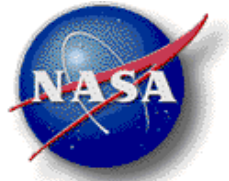
Data Distribution Service Overview



Automation for Operations

- Used to publish data where the sender is not interested in knowing who the subscribers are
- Used to subscribe for data where the origin of the data is not necessarily as important as the data itself
- Used to publish data from one publisher to many subscribers
- Used to subscribe for data of the same type published by many publishers
- Distributed, DDS' discover one another using IP multicasting and periodically publish for each type of data their publishers and subscribers
- DDS publishes only if subscribers are present

Data Distribution Service Configuration (1/2)



Automation for Operations

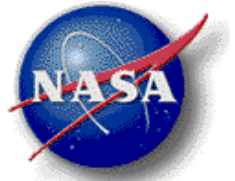
- Data Distribution Service is configured using a specialized actor descriptor (extension **.citx**)

```
<DATA_DISTRIBUTION_SERVICE>  
</DATA_DISTRIBUTION_SERVICE>
```

- Specifies
 - Credentials
 - Advertisement
 - Transport
 - Properties
- Transport requires multicast endpoint for publisher/subscriber discovery, address/port must be the same for all DDS' for every actor hosting environment that needs to communicate with one another

```
<ENDPOINT>  
  <PROTOCOL>MULTICAST</PROTOCOL>  
  <PROPERTIES>  
    <PROPERTY name="multicast.address">235.0.0.1</PROPERTY>  
    <PROPERTY name="multicast.port">5000</PROPERTY>  
    <PROPERTY name="multicast.buffersize">65535</PROPERTY>  
  </PROPERTIES>  
</ENDPOINT>
```

Data Distribution Service Configuration (2/2)



Automation for Operations

- Two data distribution service specific properties

```
<PROPERTY name="ci.dds.advertisement.publication.interval">5000</PROPERTY>
```

```
<PROPERTY name="ci.dds.subscription.expiration">15</PROPERTY>
```

- Reference to the descriptor is placed in the actor service provider's descriptor file

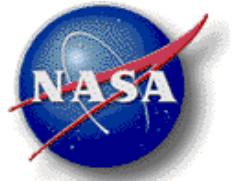
```
<DATA_DISTRIBUTION_SERVICE>
```

```
  <DESCRIPTOR>HEPowerExecutive_DDS.citx</DESCRIPTOR>
```

```
</DATA_DISTRIBUTION_SERVICE>
```

Data Distribution Service

Data Info Publication (1/2)

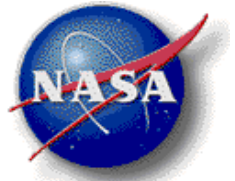


- Actor must indicate the types of data it intends to publish. Type of data is defined as a DataInfo struct (IDL):

```
struct DataInfo {  
    string Domain;  
    string Topic;  
    PropertyList QoSProperties;  
}; // DataInfo
```

- QoSProperties specifies the desired QoS properties for the publisher
- Method Signature

```
CORBA::Long addPublisher(const DataInfo &info)  
    throw (ServiceException, CORBA::SystemException)  
void changePublisher(CORBA::Long publisherID, DataInfo &info)  
    throw (DataTypeChangeException, ServiceException, CORBA::SystemException)  
void removePublisher(CORBA::Long publisherID)  
    throw (ServiceException, CORBA::SystemException)
```



- Example use

```
try {
    IDataDistributionService_var poDDS = getDataDistributionService();
    DataInfo oDataInfo;
    oInfo.Domain = CORBA::string_dup("Executive");
    oInfo.Topic = CORBA::string_dup("ExecutionEvent");

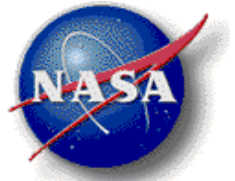
    // register for publication
    CORBA::Long lID = poDDS->addPublisher(oDataInfo);

    // change QoS
    CORBA::Any oQoSReliabilityValue;
    oQoSReliabilityValue <<= QoSReliability::RELIABLE.c_str();
    Property oQoSReliability;
    oQoSReliability.Name = IQoSPropertyNames::QOS_RELIABILITY_PROPERTY.c_str();
    oQoSReliability.Value = oQoSReliabilityValue;
    oDataInfo.QoSProperties.length(1);
    oDataInfo.QoSProperties[0] = oQoSReliability;
    poDDS.changePublisher(lID, oDataInfo);

    // deregister the publication
    poDDS->removePublisher(lID);
} catch (...) {}
```

Data Distribution Service

Obtain Published Data Info



Automation for Operations

- It is possible to request a list of all the types of data published by an actor

- Method Signature

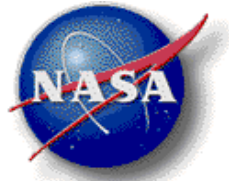
```
DataInfoList* getPublishedDataTypes(const Credentials &actor)  
    throw (ServiceException, CORBA::SystemException)
```

- Example use

```
try {  
    IDataDistributionService_var poDDS = getDataDistributionService();  
    Credentials_var poActor = getCredentials();  
    DataInfoList_var poDataInfoList = poDDS->getPublishedDataTypes(poActor);  
    for (CORBA::ULong i = 0; i < poDataInfoList->length(); i++) {  
        DataInfo oDataInfo = (*poDataInfoList)[i];  
        // do something with oDataInfo  
    } // end for  
} catch (...) {}
```

Data Distribution Service

Data Subscription (1/3)



- To receive data an Actor must subscribe for that data specifying the type of data it wishes to subscribe for. Type of data is defined as a DataInfo struct (IDL):

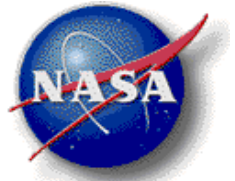
```
struct DataInfo {  
    string Domain;  
    string Topic;  
    PropertyList QoSProperties;  
}; // DataInfo
```

- QoSProperties specifies the desired QoS properties for the subscriber. A subscriber's QoS has precedence over that of the publisher's.
- Method Signature

```
CORBA::Long subscribe(const DataInfo &info, IDataSubscriber_ptr subscriber)  
    throw (ServiceException, CORBA::SystemException)  
CORBA::Long subscribeFrom(const DataInfo &info, const Credentials &actor,  
    IDataSubscriber_ptr subscriber)  
    throw (ServiceException, CORBA::SystemException)  
void unsubscribe(CORBA::Long subscriberID)  
    throw (ServiceException, CORBA::SystemException)
```

Data Distribution Service

Data Subscription (2/3)



Automation for Operations

- Example use

```
try {
    IDataDistributionService_var poDDS = getDataDistributionService();
    DataInfo oDataInfo;
    oInfo.Domain = CORBA::string_dup("Executive");
    oInfo.Topic = CORBA::string_dup("ExecutionEvent");

    // create subscriber
    MySubscriber_impl oSubscriber();
    IDataSubscriber_var poSubscriberRef = oSubscriber._this();

    // subscribe for data
    CORBA::Long lID1 = poDDS->subscribe(oDataInfo, poSubscriberRef);

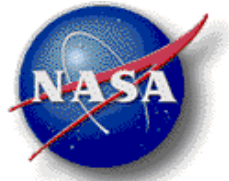
    // subscribe for data from specific actor
    Credentials_var poFrom = getCredentials();
    CORBA::Long lID2 = poDDS->subscribeFrom(oDataInfo, poSubscriberRef, poFrom);

    // unsubscribe for data
    poDDS->unsubscribe(lID2);
    poDDS->unsubscribe(lID1);

    // deactivate the subscriber from the POA before deleting it
    oSubscriber.deactivate();
} catch (...) {}
```

Data Distribution Service

Data Subscription (3/3)



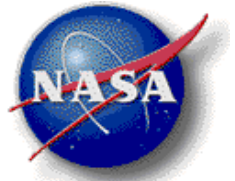
Automation for Operations

- Example use (cont'd)

```
class MySubscriber_impl: public virtual
    POA_gov::nasa::ci::corba::api::data::IDataSubscriber {
public:
    MySubscriber_impl();
    virtual ~MySubscriber_impl();
    virtual void onDataReceipt(const DataObject &data)
        throw (CORBA::SystemException);
    virtual void deactivate();
}; // MySubscriber_impl

void MySubscriber_impl::onDataReceipt(const DataObject &data)
    throw (CORBA::SystemException) {
    // process the DataObject
    CORBA::Any oContent = data.Content;
    const char* sContent;
    if (oContent >>= sContent) {
        // process the XML Executive event
    } else {
        // failed to extract the string contents
    } // end if
} // onDataReceipt
```

Data Distribution Service Data Object



Automation for Operations

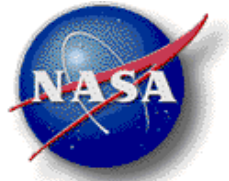
- Data to be published is defined in a DataObject struct (IDL)

```
struct DataObject {  
    string ID;  
    long long Timestamp;  
    string Domain;  
    string Topic;  
    Credentials Sender;  
    PropertyList FilterableData;  
    string Language;  
    any Content;  
}; // DataInfo
```

- The FilterableData can be populated but the DDS currently does not process it.

Data Distribution Service

Data Publication (1/2)



Automation for Operations

- An Actor can publish data at any time provided the actor indicated to the DDS that it published that data (a `PublicationException` is raised if this is not the case)
- Actors publish Data Objects using the DDS.
- The DDS determines if any subscribers have registered for the data, if so it publishes the data to those subscribers, if not the data won't be published.

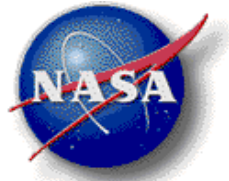
- Method signature

```
void publishData(const DataObject &data)  
    throw (PublicationException, ServiceException, CORBA::SystemException)
```

- Example use

Data Distribution Service

Data Publication (2/2)



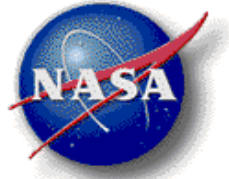
Automation for Operations

- Example use

```
try {
    IDataDistributionService_var poDDS = getDataDistributionService();
    ITimeService_var poTS = getTimeService();
    Credentials_var poSender = getCredentials();
    DataObject oData;
    oData.ID = CORBA::string_dup("Event1");
    oData.Timestamp = poTS.getDateTimeMS();
    oData.Sender = *poSender;
    oData.Domain = CORBA::string_dup("Executive");
    oData.Topic = CORBA::string_dup("ExecutionEvent");
    oData.Language = CORBA::string_dup("plexil");
    oData.Content <=<= "<Event><EventId>1</EventId>...</Event>";

    // publish the data
    poDDS->publishData(oData);
} catch (...) {}
```

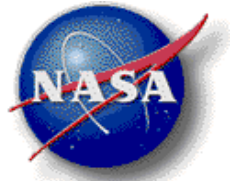
Data Distribution Service Quality of Service Properties



Automation for Operations

- Both the publisher and subscriber can specify quality of service properties for the type of data they publish or subscribe to (DataInfo.QoSProperties)
- CI currently supports only reliability property
`"ci.communication.qos.reliability"` or
`gov::nasa::ci::api::communication::IQoSPropertyNames::QOS_RELIABILITY_PROPERTY`
- The reliability can have two possible values
 - “RELIABLE” – attempt to use a reliable transport to transmit the data (TCP/IP based), data delivery guaranteed provided the destination host/port can be reached.
 - “BEST Effort” – attempt to use a best-effort transport to transmit the data (UDP based), data delivery is not guaranteed
- Default reliability for data is: BEST Effort
- The DDS gives the subscriber’s QoS precedence over the publisher’s.

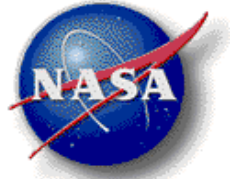
Translation Service



Automation for Operations

- Overview
- Message Content Translation
- Data Content Translation
- Translation Service Configuration
- Registering a Translator
- Writing a Translator

Translation Service Overview

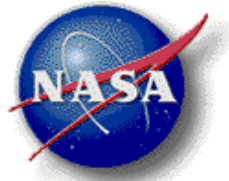


Automation for Operations

- Each actor hosting environment publishes one translation service.
- The translation service allows for automatic content translation of both message and data content
- Uses the language properties of a CommunicativeAct's payload and DataObject with the language specified in an actor's descriptor
- Actor developer does not interact directly with the translation service
- The translation service supports multiple translators
- Translators are described in translation service's descriptor
- Translators are written in Java

Translation Service

Message Content Translation

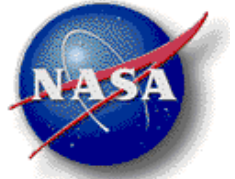


Automation for Operations

- A CommunicativeAct's payload can optionally specify a language (string).
- If a language is specified for the message and the intended recipient of the message specifies no language preference then no translation takes place and the message is delivered as is
- If a language is specified for the message and the intended recipient of the message specifies one or more languages then the message content is translated to the first language for the recipient for which translation succeeds, if none of the translations succeed, the message is not delivered to the recipient
- If no language is specified for the message and the recipient specifies one or more languages then no translation takes place and the message is delivered to the recipient as is. The recipient actor can attempt to interpret the content or return a not_understood response to the sender.

Translation Service

Data Content Translation

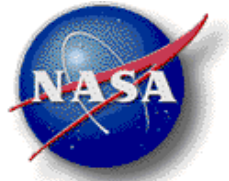


Automation for Operations

- A DataObject can optionally specify a language (string), empty string if no language is specified.
- If a language is specified for the data and the subscriber of the data specifies no language preference then no translation takes place and the data is delivered as is
- If a language is specified for the data and the subscriber of the data specifies one or more languages then the data content is translated to the first language for the subscriber for which translation succeeds, if none of the translations succeed, the data is not delivered to the subscriber
- If no language is specified for the data and the subscriber specifies one or more languages then no translation takes place and the data is delivered to the subscriber as is. The subscriber can attempt to interpret the content or discard the data

Translation Service

Translation Service Configuration



Automation for Operations

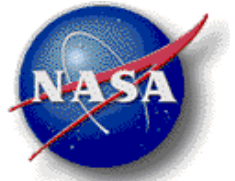
- The translation service descriptor describes the translators made available through the translation service
- The descriptor is placed in the actor service provider's configuration directory
- Extension of the translation service descriptor file is: **.cilx**

```
<?xml version="1.0" encoding="UTF-8"?>  
<TRANSLATION_SERVICE>  
  <TRANSLATORS>  
  </TRANSLATORS>  
</TRANSLATION_SERVICE>
```

- Add a **<TRANSLATION>** tag to the actor service provider's descriptor to identify the translation service descriptor file

```
<TRANSLATION_SERVICE>  
  <DESCRIPTOR>TranslationService.cilx</DESCRIPTOR>  
</TRANSLATION_SERVICE>
```

Translation Service Registering a Translator

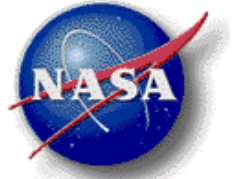


- For each translator add a <TRANSLATOR> tag to the translation service's descriptor.
- Language strings are case insensitive

```
<?xml version="1.0" encoding="UTF-8"?>
<TRANSLATION_SERVICE>
  <TRANSLATORS>
    <TRANSLATOR name="PlanStructurePlexilTranslator"
      class="gov.nasa.ci.examples.translators.
        PlanStructurePlexilTranslator">
      <TRANSLATES from="PlanStructure" to="PLEXIL"/>
      <TRANSLATES from="PLEXIL" to="PlanStructure"/>
    </TRANSLATOR>
  </TRANSLATORS>
</TRANSLATION_SERVICE>
```

Translation Service

Writing a Translator (1/3)



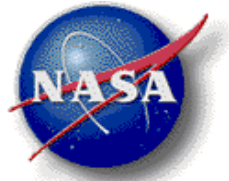
- Translators are developed in Java
- Translator must:
 - implement the `ITranslator` interface or
 - extend the `AbstractTranslator` class
- `AbstractTranslator` adds built-in support for configuring the translator extracting and setting the name and `TranslationSupport` from the translator's descriptor and providing an implementation for the `boolean translates(String from, String to)` method.

- Using `AbstractTranslator` implement:

```
public Any translateMessageContent(Any content, String from, String to)
    throws TranslationException;
public Any translateDataContent(IDataInfo info, Any content, String from, String to)
    throws TranslationException;
```

Translation Service

Writing a Translator (2/3)



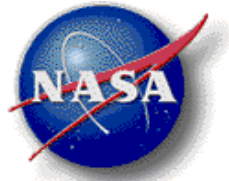
- Example

```
public class PlanStructurePlexilTranslator extends AbstractTranslator {
    public PlanStructurePlexilTranslator() {
        m_moPlanStructureToPlexilMapping = new HashMap<Integer, String>();
        m_moPlexilToPlanStructureMapping = new HashMap<String, Integer>();
    } // PlanStructurePlexilTranslator
    private Map<Integer, String> m_moPlanStructureToPlexilMapping;
    private Map<String, Integer> m_moPlexilToPlanStructureMapping;

    public Any translateMessageContent(Any content, String from, String to)
        throws TranslationException {
        if (translates(from, to)) {
            if (from.equals(DecisionSupportAgent.LANGUAGE.toLowerCase())) {
                // PlanStructure to PLEXIL
                try {
                    ...
                    Any oTranslatedContent = ORBHelper.getORB().create_any();
                    oTranslatedContent.insert_string(sPlexil);
                    return oTranslatedContent;
                } catch (Throwable tx) {
                    throw new NotTranslatableException("Failed to translate message content.", tx);
                } // end try
            } else {
                // PLEXIL to PlanStructure
                ...
            } // end if
        } else {
            throw new TranslationNotSupportedException("Unable to translate message content from "+
                from+" to "+to);
        } // end if
    } // translateMessageContent
}
```

Translation Service

Writing a Translator (3/3)

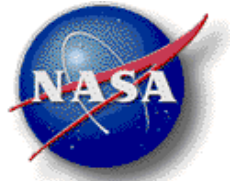


- Example (Cont'd)

```
public Any translateDataContent(IDataInfo info, Any content, String from, String to)
    throws TranslationException {
    if (translates(from, to)) {
        // only support translation from plexil to ExecStatus/PlanStructure
        if (from.equals(UniversalExecutive.LANGUAGE.toLowerCase())) {
            try {
                String sEvent = content.extract_string();
                ExecStatus oStatus = ExecutiveStatusConverter.convertXMLStatus(this, sEvent);
                if (oStatus != null) {
                    Any oTranslatedContent = ORBHelper.getORB().create_any();
                    ExecStatusHelper.insert(oTranslatedContent, oStatus);
                    return oTranslatedContent;
                } else {
                    throw new NotTranslatableException("Failed to translate data content.");
                } // end if
            } catch (Throwable tx) {
                throw new NotTranslatableException("Failed to translate data content.", tx);
            } // end try
        } else {
            throw new TranslationNotSupportedException("Unable to translate message content from "+
                from+" to "+to);
        } // end if
    } else {
        throw new TranslationNotSupportedException("Unable to translate message content from "+
            from+" to "+to);
    } // end if
} // translateMessageContent

} // PlanStructurePlexilTranslator
```

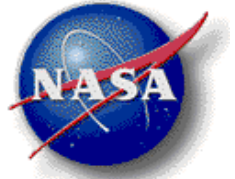
Management Service



Automation for Operations

- Overview
- Heart Beat Publication
- Heart Beat Subscription
- Managed Attributes
- Managed Operations
- Operation Executors

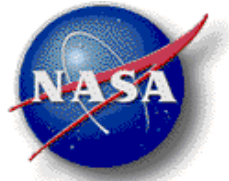
Management Service Overview



Automation for Operations

- Provides remote control and monitoring service for an actor
- Used to control and monitor actor's state
- Used to remotely configure an actor
- Management messages processed by management service not the actor
- CI Console uses the management service to monitor, control and configure actors, actor hosting environments and process managers

Management Service Heart Beat Publication (1/2)



Automation for Operations

- Actor can have the MS periodically publish a heart beat
- Heart beat specified actor's current status
- Heart beat is automatically started if the interval is specified for an actor in its descriptor

```
<PROPERTIES>
```

```
  <PROPERTY name="ci.actor.heartbeat.interval">2500</PROPERTY>
```

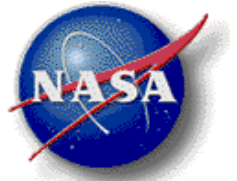
```
</PROPERTIES>
```

- MS used the DDS to publish the heart beat (domain: HeartBeat, topic: actor.qualifiedname)
- Actor can itself control the heart beat publication
- Method signatures

```
virtual void startHeartBeat(CORBA::Long interval) throw (CORBA::SystemException, ServiceException);
```

```
virtual void stopHeartBeat() throw (CORBA::SystemException, ServiceException);
```

Management Service Heart Beat Publication (2/2)

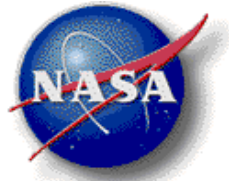


Automation for Operations

- Example use

```
try {  
  IManagementService_var poMS = getManagementService();  
  oMS->startHeartBeat(2500);  
  oMS->stopHeartBeat();  
} catch (...) {  
} // end try
```

Management Service Heart Beat Subscription (1/2)



Automation for Operations

- The MS provides methods to register heart beat listeners used to monitor the status of another actor.
- Actor to be monitored must be identified by its qualified name

- **Method signatures**

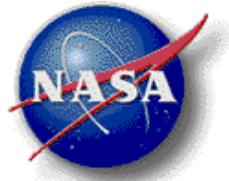
```
virtual void subscribeForHeartBeat(  
    const char* name,  
    IHeartBeatListener_ptr listener) throw (CORBA::SystemException, ServiceException);
```

```
virtual void unsubscribeForHeartBeat(  
    const char* name) throw (CORBA::SystemException, ServiceException);
```

- **Example use**

```
try {  
    IManagementService_var poMS = getManagementService();  
    ExecutiveHBLListener_impl oListener;  
    IHeartBeatListener_var poListenerRef = oListener._this();  
    poMS->subscribeForHeartBeat("gov.nasa.executive.PowerExecutive", poListenerRef);  
    poMS->unsubscribeForHeartBeat("gov.nasa.executive.PowerExecutive");  
    oListener.deactivate();  
} catch (...) {  
} // end try
```

Management Service Heart Beat Subscription (2/2)



Automation for Operations

- Example use (Cont'd)

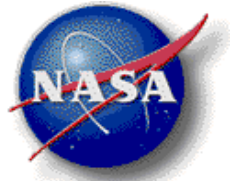
```
class ExecutiveHBLListener_impl: public virtual
    POA_gov::nasa::ci::corba::api::management::IHeartBeatListener {
public:
    ExecutiveHBLListener_impl();
    virtual ~ExecutiveHBLListener_impl();
    virtual void onNewHeartBeat(
        const Credentials &actor,
        const ActorStatus &status)
        throw (CORBA::SystemException);
    virtual void deactivate();
}; // ExecutiveHBLListener_impl

void ExecutiveHBLListener_impl::onNewHeartBeat(
    const Credentials &actor,
    const ActorStatus &status)
    throw (CORBA::SystemException) {
    // process the status
    if (status.Alive) {

    } else {

    } // end if
} // onNewHeartBeat
```

Management Service Managed Attributes (1/3)



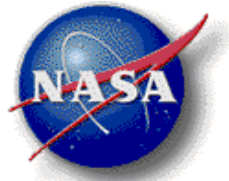
- Managed attributes are used to allow for the remote monitoring and optional control of an actor's configuration.

```
struct ManagedAttribute {  
    string Name;  
    string Type;           // IDL basic type or qualified user defined type  
    boolean IsReadOnly;  
} // ManagedAttribute
```

- Method signatures

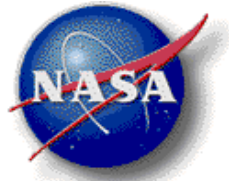
```
virtual CORBA::Boolean addManagedAttribute(const ManagedAttribute &attr)  
    throw (CORBA::SystemException, ServiceException)  
virtual ManagedAttribute* getManagedAttribute(const char* name)  
    throw (CORBA::SystemException, NoSuchAttributeException, ServiceException)  
virtual ManagedAttributeList* getManagedAttributes()  
    throw (CORBA::SystemException, ServiceException)  
virtual void setAttributeValue(const char* name, const CORBA::Any value)  
    throw (CORBA::SystemException, NoSuchAttributeException, ServiceException)  
virtual CORBA::Any* getAttributeValue(const char* name)  
    throw (CORBA::SystemException, NoSuchAttributeException, NoValueException, ServiceException)  
virtual CORBA::Boolean removeManagedAttribute(const char* name)  
    throw (CORBA::SystemException, ServiceException)  
virtual CORBA::Long addManagedAttributeChangeListener(  
    IManagedAttributeChangeListener_ptr listener)  
    throw (CORBA::SystemException, ServiceException)  
virtual void removeManagedAttributeChangeListener(CORBA::Long listenerID)  
    throw (CORBA::SystemException, ServiceException)
```

Management Service Managed Attributes (2/3)



- Example use

```
try {
  IManagementService_var poMS = getManagementService();
  // add a managed attribute
  ManagedAttribute oAttribute;
  oAttribute.Name = CORBA::string_dup("enabled");
  oAttribute.Type = CORBA::string_dup("boolean");
  oAttribute.IsReadOnly = false;
  oMS->addManagedAttribute(oAttribute);
  // set initial value
  CORBA::Any oEnabled;
  oEnabled <=< CORBA::Any::from_boolean(true);
  oMS->setAttributeValue("enabled", oEnabled);
  // register a listener to be notified of any value changes made to the
  // attribute's value by a remote console/actor
  MyChangeListener_impl oChangeListener;
  IManagedAttributeChangeListener_var poChangeListenerRef = oChangeListener._this();
  CORBA::Long nID = poMS->addManagedAttributeChangeListener(poChangeListenerRef);
  poMS->removeManagedAttributeChangeListener(nID);
  poMS->removeManagedAttribute("enabled");
  oChangeListener.deactivate();
} catch (...) {}
```

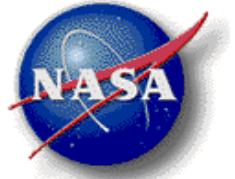


- Example use (Cont'd)

```
class MyChangeListener_impl: public virtual
    POA_gov::nasa::ci::corba::api::management::IManagedAttributeChangeListener {
public:
    MyChangeListener_impl();
    virtual ~MyChangeListener_impl();
    virtual void onAttributeChange(
        const ManagedAttributeChangeEvent &evt)
        throw (CORBA::SystemException);
    virtual void deactivate();
}; // MyChangeListener_impl

void MyChangeListener_impl::onAttributeChange(
    const ManagedAttributeChangeEvent &evt)
    throw (CORBA::SystemException) {
    // process the event
    if (strcmp(evt.Name, "enabled") {
        Credentials oSource = evt.Source;
        CORBA::Any oOldValue = evt.OldValue;
        CORBA::Any oNewValue = evt.NewValue;
        CORBA::Boolean bEnabled;
        if (oNewValue >>= CORBA::Any::to_boolean(bEnabled)) {
            // enable or disable the actor
        } // end if
    } // end if
} // onAttributeChange
```

Management Service Managed Operations (1/3)



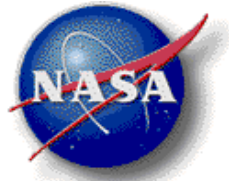
Automation for Operations

- Managed Operations are used to publish operations that can be invoked to control or configure the actor.
- Execution of managed operations is performed by the management service (not the actor) invoking the registered operation executor for the operation.

```
struct ManagedOperation {
    string Name;
    ManagedOperationParameterList Parameters;
    string ReturnType;          // IDL basic type or qualified user defined type or "void"
} // ManagedOperation
typedef sequence<ManagedOperationParameter> ManagedOperationParameterList;
struct ManagedOperationParameter {
    string Name;
    string Type;              // IDL basic type or qualified user defined type
} // ManagedOperationParameter
```

- Each actor has a fixed set of managed operations and operation executors registered by the CI: start, suspend, resume, stop, reset(CheckPoint), shutdown, getStatus, getState

Management Service Managed Operations (2/3)



Automation for Operations

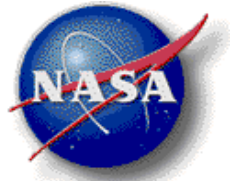
- Method signatures

```
virtual CORBA::Boolean addManagedOperation(const ManagedOperation &operation)
    throw (CORBA::SystemException, ServiceException)
virtual ManagedOperation* getManagedOperation(const char* name)
    throw (CORBA::SystemException, NoSuchOperationException, ServiceException)
virtual ManagedOperationList* getManagedOperations()
    throw (CORBA::SystemException, ServiceException)
virtual CORBA::Boolean removeManagedOperation(const char* name)
    throw (CORBA::SystemException, ServiceException)
```

- Example use

```
try {
    IManagementService_var poMS = getManagementService();
    // create the operation
    ManagedOperation oOperation;
    oOperation.Name = CORBA::string_dup("reset");
    ManagedOperationParameterList oParameters;
    oParameters.length(1);
    oParameter[0].Name = CORBA::string_dup("cp");
    oParameter[0].Type = CORBA::string_dup("gov::nasa::ci::corba::api::Checkpoint");
    oOperation.ReturnType = CORBA::string_dup("void");
```

Management Service Managed Operations (3/3)

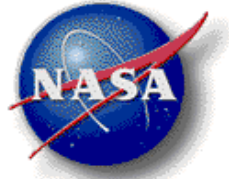


Automation for Operations

- Example use (Cont'd)

```
// register the operation  
poMS->addManagedOperation(oOperation);  
  
poMS->removeManagedOperation("reset");  
} catch (...) {}
```

Management Service Operation Executors (1/3)

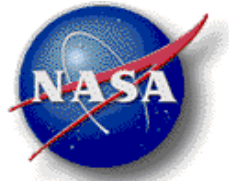


Automation for Operations

- Operation executors provide the implementation for the registered operations
- Operation executors are invoked by the management service when a request for the operation is received
- Method signatures

```
virtual void setOperationExecutor(  
    const char* name, IManagedOperationExecutor_ptr executor)  
    throw (CORBA::SystemException, NoSuchOperationException, ServiceException)  
virtual IManagedOperationExecutor_ptr getOperationExecutor(const char* name)  
    throw (CORBA::SystemException, NoSuchOperationException, NoSuchExecutorException,  
    ServiceException)  
virtual void removeOperationExecutor(const char* name)  
    throw (CORBA::SystemException, ServiceException)
```

Management Service Operation Executors (2/3)



Automation for Operations

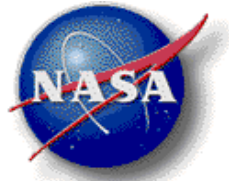
- Example use

```
try {
    IManagementService_var poMS = getManagementService();

    // create the executor
    ResetExecutor_impl oExecutor;
    IManagedOperationExecutor_var poExecutorRef = oExecutor._this();
    poMS->setOperationExecutor("reset", poExecutorRef);

    poMS->removeOperationExecutor("reset");
    oExecutor.deactivate();
} catch (...) {}

class ResetExecutor_impl: public virtual
    POA_gov::nasa::ci::corba::api::management::IManagedOperationExecutor {
public:
    ResetExecutor_impl();
    virtual ~ResetExecutor_impl();
    virtual void processOperation(
        const char* name,
        const PropertyList &arguments,
        CORBA::Any_out result)
        throw (CORBA::SystemException,
            BadArgumentException, MissingArgumentException, InvocationException);
    virtual void deactivate();
}; // ResetExecutor_impl
```



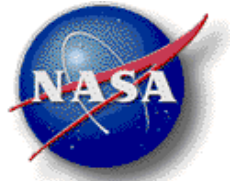
- Example use (Cont'd)

```
virtual void ResetExecutor_impl::processOperation(
    const char* name,
    const PropertyList &arguments,
    CORBA::Any_out result)
    throw (CORBA::SystemException,
          BadArgumentException, MissingArgumentException, InvocationException) {
    // make sure it is the correct operation
    if (strcmp(name, "reset") != 0) {
        CORBA::Any oError;
        oError <<= "Invalid operation name";
        throw InvocationException("Invalid operation name",
            IManagementServiceErrorCodes::INVALID_OPERATION_ERROR,
            oError);
    } // end if

    // get the arguments for the operation
    CheckPoint *poCP;
    CORBA::Any *poValue = PropertyListHelper::getPropertyValue(arguments, "cp");
    if (poValue != 0) {
        if (!((*poCPValue) >= poCP)) {
            delete poValue;
            throw BadArgumentException("Argument value for parameter 'cp' is not of type CheckPoint.", "cp");
        } // end if
    } else {
        delete poValue;
        throw MissingArgumentException("Missing required 'CheckPoint cp' argument", "cp");
    } // end if

    // invoke the operation
    getActor().reset(*poCP);
    delete poValue;
} // processOperation
```

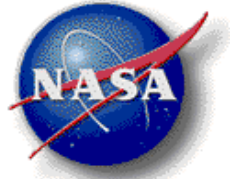
Time Service



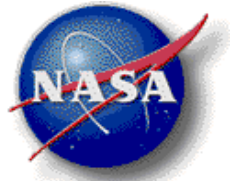
Automation for Operations

- Overview
- Time services

Time Service Overview



- Provides time services as were provided by the CI in FY06 except for simulation time support
- One Time Service per actor hosting environment to prevent a single point of failure
- Time synchronization currently requires installation and configuration of NTP (Network Time Protocol) on all operating systems used for A4O



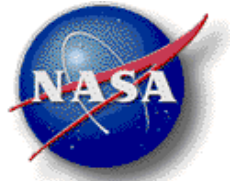
- **Method signatures**

```
virtual CORBA::LongLong getDateMS() throw (CORBA::SystemException);  
virtual DateTime getDate() throw (CORBA::SystemException);  
virtual CORBA::Long notifyAtInterval(  
    ITimeServiceSubscribr_ptr subscriber,  
    CORBA::Long interval) throw (CORBA::SystemException);  
virtual unsubscribe(CORBA::Long subscriberID) throw (CORBA::SystemException);
```

- **Example use**

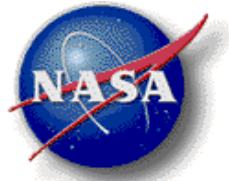
```
try {  
    ITimeService_var poTS = getTimeService();  
    DataObject oDataObject;  
    oDataObject.Timestamp = poTS->getDateMS();  
} catch (...) {}
```

Logging Service



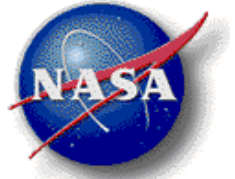
- Overview
- Logging Methods

Logging Service Overview



- Process Manager and Actor Hosting Environment use Java's Log4J to log messages. Log4J is highly configurable using a configuration file (property or XML based) located in the same directory as the actor hosting environment/process manager's descriptor file with the same name followed by `Logger.cfg` or `Logger.xml`
- Logging Service logs messages using Log4J
- Logging Service generally only needed for C++ actors. C++ actors however can opt to use the ACE logging service since using the Logging Service is not efficient yet. Actor hosting environment's descriptor will have an option to specify the ACE `svc.cfg` file to use to configure the ACE logging service.

Logging Service Logging Methods



- Method signatures

```
virtual void debug(const char* name, const char* message)
    throw (CORBA::SystemException);
```

```
virtual void info(const char* name, const char* message)
    throw (CORBA::SystemException);
```

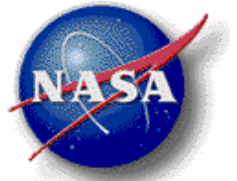
```
virtual void warn(const char* name, const char* message)
    throw (CORBA::SystemException);
```

```
virtual void error(const char* name, const char* message)
    throw (CORBA::SystemException);
```

- Example use

```
try {
    ILoggingService_var oLS = getLoggingService();
    oLS->info("gov.nasa.ci.executive.PowerExecutive",
            "Starting actor.");
} catch (...) {}
```

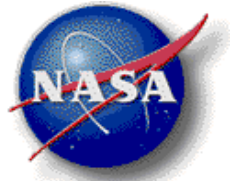
Actor Hosting Environment



Automation for Operations

- Overview
- Configuration
- Actor Deployment

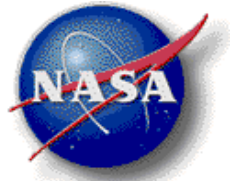
Actor Hosting Environment Overview



Automation for Operations

- Hosts one or more actors (C++ and/or Java)
- Provides core services to hosted actors
 - One shared directory service
 - One shared data distribution service
 - One shared translation service
 - One shared time service
 - One shared logging service
 - One transport service per actor
 - One data distribution service interface per actor
 - One management service per actor
- Manages the life cycle of an actor
- Is itself an actor

Actor Hosting Environment Configuration (1/2)

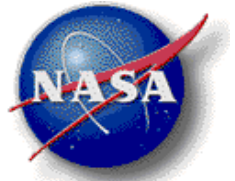


Automation for Operations

- Configured using a specialized actor descriptor (extension **.cihx**)

```
<HOSTING_ENVIRONMENT>  
</HOSTING_ENVIRONMENT>
```

- Specifies
 - Credentials
 - Advertisement
 - Transport (optional)
 - *Actor Service Provider*
 - *Actors*
 - *Properties*



- Hosting environment specific sections

```
<ASP>
```

```
  <DESCRIPTOR>heasp/ASP.ciaspx</DESCRIPTOR>
```

```
</DIRECTORY>
```

```
<ACTORS>
```

```
  <ACTOR>
```

```
    <DESCRIPTOR>PowerExecutive.ciax</DESCRIPTOR>
```

```
  </ACTOR>
```

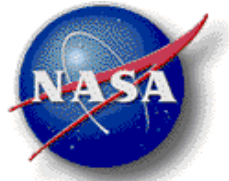
```
</ACTORS>
```

- One hosting environment specific property

```
<PROPERTY
```

```
  name="ci.ahe.deploydir">/usr/ahe/powerexec/deploy</PROPERTY>
```

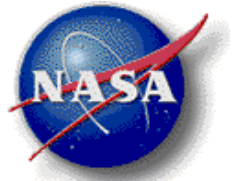
Actor Hosting Environment Actor Deployment



Automation for Operations

- Actor Service Provider descriptor must be located in the hosting environment's configuration directory or a directory relative to it.
- Actor descriptor files must be located in the hosting environment's deployment directory
- Hosting environment automatically loads, initializes and starts actors defined in the hosting environment's descriptor.
- Through the hosting environment management service it is possible to obtain a list of all actor's descriptors in the deployment directory. The hosting environment can be requested to load and initialize actors from the descriptors.
- Through the actor's management service actors can be started.
- Through the hosting environment management service actors can also be unloaded.

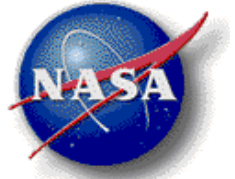
Actor Service Provider



Automation for Operations

- Overview
- Configuration

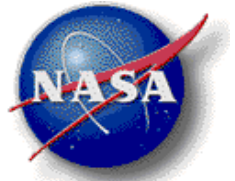
Actor Service Provider Overview



Automation for Operations

- Manages all CI services (transport, directory, data distribution, translation, time, logging)
- Manages actors
- Provides interface to CI services to registered actors

Actor Service Provider Configuration (1/2)



Automation for Operations

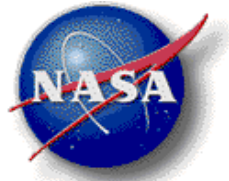
- Configured using a specialized actor descriptor (extension **.ciaspx**)

```
<ASP>
```

```
</ASP>
```

- Specifies
 - Transport Service (Required)
 - Directory Service (Optional)
 - Data Distribution Service (Optional)
 - Translation Service (Optional)
 - Properties

Actor Service Provider Configuration (2/2)



Automation for Operations

- **Service descriptor references**

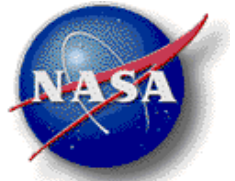
```
<TRANSPORT_SERVICE>
  <DESCRIPTOR>TransportService.cicx</DESCRIPTOR>
</TRANSPORT_SERVICE>
<DIRECTORY_SERVICE>
  <DESCRIPTOR>DirectoryService.cidx</DESCRIPTOR>
</DIRECTORY_SERVICE>
<DATA_DISTRIBUTION_SERVICE>
  <DESCRIPTOR>DataDistributionService.cidx</DESCRIPTOR>
</DATA_DISTRIBUTION_SERVICE>
<TRANSLATION_SERVICE>
  <DESCRIPTOR>TranslationService.cidx</DESCRIPTOR>
</TRANSLATION_SERVICE>
```

- **One property**

```
<PROPERTY name="ci.asp.dir.config">.</PROPERTY>
```

Specifies the directory in which to find the service descriptor files. '.' indicates the same directory as the ASP's descriptor.

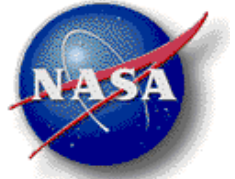
Process Manager



Automation for Operations

- Overview
- Configuration
- Process Descriptors

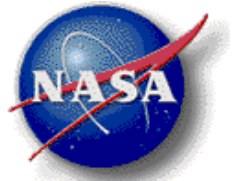
Process Manager Overview



Automation for Operations

- One Process Manager per Operating System
- Preferably started as a service/daemon
- Used to start/stop/monitor actor hosting environment processes
- Processes are defined using process descriptors
- Is itself a special actor not requiring a hosting environment with built-in transport, data distribution and management services.

Process Manager Configuration (1/2)

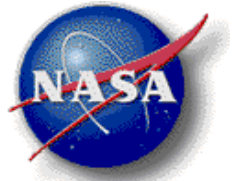


Automation for Operations

- Configured using a specialized actor descriptor (extension **.cimx**)

```
<PROCESS_MANAGER>  
</PROCESS_MANAGER>
```

- Specifies
 - Credentials
 - Advertisement
 - Transport (optional)
 - *Actor Service Provider*
 - *Processes*
 - *Properties*



- Process manager specific sections

```
<ASP>
```

```
  <DESCRIPTOR>pmasp/ASP.ciaspx</DESCRIPTOR>
```

```
</ASP>
```

```
<PROCESSES>
```

```
  <PROCESS>
```

```
    <DESCRIPTOR>HEPowerExecutiveProcess.cipx</DESCRIPTOR>
```

```
  </PROCESS>
```

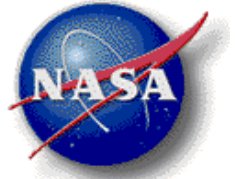
```
</PROCESSES>
```

- One process manager specific property

```
<PROPERTY name="ci.pm.deploydir">/usr/pm/deploy</PROPERTY>
```

Process Manager

Process Descriptor (1/4)

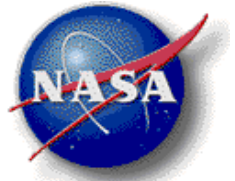


Automation for Operations

- For each process that should be startable by the process manager a process descriptor must be defined and placed in the process manager's deployment directory.
- Process descriptor's file extension is **.cipx**

```
<PROCESS id="gov.nasa.ci.HEPowerExecutive" name="HEPowerExecutive">  
</PROCESS>
```
- Specifies:
 - Process description
 - Environment variables
 - Program to start
 - Working directory
 - Optional Java specific settings for Java applications
 - boot classpath
 - extension classpath
 - classpath
 - VM options
 - Main class
 - Program arguments

Process Manager Process Descriptor (2/4)



- Example

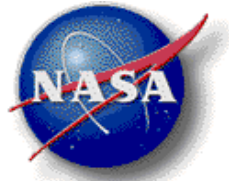
```
<PROCESS id="gov.nasa.ci.HEPowerExecutive" name="HEPowerExecutive">
  <DESCRIPTION>
    Starts the Actor Hosting Environment running the Power Universal Executive.
  </DESCRIPTION>

  <ENVIRONMENT>
    <SET name="JAVA_HOME">C:/JDK6</SET>
    <SET name="APPLICATION_ROOT">D:/CI/Testing/cidistribution</SET>
    <SET name="CI_AHE_ROOT">D:/CI/Java/cihostingenvironment</SET>
    <SET name="CI_INTERFACE_ROOT">D:/CI/Java/ciinterface</SET>
    <SET name="CI_EXAMPLES_ROOT">D:/CI/Java/ciexamples</SET>
    <SET name="JACORB_ROOT">D:/JacORB_2.3.0</SET>
    <SET name="LOG4J_ROOT">D:/Log4J/logging-log4j-1.2.14</SET>
  </ENVIRONMENT>

  <PROGRAM>
    $JAVA_HOME/bin/java.exe
  </PROGRAM>

  <WORKING_DIRECTORY>
    $APPLICATION_ROOT
  </WORKING_DIRECTORY>
```

Process Manager Process Descriptor (3/4)



Automation for Operations

- Example (Cont'd)

```
<JAVA>
  <BOOTCLASSPATH>
    <PATHELEMENT>$JACORB_ROOT/lib/jacorb.jar</PATHELEMENT>
    <PATHELEMENT>$JACORB_ROOT/lib/logkit-1.2.jar</PATHELEMENT>
    <PATHELEMENT>$JACORB_ROOT/lib/avalon-framework-4.1.5.jar</PATHELEMENT>
  </BOOTCLASSPATH>

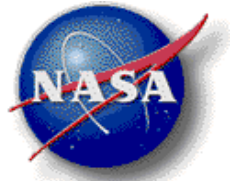
  <EXTENSIONCLASSPATH>
    <PATHELEMENT>$CI_INTERFACE_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$CI_AHE_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$CI_EXAMPLES_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$LOG4J_ROOT/dist/lib</PATHELEMENT>
  </EXTENSIONCLASSPATH>

  <CLASSPATH>
    <PATHELEMENT>$APPLICATION_ROOT/config</PATHELEMENT>
  </CLASSPATH>

  <VMOPTIONS>
    <VMOPTION>-Xincgc</VMOPTION>
    <VMOPTION>-Xmx768m</VMOPTION>
    <VMOPTION>-Xss1024k</VMOPTION>
    <VMOPTION>-Xms32m</VMOPTION>
  </VMOPTIONS>
```

Process Manager

Process Descriptor (4/4)



Automation for Operations

- Example (Cont'd)

```
<MAINCLASS>gov.nasa.ci.ahe.HostingEnvironment</MAINCLASS>  
  
</JAVA>  
  
<ARGUMENTS>  
  <ARGUMENT>HEPowerExecutive</ARGUMENT>  
</ARGUMENTS>  
</PROCESS>
```

- The CI Console can obtain the list of all available process descriptors for a process manager and select which processes to load and start.
- The CI Console can monitor the status of all processes and if necessary stop and unload processes.