FUSION FOR ENERGY

MARTe2 Users Meeting
Objects, containers and smart pointers

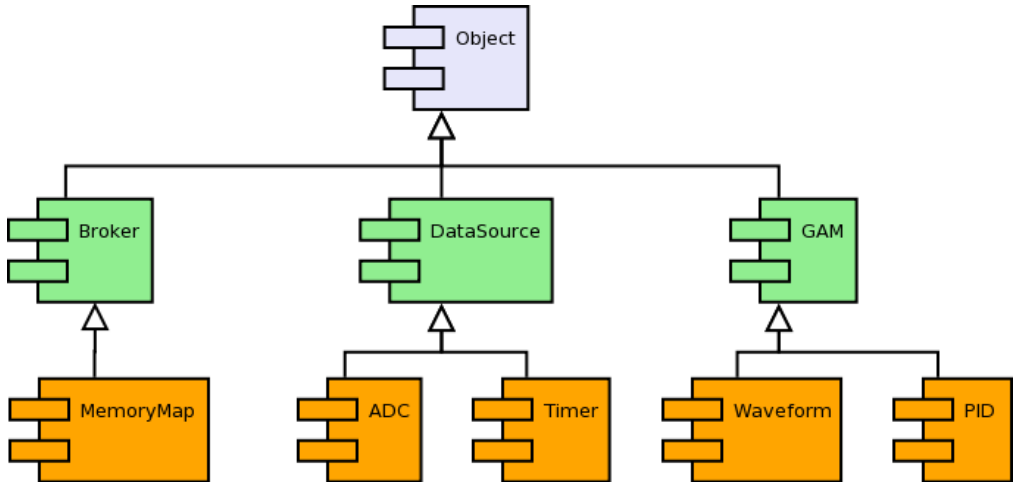
Andre Neto, Filippo Sartori

May, 2019

Object

Root class of the framework and offers the following features

- Automatic life cycle management using a smart pointer mechanism (see References);
- Data driven construction (in runtime) using the class name;
- Standard initialisation and configuration interface (more information here);
- Allocation on a user selectable heap.



ClassRegistryDatabase

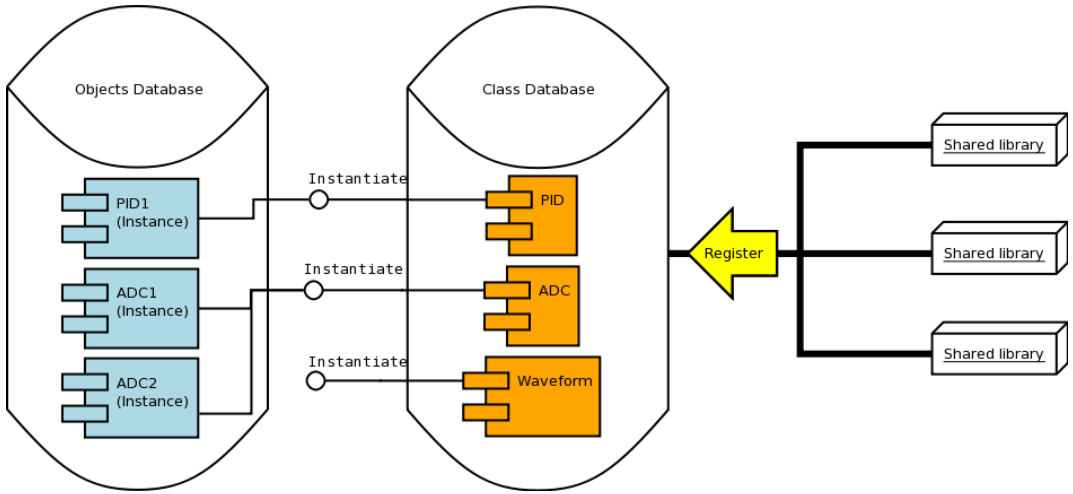
Database with all the classes that can be data-driven instantiated

```
class ControllerEx1: public MARTe::Object {
public:
  /**
   * Compulsory for the class to be automatically registered.
   */
  CLASS_REGISTER_DECLARATION()

    ControllerEx1() {
  }
  //Any other class methods
  ...
};

//Usually declared in the .cpp unit file.
CLASS_REGISTER(ControllerEx1, "")
```

- Classes automatically registered at application start-up or;
- Classes automatically registered when shared library is opened

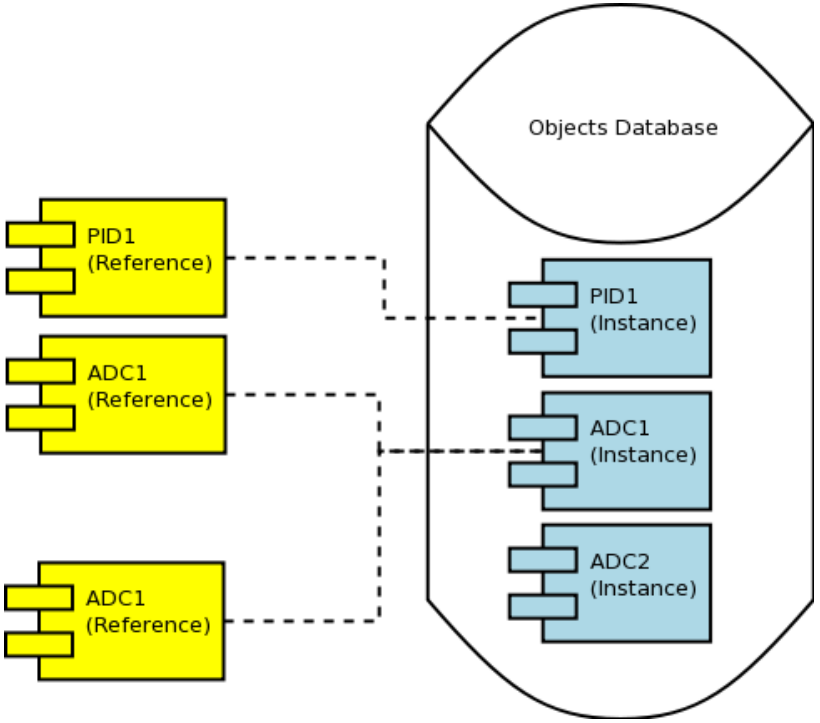


References

Smart pointer mechanism which guarantees that any MARTe Object can be safely shared (thread and interrupt safe)

- Also allows to verify that classes are compatible with a given implementation

```
ReferenceT<PID1> myController("PID1", heap);
ReferenceT<ADC1> myADC = myController;
//This will fail because ADC1 is not of type PID1
if (myADC IsValid()) {
...
ReferenceT<GAM> myGAM = myController;
//This will work because ADC1 is inherits from GAM
if (myGAM IsValid()) {
...
```

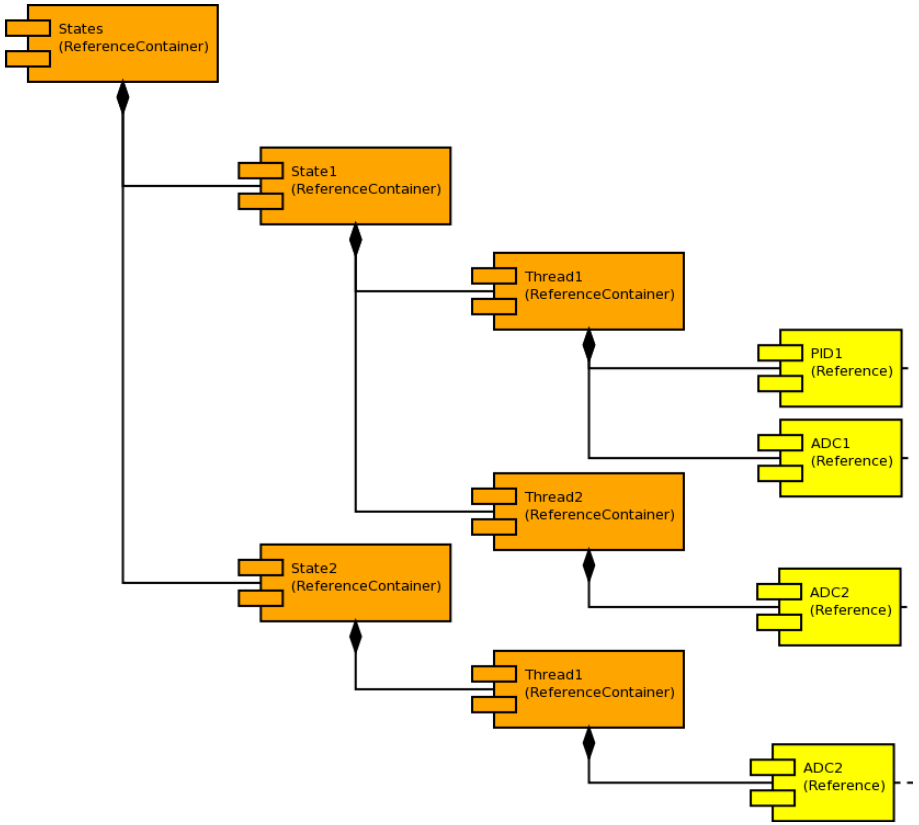


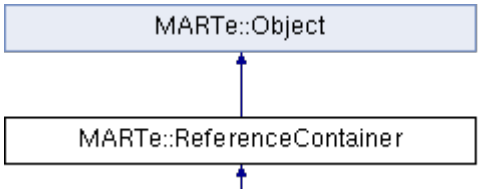
Note

The new and delete operators shall not be used to manage the life cycle of any of the framework objects. The framework offers an object creation mechanism that returns a reference.

ReferenceContainer

Core Class that is a navigable container of (references to) objects






```
const char8 * const path = "State1.Thread1.PID1";
ReferenceT<Controller> myController = Find(path);
```

Introspection

Register in the ClassRegistryDatabase members of a class or a C struct.

- Requires some complex macros to be hand-written
  - Plan is to have tools which will automatically perform this work
  - In MARTe1 this was achieved with ***cint***

```
struct A {
  MARTe::float32 f1;
  MARTe::float32 f2;
};
struct B {
  struct A a1;
  struct A a2;
};
...
DECLARE_CLASS_MEMBER(A, f1, float32, "", "");
DECLARE_CLASS_MEMBER(A, f2, float32, "", "");
static const MARTe::IntrospectionEntry* AStructEntries[] = { &A_f1_introspectionEntry, &A_f2_introspectionEntry, 0 };
DECLARE_STRUCT_INTROSPECTION(A, AStructEntries)
```

Exercises

- Develop and register classes
- Use ReferenceContainers
- Further reading and examples:
  - https://vcis.f4e.europa.eu/marte2-docs/master/html/core/scheduling/threads.html
  - https://vcis.f4e.europa.eu/marte2-docs/master/html/core/scheduling/services.html

Register a new class

Objective: register a new class and list all the classes

- Modify Other/Examples/ObjectsExample1.cpp
- Add a new class making sure that it registered using the macros
- Compile

```
cd ~/Projects/MARTe2-demos-padova/
export MARTe2_DIR=~/Projects/MARTe2-dev
export MARTe2_Components_DIR=~/Projects/MARTe2-components/
make -f Makefile.x86-linux
```

- Run the application

```
cd ~/Projects/MARTe2-demos-padova/
export LD_LIBRARY_PATH=.:$MARTe2_DIR/Build/x86-linux/Core/Build/x86-
linux/Components/Other/Examples/ObjectsExample1.ex
```

- Check that the class is listed

Success: application executes and lists all the registered classes

References

Objective: use references and underlying RTTI

- Modify Other/Examples/ReferencesExample3.cpp
- Create and register a new class, named DieselEx1, that inherits from MotorEx1
- In the main, create a new instance of DieselEx1 and verify that indeed it can be used as a MotorEx1 (e.g. by printing in the screen the output of IsValid())
- Compile

```
cd ~/Projects/MARTe2-demos-padova/
export MARTe2_DIR=~/Projects/MARTe2-dev
export MARTe2_Components_DIR=~/Projects/MARTe2-components/
make -f Makefile.x86-linux
```

- Run the application

```
cd ~/Projects/MARTe2-demos-padova/
export LD_LIBRARY_PATH=.:$MARTe2_DIR/Build/x86-linux/Core/
Build/x86-linux/Components/Other/Examples/ReferencesExample3.ex
```

Success: application executes and it is shown that DieselEx1 inherits from MotorEx1

Reference Containers

Objective: understand that ReferenceContainers can be used to group and organize references

- Modify Other/Examples/ReferencesExample5.cpp
- Implement the function **Do()** so that it calls the method **AFunction** on all of its childs that are of type ControllerEx1
- Compile

```
cd ~/Projects/MARTe2-demos-padova/
export MARTe2_DIR=~/Projects/MARTe2-dev
export MARTe2_Components_DIR=~/Projects/MARTe2-components/
make -f Makefile.x86-linux
```

- Run the application

```
cd ~/Projects/MARTe2-demos-padova/
export LD_LIBRARY_PATH=.:$MARTe2_DIR/Build/x86-linux/Core/
Build/x86-linux/Components/Other/Examples/ReferencesExample5.ex
```

Success: application calls AFunction on all the childs of a ReferenceContainer

FUSION FOR ENERGY

Thank you for your attention

Follow us on:

www.f4e.europa.eu

www.twitter.com/fusionforenergy

www.youtube.com/fusionforenergy

www.linkedin.com/company/fusion-for-energy

www.flickr.com/photos/fusionforenergy