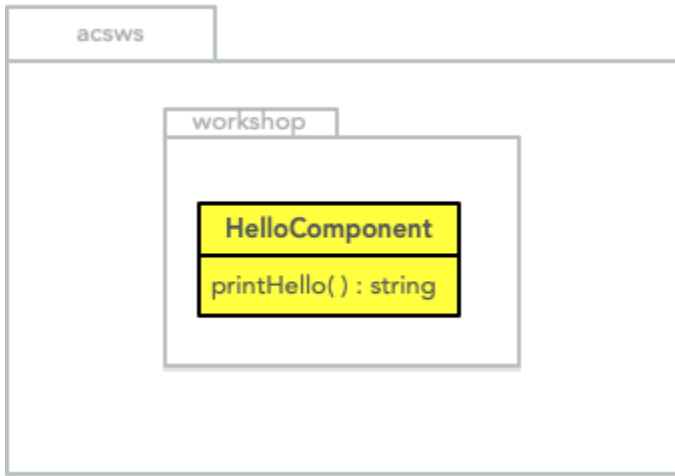


On this day we will focus on preparing the initial implementation of our first component. By the end of the day, we expect the groups to:

- ☐ Go through "HelloComponent" implementation for the assigned programming language
- ☐ Inheritance from IDL interface to be implemented
- ☐ Basic implementation of the methods
- ☐ Modifying CDB configuration in your test directory
- ☐ Exercising your component implementation from objexp and/or a PySimpleClient instance

As a guide, we're going to show you a very simple component implementation.

The UML description for our component is:



First we create the directory for the IDL:

```
getTemplateForDirectory MODROOT_WS idlHelloComp
cd idlHelloComp/src
touch ../idl/HelloComponent.idl
```

We fill the IDL with the following:

```
idlHelloComp/idl/HelloComponent.idl

#ifndef _HELLOCOMPONENT_IDL_
#define _HELLOCOMPONENT_IDL_

#pragma prefix "acsws"

#include <acscomponent.idl>

module workshop {
    interface HelloComponent : ACS::ACSCOMPONENT {
        string printHello();
    };
};

#endif
```

We modify the Makefile:

idlHelloComponent/src/Makefile

```
...  
IDL_FILES = HelloComponent  
HelloComponentStubs_LIBS = acscomponentStubs  
...  
COMPONENT_HELPERS=on  
...
```

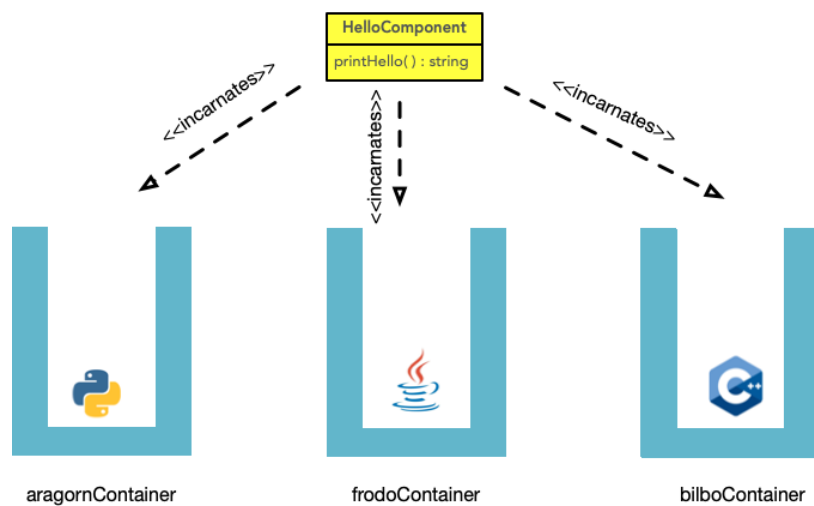
We then compile and install the IDL definitions:

Compilation

```
make -j all install
```

Configuration

The following image depicts the deployment configuration needed:



Add the following configuration in the `$ACS_CDB/CDB/MACI/Components/Components.xml`:

Components.xml

```
...
<e Name="CPP_HELLO_COMP"
  Code="HelloComponentImpl"
  Type="IDL:acsws/workshop/HelloComponent:1.0"
  Container="bilboContainer"
  ImplLang="cpp"/>
<e Name="JAVA_HELLO_COMP"
  Code="acsws.workshop.HelloComponentImpl.HelloComponentComponentHelper"
  Type="IDL:acsws/workshop/HelloComponent:1.0"
  Container="frodoContainer"
  ImplLang="java"/>
<e Name="PY_HELLO_COMP"
  Code="ws.HelloComponentImpl"
  Type="IDL:acsws/workshop/HelloComponent:1.0"
  Container="aragornContainer"
  ImplLang="py"/>
...
```

The above configuration is assuming that we're going to use the default 3 containers.

Implementation

Python

pyHelloComp

```
getTemplateForDirectory MODROOT_WS pyHelloComp
cd pyHelloComp/src
mkdir ws
touch ws/__init__.py
touch ws/HelloComponentImpl.py
```

We modify the Makefile for this component:

pyHelloComp/src/Makefile

```
...
PY_PACKAGES = ws
...
```

We fill the component code as follows:

pyHelloComp/src/ws/HelloComponentImpl.py

```
# Client stubs and definitions, such as structs, enums, etc.
import workshop
# Skeleton infrastructure for server implementation
import workshop__POA

# Base component implementation
from Acspy.Servants.ACSCComponent import ACSCComponent
# Services provided by the container to the component
from Acspy.Servants.ContainerServices import ContainerServices
# Basic component lifecycle (initialize, execute, cleanUp and aboutToAbort methods)
from Acspy.Servants.ComponentLifecycle import ComponentLifecycle

class HelloComponentImpl(workshop__POA.HelloComponent, ACSCComponent, ContainerServices, ComponentLifecycle):
    def __init__(self):
        ACSCComponent.__init__(self)
        ContainerServices.__init__(self)
        self._logger = self.getLogger()
    def printHello(self):
        print("Just printing 'Hello World!'")
        return "Hello World!"
```

Java

jHelloComp

```
getTemplateForDirectory MODROOT_WS jHelloComp
cd jHelloComp/src
mkdir -p acsws/workshop/HelloComponentImpl
touch acsws/workshop/HelloComponentImpl/HelloComponentImpl.java
cp ../../idlHelloComp/src/acsws/workshop/HelloComponentImpl/HelloComponentComponentHelper.java.tpl acsws
/workshop/HelloComponentImpl/HelloComponentComponentHelper.java
```

We modify the Makefile for this component:

jHelloComp/src/Makefile

```
...
JARFILES = HelloComponentImpl
HelloComponentImpl_DIRS = acsws
...
```

We fill the component code as follows:

jHelloComp/src/acs/ws/workshop/HelloComponentImpl/HelloComponentImpl.java

```
//Suggested: import alma.<Module>.<Interface>Impl; //But anything, really
package acs.ws.workshop.HelloComponentImpl;

//Base component implementation, including container services and component lifecycle infrastructure
import alma.acs.component.ComponentImplBase;

//Skeleton interface for server implementation
import acs.ws.workshop.HelloComponentOperations;

//ClassName usually is <Interface>Impl, but can be anything
public class HelloComponentImpl extends ComponentImplBase implements HelloComponentOperations {
    public HelloComponentImpl() {
    }
    public String printHello() {
        System.out.println("Just printing 'Hello World!'");
        return new String("Hello World!");
    }
}
```

C++

We create the needed directories:

cppHelloComp

```
getTemplateForDirectory MODROOT_WS cppHelloComp
cd cppHelloComp/src
touch HelloComponentImpl.cpp
touch ../include/HelloComponentImpl.h
```

We modify the Makefile for this component:

cppHelloComp/src/Makefile

```
...
INCLUDES = HelloComponentImpl.h
...
LIBRARIES = HelloComponentImpl
HelloComponentImpl_OBJECTS = HelloComponentImpl
HelloComponentImpl_LIBS = HelloComponentStubs acscomponent
...
```

We fill the component code as follows:

cppHelloComp/include/HelloComponentImpl.h

```
#ifndef _HELLO_COMPONENT_IMPL_H
#define _HELLO_COMPONENT_IMPL_H

#ifdef __cplusplus
#error This is a C++ include file and cannot be used from plain C
#endif

//Base component implementation, including container services and component lifecycle infrastructure
#include <acscomponentImpl.h>

//Skeleton interface for server implementation
#include <HelloComponentS.h>

//Error definitions for catching and raising exceptions
class HelloComponentImpl : public virtual acscomponent::ACSCComponentImpl, public virtual POA_workshop::
HelloComponent {
public:
    HelloComponentImpl(const ACE_CString& name, maci::ContainerServices * containerServices);
    virtual ~HelloComponentImpl();
    char* printHello();
};

#endif
```

cppHelloComp/src/HelloComponentImpl.cpp

```
#include <HelloComponentImpl.h>

HelloComponentImpl::HelloComponentImpl(const ACE_CString& name, maci::ContainerServices * containerServices) :
ACSCComponentImpl(name, containerServices) {
}

HelloComponentImpl::~HelloComponentImpl() {
}

char* HelloComponentImpl::printHello() {
    std::cout << "Just printing 'Hello World!'" << std::endl;
    return CORBA::string_dup("Hello World!");
}

/* ----- [ MACI DLL support functions ] ----- */
#include <maciACSCComponentDefines.h>
MACI_DLL_SUPPORT_FUNCTIONS(HelloComponentImpl)
/* ----- */
```

With this information we are ready to compile our first component that will print the "Hello World" message.

Client

First launch your [acs container](#) with the modified ACS CDB

Start container

```
acsStartContainer -py aragornContainer
acsStartContainer -java frodoContainer
acsStartContainer -cpp bilboContainer
```

For simplicity, we implement a simple client in Python to communicate with the 3 programming languages:

```

from Acspy.Clients.SimpleClient import PySimpleClient

client = PySimpleClient()

hc_py = client.getComponent("PY_HELLO_COMP")
print(hc_py.printHello())

hc_java = client.getComponent("JAVA_HELLO_COMP")
print(hc_java.printHello())

hc_cpp = client.getComponent("CPP_HELLO_COMP")
print(hc_cpp.printHello())

```

Output

The output is seen on each container:

Py

```

...
2020-07-28T21:14:29.850 aragornContainer run - Calling maci::CBComponentInfo::done with descOut.id_tag = 2 for
'PY_HELLO_COMP'
2020-07-28T21:14:29.858 aragornContainer run - Call to maci::CBComponentInfo::done with descOut.id_tag = 2 for
'PY_HELLO_COMP' completed
Just printing 'Hello World!'
...

```

Java

```

...
2020-07-28T21:09:13.385 DELOUSE [frodoContainer] Calling maci::CBComponentInfo::done with descOut.id_tag = %d.1
for 'JAVA_HELLO_COMP'
2020-07-28T21:09:13.411 DELOUSE [frodoContainer] Call to maci::CBComponentInfo::done with descOut.id_tag = %d.1
for 'JAVA_HELLO_COMP' completed
2020-07-28T21:09:16.598 DELOUSE [frodoContainer] intercepted a call to 'JAVA_HELLO_COMP#printHello'.
Just printing 'Hello World!'
2020-07-28T21:09:16.599 DELOUSE [frodoContainer] returning from JAVA_HELLO_COMP#printHello after 0 ms.
...

```

C++

```

...
2020-07-28T21:19:53.080 [Container-ActivationMethod - maci::ActivationMethod::call] Calling maci::
CBComponentInfo::done with descOut.id_tag = 4.
2020-07-28T21:19:53.082 [Container-ActivationMethod - maci::ActivationMethod::call] Call to maci::
CBComponentInfo::done with descOut.id_tag = 4 completed.
Just saying 'Hello World!'
...

```

PySimpleClient

```

...
Hello World!
Hello World!
Hello World!
...

```