On this day we will focus on implementing the logic for our component. By the end of the day, we expect the groups to:

- [ ] Learn the details of their interfaces and the interaction with other components as described in the first session and ACS Workshop - Project Details confluence page
- [ ] Learn about lifecycle methods
- [ ] Learn how to interact with other components
- [ ] Logic implementation on the methods
- [ ] Adding logging to the implementations
- [ ] Start the error handling (catch and throw) of ACS exceptions

As a guide, we're going to show you sample code for logging, to define exceptions, throw and catch exceptions.

Component Details

The component details will be presented in the first session from ACS Workshop - Project Details page to describe the interfaces and interactions of the project's components.

Lifecycle Methods

The ACS components have a couple of methods associated with its lifecycle that automatically gets executed at different phases of the component initialization and deactivation.

- initialize():
  - Called to give the component time to initialize itself. For instance, the component could retrieve connections, read in configuration files /parameters, build up in-memory tables, etc.
  - In fact, this method might be called quite some time before functional requests can be sent to the component
- execute():
  - Called after **initialize** to tell the component that it has to be ready to accept incoming functional calls any time
- cleanUp():
  - Called after the last functional call to the component has finished
  - The component should then orderly release resources etc.
- aboutToAbort():
  - Called when due to some error condition the component is about to be forcefully removed some unknown amount of time later (usually not very much...).
  - The component should make an effort to die as neatly as possible
  - Because of its urgency, this method will be called asynchronously to the execution of any other method of the component

Implementing Lifecycle functionality

- Python

```
class <name>(...):
    ...
    def initialize():
        #Assign variable values
        #Initialize data
        ...
    def execute():
        #Retrieve components
        #Consider ready to receive calls (Change states if appropriate)
        ...
    def cleanUp():
        #Release components
        #Release resources
        ...
    def aboutToAbort():
        #Do any critical clean up
        #Continue with less critical stuff such as releasing components and other activities similar to
cleanUp
        ...
    ...
```

- Java

```
import alma.acs.container.ContainerServices;
import alma.acs.component.ComponentLifecycleException;

public class <name> ... {
    ...
    public void initialize(ContainerServices containerServices) throws ComponentLifecycleException {
        super.initialize(containerServices);
        //Assign variable values
        //Initialize data
        ...
    }

    public void execute() {
        //Retrieve components
        //Consider ready to receive calls (Change states if appropriate)
        ...
    }
    public void cleanUp() {
        //Release components
        //Release resources
        ...
    }
    public void aboutToAbort() {
        //Do any critical clean up
        //Continue with less critical stuff such as releasing components and other activities similar to
cleanUp
        ...
    }
    ...
}
```

- C++

```
class <name> : ... {
    ...
    void initialize();
    void execute();
    void cleanUp();
    void aboutToAbort();
    ...
};
```

```
void <name>::initialize() {
    //Assign variable values
    //Initialize data
    ...
}

void <name>::execute() {
    //Retrieve components
    //Consider ready to receive calls (Change states if appropriate)
    ...
}

void <name>::cleanUp() {
    //Release components
    //Release resources
    ...
}

void <name>::aboutToAbort() {
    //Do any critical clean up
    //Continue with less critical stuff such as releasing components and other activities similar to
cleanUp
    ...
}
```

Retrieving and Releasing Components

During component interaction it will be needed to request and release components. When interacting with components from other components, you need to obtain the container services references to make the request to the Manager. Here we will show how to retrieve two types of components:

- Component By Name
- Default Component (By IDL)

Example Request and Release

- Python

```
#By Name
comp = self.getComponent("<Name>")

#By Interface. Must be at least one component configured as default!
comp = self.getDefaultComponent("IDL:<prefix>/<Module>/<Interface>:1.0")


#Release Components
self.releaseComponent(comp.name())
```

- Java

```
//Shared
import alma.<Module>.<Interface>;
import alma.<Module>.<Interface>Helper;

//By Name
<Interface> comp = <Interface>Helper.narrow(this.m_containerServices.getComponent("<Name>"));

//By Interface. Must be at least one component configured as default!
<Interface> comp = <Interface>Helper.narrow(this.m_containerServices.getDefaultComponent("IDL:<prefix>
/<Module>/<Interface>:1.0"));

//Release Components
m_containerServies.releaseComponent(comp.name());
```

- C++

```
//By Name
<Module>::<Interface>_var comp = this->getContainerServices()->getComponent<<Module>::<Interface>>
("<Name>");

//By Interface. Must be at least one component configured as default!
<Module>::<Interface>_var comp = this->getContainerServices()->getDefaultComponent<<Module>::<Interface>>
("IDL:<prefix>/<Module>/<Interface>:1.0");

//Release Components
this->getContainerServices()->releaseComponent(comp->name());
```

Logging

ACS Logging has two main benefits:

- Provides a standard mechanism for printing log messages
- Logs go to the "Remote Logger" for distribution of logs and storage

Example Logging

- Python

```
logger = self.getLogger()
logger.logTrace("...")
logger.logDebug("...")
logger.logInfo("...")
logger.logWarning("...")
logger.logError("...")

#An example
logger.info("A real log with a string '%s' and an int (%d)" % ("Log Entry", 3))
```

- Java

```
m_logger.finer("...");
m_logger.fine("...");
m_logger.info("...");
m_logger.warning("...");
m_logger.severe("...");


#An example
m_logger.info("A real log with a string '" + "Log Entry" + "' and an int (" + String.valueOf(3) + ")");
```

- C++

```
ACS_TRACE("...");
ACS_DEBUG("...");
ACS_SHORT_LOG((LM_INFO, "..."));
ACS_SHORT_LOG((LM_WARNING, "..."));
ACS_SHORT_LOG((LM_ERROR, "..."));


//An example
ACS_SHORT_LOG((LM_INFO, "A real log with a string '%s' and an int (%d)", "Log Entry", 3));
```

Error Handling

Error Handling in ACS Components has 4 pieces:

- Error Definitions (XML)
- Error Declarations (IDL)
- Throwing Exceptions
- Handling Exceptions

Error Definitions

Error definitions are elements in XML files with the representation for errors and completions. For instance, a simple definition present in the project:

```
...
<ErrorCode name="AlreadyInAutomatic"
           shortDescription="Already in automatic mode"
           description="Trying to set automatic mode, failed. It has already been set"/>
...
```

Error Declarations

The errors defined in the XML files need to be declared in the IDLs to be used in component communications:

```
...
    void setMode(in boolean mode) raises(SYSTEMErr::AlreadyInAutomaticEx);
...
```

Throwing Exceptions

Throwing exceptions is done in the servant that implements the IDL method that declared the error.

- Python

```
import SYSTEMErrImpl
...
    raise SYSTEMErrImpl.AlreadyInAutomaticExImpl().getAlreadyInAutomaticEx()
...
```

- Java

```
import alma.SYSTEMErr.wrappers.AcsJAlreadyInAutomaticEx;
...
    throw new AcsJAlreadyInAutomaticEx("Some message...").toAlreadyInAutomaticEx();
...
```

- C++

```
#include <SYSTEMErr.h>
...
    throw SYSTEMErr::AlreadyInAutomaticExImpl(__FILE__, __LINE__, "Some message...").
getAlreadyInAutomaticEx();
...
```

Handling Exceptions

Handling is done in the component or client that calls a component through its stub.

- Python

```
import SYSTEMErr
...
    try:
        ...
    except SYSTEMErr.AlreadyInAutomaticEx as e:
        ...
...
```

- Java

```
import alma.SYSTEMErr.AlreadyInAutomaticEx;
...
    try {
        ...
    } catch (AlreadyInAutomaticEx e) {
        ...
    }
...
```

- C++

```
#include <SYSTEMErr.h>
...
    try {
        ...
    } catch(SYSTEMErr::AlreadyInAutomaticEx &_ex) {
        ...
    }
...
```

Python

ImageType

```
#Sending --- Img (list of int[0-255]) to string (OctetSeq mapping)
img = [2,3,12,...]
octseq = bytes(bytearray(img))

#Receiving --- OctetSeq mapping to Img (list of int[0-255])
octseq = takeImage(...)
b = bytearray()
b.extend(octseq)
img = list(b)
```

ImageList

```
#Sending
img = [2,3,12,...]
octseq = str(bytearray(img))
imgList = [octseq, octseq, octseq]


#Receiving
imgList = ...
imgs = []
for octseq in imgList
    b = bytearray()
    b.extend(octseq)
    img = list(b)
    imgs.append(img)
```

Position

```
from TYPES import Position


#Sending
pos = Position(az, el)


#Receiving
pos = ...
az = pos.az
el = pos.el
```

Target

```
from TYPES import Target
from TYPES import Position


#Sending
pos = Position(az, el)
tar = Target(tid, pos, expTime)


#Receiving
tar = ...
tid = tar.tid
pos = tar.pos
eTime = tar.expTime
```

TargetList

```
from TYPES import Target
from TYPES import Position

#Sending
pos = Position(az, el)
tar = Target(tid, pos, expTime)
tList = [tar, tar, tar]

#Receiving
tarList = ...
for tar in tarList:
    tid = tar.tid
    pos = tar.pos
    eTime = tar.expTime
    ...
```

Proposal

ProposalList

RGB

Java

ImageType

```
#Sending --- Img (list of int[0-255]) to string (OctetSeq mapping)
img = [2,3,12,...]
octseq = str(bytearray(img))

#Receiving --- OctetSeq mapping to Img (list of int[0-255])
octseq = takeImage(...)
b = bytearray()
b.extend(octseq)
img = list(b)
```

ImageList

```
#Sending
img = [2,3,12,...]
octseq = str(bytearray(img))
imgList = [octseq, octseq, octseq]


#Receiving
imgList = ...
imgs = []
for octseq in imgList
    b = bytearray()
    b.extend(octseq)
    img = list(b)
    imgs.append(img)
```

Position

```
from TYPES import Position


#Sending
pos = Position(az, el)


#Receiving
pos = ...
az = pos.az
el = pos.el
```

Target

```
from TYPES import Target
from TYPES import Position


#Sending
pos = Position(az, el)
tar = Target(tid, pos, expTime)


#Receiving
tar = ...
tid = tar.tid
pos = tar.pos
eTime = tar.expTime
```

TargetList

```
from TYPES import Target
from TYPES import Position

#Sending
pos = Position(az, el)
tar = Target(tid, pos, expTime)
tList = [tar, tar, tar]

#Receiving
tarList = ...
for tar in tarList:
    tid = tar.tid
    pos = tar.pos
    eTime = tar.expTime
    ...
```

Proposal

ProposalList

RGB

C++

ImageType

```
#Send
ImageType img;
img.length(3);
img[0] = 10;
img[1] = 20;
img[2] = 30;


#Receive
ImageType_var img = ...
for(unsigned int i = 0; i < img->length(); i++) { //You should also be able to use iterators...
    ...(*img)[i]...
}
```

ImageList

```
#Send
ImageList imgList;
imgList(2);
ImageType img1;
...
imgList[0] = img1;
ImageType img2;
...
imgList[1] = img1;


#Receive
imgList_var = ...
for(unsigned int i = 0; i < imgList->length(); i++) { //You should also be able to use iterators...
    for(unsigned int j = 0; j < (*imgList)[i].length(); j++) { //You should also be able to use iterators...
        ...(*imgList)[i][j]...
}
```

Position

Target

TargetList

Proposal

ProposalList

RGB