Asynchronous calls are mechanisms to execute actions in a decoupled fashion. There are different approaches for asynchronous calls, depending on the level of control that is required. Some of them are offered by CORBA, ACS and BACI implementations. For instance we have:

- threading: Programming Language feature
- queuing: Design/Programming Language
- oneway: CORBA functionality
- offshoot: ACS functionality
- callbacks: offshoot specialization by ACS
- actions: BACI implementation based on callbacks

Threading

One approach for asynchronous calls based on the programming language functionality, is simply to use threading inside a method call, to execute the action in a separate thread and to return instantaneously.

```
void calibrateSync() {
  // do calibration...
}

void ExampleImpl::calibrate() {
    new std::thread(calibrateSync); //Thread resources are never de-allocated for simplicity
}
```

Queuing

A second approach based on design and programming language alone, is to populate queues with the actions that will be orchestrated by a separate thread.

```
void calibrateSync() {
    // do calibration...
}

//Function running on some other thread...
void run() {
    if (queue.size() > 0) {
        if (queue.pop() == CALIBRATE_ACTION) {
            calibrateSync();
        } else {
            //Some error
        }
    }
}

void ExampleImpl::calibrate() {
    queue.push(CALIBRATE_ACTION);
}
```

Oneway

Oneway is a concept in CORBA that is defined as a keyword in the IDL interface methods, which makes sure the caller continues with the execution without waiting for anything from the client. This is similar to threading/queuing options, but is stronger, in the sense that it is not possible to receive a return value nor to raise an exception.

The IDL itself has the information about the asynchronous mechanism:

```
interface Example: ... {
    ...
    oneway void calibrate();
    ...
}
```

The actual implementation is very simple:

```
void ExampleImpl::calibrate() {
    // do calibration...
}
```

Offshoots

Offshoots are the most basic form to report back to the caller. The base interface is actually empty:

```
interface OffShoot {};
```

This is by design, allowing anyone to extend such interface as they prefer. For instance:

```
module Definitions {
    interface SimpleCallback : ACS::OffShoot {
        oneway void report(in boolean error);
    };
};
```

Has a single method that can be called 'report', which has an 'in' parameter, stating whether there was an error. It could be used in the 'calibrate' example as follows:

```
interface Example: ... {
    ...
    oneway void calibrate(in Definitions::SimpleCallback cb);
    ...
}
```

Which would lead to an implementation (Assuming the oneway case) as follows:

```
void ExampleImpl::calibrate(SimpleCallback* cb) {
    // do calibration...
    cb->report(status);
}
```

The offshoot can be used with either of the 3 asynchronous mechanisms, and you will have to decide on one of them depending on your preferences and a trade-off limitations/requirements from each approach.

Callbacks

Callbacks are in fact a specialization of the Offshoot and were designed to be used mainly by BACI (Monitoring, Actions, etc.). These are based on the basic ACS types (void, float, double, long, boolean, string, etc.).

They can be used manually, but they're discouraged, because they depend on CBDescIn and CBDescOut structures. These are used to define the expected timeout and to hold back the Tag ID of the execution, but they don't have any use on a custom implementation, so it's better to use an extension to the Offshoot if you're making your own implementation for reporting back to the caller.

An example of callback is the CBvoid interface:

```
interface Callback : OffShoot {
    boolean negotiate (in TimeInterval time_to_transmit, in CBDescOut desc);
};

interface CBvoid : Callback {
    oneway void working (in ACSErr::Completion c, in CBDescOut desc);
    oneway void done (in ACSErr::Completion c, in CBDescOut desc);
};
```

BACI Actions

BACI Actions make use of these Callback definitions to leverage the execution of asynchronous actions. The same example as before could go something like this:

```
interface Example: ... {
    ...
    oneway void calibrate(in ACS::CBvoid cb, in ACS::CBDescIn desc);
    ...
}
```

The implementation needs to have the calibrateAction returning one of the four alternatives:

- reqNone: Do nothing; the request will remain in the queue
- reqInvokeWorking: Notify a working update; the request will remain in the queue
- reqInvokeDone: Notify that the request has finished; the request is removed from the queue
- reqDestroy: Remove the request from the queue without notifying (When it should have already been destroyed or if you want to ignore the request completely)

Also it needs to modify the invokeAction method, to ensure that the correct action is called:

```
ActionRequest ExampleImpl::calibrateAction (BACIComponent *cob_p, const int &callbackID, const CBDescIn
&descIn, BACIValue *value_p, Completion &completion, CBDescOut &descOut) {
    completion = ACSErrTypeOK::ACSErrOKCompletion();
    // do calibration...
    completion = //Actual status of calibration process
    return reqInvokeDone;
}

ActionRequest ExampleImpl::invokeAction (int function, BACIComponent *cob_p, const int &callbackID, const
CBDescIn &descIn, BACIValue *value_p, Completion &completion, CBDescOut &descOut) {
    switch (function) {
        case CALIBRATE_ACTION: {
            return calibrateAction(cob_p, callbackID, descIn, value_p, completion, descOut);
        }
        default: {
            return reqDestroy;
        }
    }
}

void ExampleImpl::calibrate(ACS::CBvoid_ptr cb, const ACS::CBDescIn& desc) {
    getComponent()->registerAction(BACIValue::type_null, cb, desc, this, CALIBRATE_ACTION);
}
```

More information can be found in BACI Device Server Programming Tutorial.

Oneway vs Threading/Queuing

- In oneway operations, the code is leaner and is easier to implement
- The resources are handled by CORBA, so it's easier to implement and less error prone
    - However we can reach unexpected limits (For instance the request Thread Pool)
    - Could affect other CORBA calls
- Without oneway definition, you would have to make sure of returning fast (after creating a thread, queuing, etc.)

Offshoots vs Callbacks/BACI Actions

- Offshoots are more flexible and leaner
- Callbacks have the CBdescIn and CBdescOut dependency
- Callbacks over BACI Actions offer code already implemented and standard interface