In a work performed in collaboration with CTA, we prepared a small prototype to mock-up the ContainerServices class in C++. This mock-up allowed to retrieve components and make them interact, without requiring a CORBA connection and making it possible to do simple integration testings without ACS running and taking advantage of unit testing framework.

The benefits of this comes from performance and simplicity of the testing environment.

For reference, an ACS ticket already exists for integrating this prototype into ACS:

➕ **ACS-11** - Implement mock container services for testing without ACS running `TO DO`

There are several limitations, some of which could be mitigated by extending the ContainerServices mock-up functionalities. At the moment of writing:

- Only retrieving components is implemented in the prototype (Alarms, CDB, etc. are missing)
- This only works with components interacting in the same programming language

The C++ prototype extends the ContainerServices class. The important methods to consider here, are the activateComponent/deactivateComponent which are to be used by the setUp and cleanUp methods in Unit testing.

MockContainerServices.h

```
class MockContainerServices : public maci::ContainerServices {
  public:
    MockContainerServices(ACE_CString& componentName, PortableServer::POA_ptr poa);
    ~MockContainerServices();
    virtual CORBA::Object* getCORBAComponent(const char* name);
//... Other methods omitted for simplicity
    template<class TObject> void activateComponent(const char* name) {
        TObject* obj = new TObject(name, this);
        obj->initialize();
        obj->execute();
        this->comps[name] = obj
    };
    virtual deactivateComponent(const char* name) {
        acscomponent::ACSComponentImpl* obj = dynamic_cast<acscomponent::ACSComponentImpl*>(this->comps[name].
in());
        try {
            obj->cleanUp();
        } catch (...) {
            obj->aboutToAbort();
        }
        this->comps.erase(name);
    }
  protected:
    std::map<std::string, CORBA::Object_var> comps;
};
```

The getCORBAComponent method was also extended in order to return the component from 'comps' std::map:

MockContainerServices.cpp

```
CORBA::Object* MockContainerServices::getCORBAComponent(const char* name){
    if (comps.find(name) == comps.end()) {
        maciErrType::CannotGetComponentExImpl ex(__FILE__, __LINE__, "MockContainerServices::getComponent");
        ex.setCURL(name);
        throw ex;
    }
    return CORBA::Object::_duplicate(comps[name].in());
}
```

Later on, in the test implementation we would do something similar to:

TestExample.cpp

```cpp
MockContainerServices* mcs = nullptr;

setUp() {
    mcs = new MockContainerServices(cn, nullptr);
    mcs->activateComponent<MaciTestComponentImpl>("TEST_COMP1");
    mcs->activateComponent<MaciTestComponentImpl>("TEST_COMP2");
}

cleanUp() {
    mcs->deactivateComponent("TEST_COMP1");
    mcs->deactivateComponent("TEST_COMP2");
    delete mcs;
    mcs = nullptr;
}

//The tests should not know anything about the changes we've done, but they rely on this mcs instance just for
convenience
test_example() {
    MACI_TEST::MaciTestComponent_var comp = mcs->getComponent<MACI_TEST::MaciTestComponent>("TEST_COMP1");
    comp->some_method(...); //If some_method retrieves a component, it should also work normally, since on
creation, we passed an instance of MockContainerServices
}
```

Assume MaciTestComponent is an interface definition in an IDL file extending from ACS Component interface and that MaciTestComponentImpl is the actual implementation in C++.

This line of work opens a couple of opportunities to improve the testing of ACS and the projects that run on top of it, simplifying and accelerating some test case scenarios by avoiding the complexities required for distributed systems. It also allows some level of integration testing, without requiring real deployments. That said, this is just a minimal example, which could be improved in several ways:

- Integrate these prototypes into ACS
- Extend the idea to Java and Python
- Support other types of component instantiation (non-sticky, dynamic, etc.) for completion
- Support dummy CDBs, Alarms and other functionalities offered by the container service