The most common use of communication in ACS is the use of synchronous calls, however there are different reasons why someone could be interested in a different kind of calls. For this, CORBA provides the definition of 'oneway' methods, which basically are dispatched and no response is expected by the client. ACS increases this functionality by adding callbacks functionality to the 'oneway' mechanism offered by CORBA (If the developer desires). These callbacks are based on the off-shoot and callback interfaces and are a CORBA object on their own, although they don't represent an ACS component, they can be interacted in a very similar fashion, except for the lifecycle, which is tied to the process creating it.

OffShoot is a base interface used to identify CORBA objects created by components, clients, or other processes, and whose lifecycle is limited to the that of the component or process who created it.

A callback is the technical mechanism underlying the concepts of asynchronous notification. It is an object passed by the client to the server, so that the server can later invoke methods on the callback and thus inform the client of a change in status, completion of some operation, and the like. During this notification the roles of the client and the server are reversed. Every callback must be subclassed from the Callback interface.

ACS provides some Callbacks for basic types:

- CBvoid
- CBdouble(Seq)
- CBfloat(Seq)
- CBstring(Seq)
- CBlong(Seq)
- CBuLong(Seq)
- CBboolean(Seq)
- CBlongLong
- CBuLongLong

- Scope
    - Simple oneway calls
    - Offshoot/Callback Architecture
- Duration: 10 minutes

Component IDL

Async.idl

```
#ifndef _ASYNC_IDL_
#define _ASYNC_IDL_

#pragma prefix "alma"

#include <acscommon.idl>
#include <acscomponent.idl>

module workshop
{
    interface Async : ACS::ACSComponent {
        oneway void delayResult(in ACS::uLong delay, in ACS::CBuLong cb, in ACS::CBDescIn desc);
    };
};
#endif
```

Implementation

AsyncImpl.h

```
#ifndef _ASYNC_IMPL_H
#define _ASYNC_IMPL_H

#include <acscomponentImpl.h>
#include <AsyncS.h>

class AsyncImpl : public virtual acscomponent::ACSComponentImpl, public virtual POA_workshop::Async
{
  public:
    AsyncImpl(const ACE_CString& name, maci::ContainerServices* containerServices);
    virtual ~AsyncImpl();
    void delayResult(ACS::uLong delay, ACS::CBuLong_ptr cb, const ACS::CBDescIn& desc);
};
#endif
```

AsyncImpl.cpp

```
#include <AsyncImpl.h>
#include <ACSErrTypeOK.h>

AsyncImpl::AsyncImpl(const ACE_CString& name, maci::ContainerServices* containerServices) : acscomponent::
ACSComponentImpl(name, containerServices) {
}

AsyncImpl::~AsyncImpl() {
}

void AsyncImpl::delayResult(ACS::uLong delay, ACS::CBuLong_ptr cb, const ACS::CBDescIn& desc) {
    ACSErr::Completion completion;
    ACS::CBDescOut descOut;

    completion = ACSErrTypeOK::ACSErrOKCompletion();
    cb->working(0, completion, descOut);
    sleep(delay);
    cb->done(delay, completion, descOut);
}

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

Configuration

Components.xml

```
<e Name="ASYNC"
   Code="AsyncImpl"
   Type="IDL:alma/workshop/Async:1.0"
   Container="bilboContainer"
   ImplLang="cpp"
   KeepAliveTime="0"
/>
```

Python

AsyncClient.py

```
import ACS
import time

from Acspy.Common.Callbacks import CBuLongLong
from Acspy.Clients.SimpleClient import PySimpleClient

client = PySimpleClient()
comp = client.getComponent('ASYNC')

cb = CBuLongLong()
cbObj = client.activateOffShoot(cb)
desc = ACS.CBDescIn(0, 0, 0)

print(cb.status)
comp.delayResult(8, cbObj, desc)

print(cb.status)
time.sleep(5)
print(cb.status)
time.sleep(5)
print(cb.status)

print(cb.values)

client.releaseComponent(comp.name)
client.disconnect()
```

Java

AsyncClient.java

```
package workshop.async;

import java.util.logging.Logger;

import alma.acs.component.client.ComponentClient;

import alma.ACS.CBDescIn;
import alma.ACS.CBDescOut;
import alma.ACSErr.Completion;

import alma.workshop.Async;
import alma.workshop.AsyncHelper;

import alma.ACS.CBuLong;
import alma.ACS.CBuLongPOA;

public class AsyncClient extends ComponentClient {
    class CBuLongImpl extends CBuLongPOA
    {
        public int value;
        public String status;
        public CBuLongImpl(Logger logger) {
            status = "INIT";
            this.value = 0;
        }
        public void working(int value, Completion completion, CBDescOut desc) {
            status = "WORKING";
            this.value = value;
        }
        public void done(int value, Completion completion, CBDescOut desc) {
            status = "DONE";
            this.value = value;
        }
```

```
        public boolean negotiate(long timeToTransit, CBDescOut desc) {
            return true;
        }
    }

    private Logger m_logger;

    public AsyncClient() throws Exception {
        super(null, System.getProperty("ACS.manager"), "AsyncClient");
        m_logger = getContainerServices().getLogger();
    }

    public void doStuff() {
        try {
            org.omg.CORBA.Object obj = getContainerServices().getComponent("ASYNC");

            Async comp = AsyncHelper.narrow(obj);
            CBuLongImpl cb = new CBuLongImpl(m_logger);
            CBuLong cbObj = alma.ACS.CBuLongHelper.narrow(getContainerServices().activateOffShoot(cb));
            CBDescIn desc = new CBDescIn();

            m_logger.info(String.valueOf(cb.status));
            comp.delayResult(8, cbObj, desc);

            m_logger.info(String.valueOf(cb.status));
            Thread.sleep(5000);
            m_logger.info(String.valueOf(cb.status));
            Thread.sleep(5000);

            m_logger.info(String.valueOf(cb.status));
            m_logger.info(String.valueOf(cb.value));

            getContainerServices().releaseComponent("ASYNC");
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            AsyncClient client = new AsyncClient();
            client.doStuff();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

C++

AsyncClient.cpp

```
#include <maciSimpleClient.h>
#include <AsyncC.h>
#include <ACSErrTypeCommon.h>
#include <acsutilTimeStamp.h>

using namespace maci;
class CBuLongImpl : public virtual POA_ACS::CBuLong {
  public:
    CBuLongImpl() {value = 0; status=std::string("INIT");}
    virtual ~CBuLongImpl() {}
    void working (ACS::uLong value, const ACSErr::Completion &c, const ACS::CBDescOut &desc) {
        status = std::string("WORKING");
        this->value = value;
    }
    void done (ACS::uLong value, const ACSErr::Completion &c, const ACS::CBDescOut &desc){
        status = std::string("DONE");
```

```
        this->value = value;
    }
    CORBA::Boolean negotiate (ACS::TimeInterval time_to_transmit, const ACS::CBDescOut &desc) {
            return true;
    }
  public:
    ACS::uLong value;
    std::string status;
};

int main(int argc, char *argv[]) {
    SimpleClient client;
    workshop::Async_var comp;

    if (client.init(argc,argv) == 0) {
        return -1;
    } else {
        client.login();
    }

    try {
        comp = client.getComponent<workshop::Async>("ASYNC", 0, true);
    } catch(maciErrType::CannotGetComponentExImpl& _ex) {
        _ex.log();
        return -1;
    }

    CBuLongImpl cb{};
    ACS::CBuLong_var cbObj = cb._this();
    ACS::CBDescIn desc;
    desc.id_tag = 2;

    ACS_SHORT_LOG((LM_INFO, "%s", cb.status));
    comp->delayResult(8, cbObj.in(), desc);

    ACS_SHORT_LOG((LM_INFO, "%s", cb.status));
    sleep(5);
    ACS_SHORT_LOG((LM_INFO, "%s", cb.status));
    sleep(5);

    ACS_SHORT_LOG((LM_INFO, "%s", cb.status));
    ACS_SHORT_LOG((LM_INFO, "%ul", cb.value));

    try {
        client.releaseComponent("ASYNC");
    } catch(maciErrType::CannotReleaseComponentExImpl &_ex) {
        _ex.log();
        return -1;
    }

    client.logout();

    ACE_OS::sleep(3);
    return 0;
}
```

Makefile

Makefile

```
#Component IDL
IDL_FILES_L = Async
AsyncStubs_LIBS = acscomponentStubs acscommonStubs

#C++ Component
LIBRARIES_L     = AsyncImpl
AsyncImpl_OBJECTS = AsyncImpl
AsyncImpl_LIBS = AsyncStubs acscomponent

#C++ Client
EXECUTABLES_L   = AsyncExample
AsyncExample_OBJECTS = AsyncClient
AsyncExample_LIBS = maciClient AsyncStubs

#Java Client
JARFILES_L = AsyncJar
AsyncJar_DIRS = workshop

#Python Client
PY_SCRIPTS_L       = AsyncClient
```

- Asynchronous calls and oneway operations
    - It's not mandatory to make use of oneway keyword, depends on the design
        - For instance, adding to a queue may return immediately with a boolean with success or failure status of adding the element to a queue and may still register a callback for when the element is executed and something needs to be done
- Existing CBxxxx are used for monitoring, alarms and other internals of ACS, they're not meant for generic use. Your own callbacks can have the desired set of methods
- Improvements / Suggestions