MDSplus interfaces in MARTe

1. Introduction

A set of MARTe components allow integrating real-time control carried out by the MARTe system with the MDSplus framework for data management and storage. The integration of the two frameworks introduces the following functionality in MARTe:

1) ***MARTe parameter definition in MDSplus experiment models***. An MDSplus experiment model represents a database template which contains all the configuration data for a given experiment. This template is then populated with acquired data during the execution of the experiment, eventually producing a Pulse file, representing a complete description of the executed experiment (e.g. a plasma discharge). Parameters used by MARTe GAMs (and other components) can be defined in the experiment model and made available to the MARTe components during real-time computation.
2) ***Storage of signals produced by MARTe during real-time computation in MDSplus pulse files.*** It is possible to define in the MARTe configuration file instances of OutputGAM which are connected to a MDSplus-aware GACQM. Data received in real-time are then stored on-line in MDSPlus pulse files, possibly by CPU cores not used for real-time computation, and are made soon available in the target MDSplus pulse file.
3) ***Generation of reference signals stored in MDSplus experiment models during MARTe real-time computation***. Reference signals represent time-dependent parameters and can be defined in MDSplus experiment models as the other parameters. A reference signal will be specified by two arrays for the actual parameter values and the associated times, respectively. During real-time, the interpolated value based on the current time will be generated by MARTe InputGAMs connected to a MDSplus-aware GACQM.

2. MDSplus abstraction for MARTe interaction

A MDSplus device (called MARTE_DEVICE) is available in the MDSplus distribution to handle both configuration data for MARTe and to store signals generated by MARTe during real-time operation. MDSplus devices represent sets of related data items which are handled by MDSplus as unique entities. The concept is similar to that of Objects in Object Oriented (OO) programming. As an object can contain several fields, an MDSplus device contains several MDSplus data items, and it is actually represented as a subtree in the tree (hierarchical) structure of MDSplus experiment models and pulse files. As an object belongs to a class in OO, defining the structure of every instance for that object, every MDSplus device belongs to Device type (class) defining the structure of every associated subtree in main MDSplus data tree. Finally, as methods can be associated to objects on OO, methods are also defined for MDSplus devices, representing operations using the associated data

structure. A typical usage of MDSplus devices is the representation of hardware modules in MDSplus pulse files. Every module (e.g. an ADC device) has a set of associated configuration data and will define a set of acquired signal. Its methods will refer to initialization or data acquisition operations for that module instance which require data stored in the device instance or produce signals to be stored in the associated device instance. A common method for every MDSplus device is the Setup() one which is activated by the visual tools for browsing MDSplus experiment models and which show a Dialog for user-friendly device configuration during experiment set-up.

The MARTE_DEVICE MDSplus device will host both MARTe configuration (parameters and reference signals) and a number of signals produced by MARTe in real-time execution. Several MARTE_DEVICE instances can be defined in the MDSplus pulse file and can be referenced in the MARTe configuration file. The ID field of the device will be used to identify the particular instance of the device in the pulse file. It will contain a unique integer identifier which will be specified in the associated MARTe configuration (see below).
The graphical interface of MARTE_DEVICE is shown in Fig. 1:

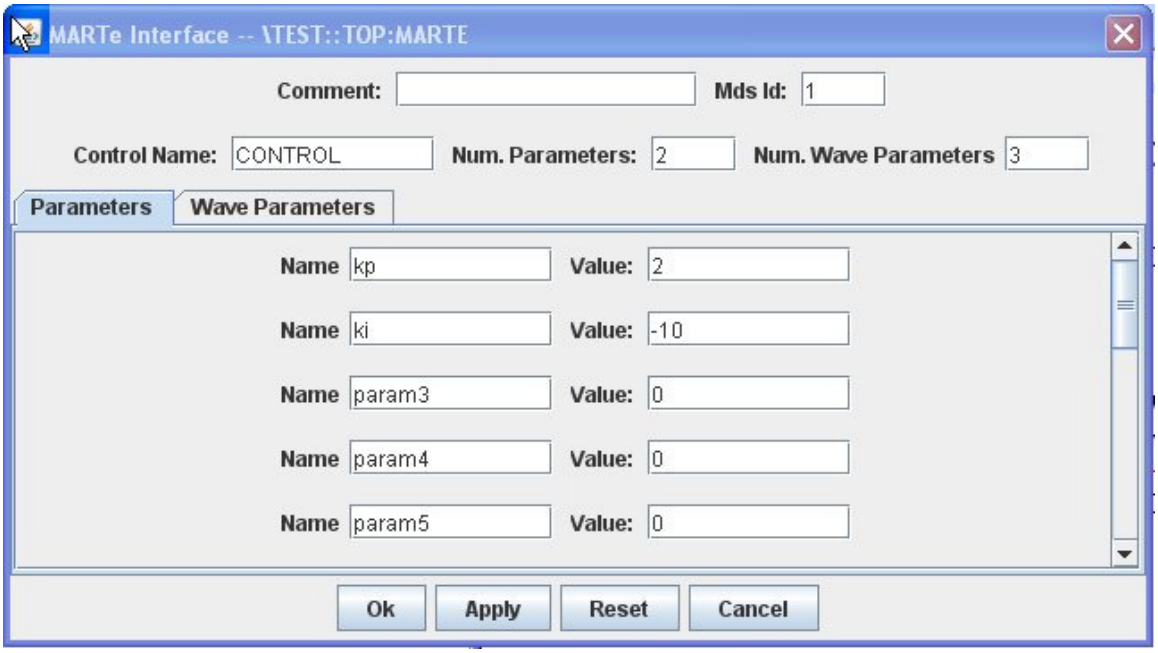

Figure *1: Parameter definition of MARTE_DEVICE*

The following fields are defined:

- ***Comment***: a string not used in MARTe to contain any additional textual information for that device instance;
- ***MdsId***: the unique identifier for that instance. The same numerical value will be also reported in the MARTe configuration file to associate the proper instance to MARTe data;
- ***Control Name***: a descriptive string for this device instance, which will be shown in the associated MARTe HTTP interface;

- ***Num. Parameters***: actual number of used parameters (the device allows the definition of up to 16 parameters);
- ***Num. Wave Parameters***: actual number of used reference waveforms (the device allows the definition of up to 8 reference waveforms);

Every parameter is defined by a Name and a Value. The name will correspond to the name defined in the MARTe configuration, and the associated value will be loaded before real-time computation.
The configuration of reference waveforms is shown in Fig. 2

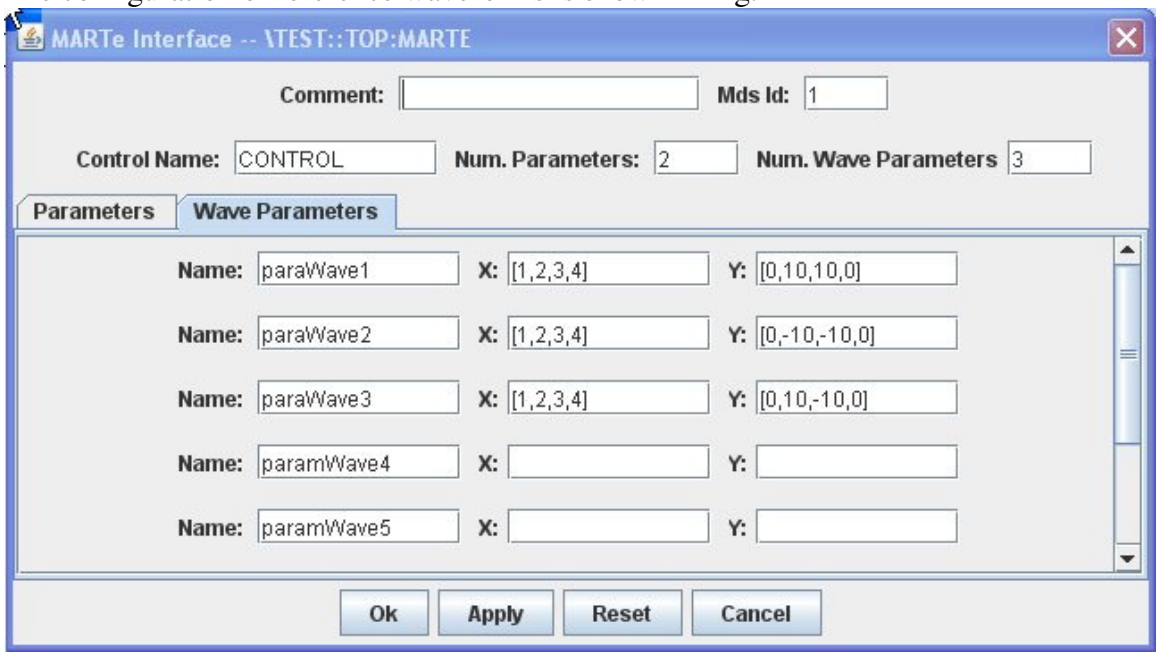

Figure 2: Wave parameter definition for MARTE_DEVICE

In this case, in addition to the name, two arrays are defined. The X and Y array define the times and the values of the reference waveform, respectively.
During real-time computation signal produced run-time by MARTe are stored in the corresponding device instance. Signals are defined within the device subtree as shown in Fig.3 .

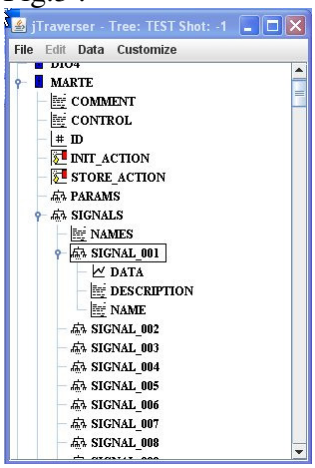

Figure 3: Signals in the MARTE_DEVICE subtree

The device makes room for up to 256 signals, and every signal is identified by the actual data, the associated name and (optional) description. The name of the signals depends on the current MARTe configuration and therefore is not defined in the pulse file. Just before entering in real-time operation, the names and descriptions declared in the MARTe configuration are reported in the pulse file. In order to simplify the access to the right signal, given its name, the MDSplus Function

MarteGetUserSignal(<device root>, <Signal name>)

Is provided, where <device root> is the path name of the root of the associated device, and <signal name> is the signal name as declared in the MARTe configuration. This function can be used, for example, in the configuration of the MDSplus jScope tool for the visualization of the acquired waveforms.

Integration of MDSplus functionality into MARTe is carried out by two MARTe components, namely the ***MDSInterface*** service and the ***MDSDriver*** GACQM.

3. MDSInterface MARTe Service

The MDSInterface service must be always declared in the MARTe configuration file whenever MDSplus functionality is involved. Following is a sample declaration for the MDSInetrface service in the MARTe configuration file

```
…
+MDSINTERFACE = {
    Class = MDSInterface
    ExecuteOnCpu = 2
    TargetStateMachine = StateMachine
    MdsEvent = "MARTE"
}
+StateMachine = {
    Class = StateMachine
……
```

Parameter **`ExecuteOnCpu`** specifies the core to be used for storing in run-time signals generated within MARTe into the MDSplus pulse file. A separate thread is in fact activated by the MDSInterface service which receives enqueued requests from real-time threads whenever data samples must be stored in MDSplus. The enqueuing of the requests ensures that real-time threads are not affected by delays in (possibly remote) write operation, provided, of course, that the average time for storing signals does not exceed the cycle time in MARTe real-time computation.
Parameter **`TargetStateMachine`** is used by the service to identify the StateMachine MARTe component. It is in fact possible to let the MDSInterface service advance the MARTe State Machine (in case, for example, the experiment sequence is driven by the MDSplus sequence supervision tools).

Parameter **`MdsEvent`** defines the name of the MDSplus event to be used for the communication between the MDSInterface service and the MDSplus framework. MDSplus events represent asynchronous events which are handled in a publish-subscribe pattern and which can bring also data. The MDSInterface service registers as a listener for the specified event name and, based on the associated data, will carry out the specified actions. The actions which can be associated to events are the following:

- ***Configuration***: load the current parameter and reference waveform setting for the specified MARTE_DEVICE instance. The configuration specified in the target device is read by the MDSInterface service, but not made available to MARTe components yet. This event can be received more than once in the pulse preparation in case more than one MARTE_DEVICE instance are defined in the experiment model.

- ***Load***: make the currently received configuration available to MARTe components. This is achieved by getting the current MARTe configuration as a CDB object and enriching it with the parameter configuration received in the previous Configuration message(s). MDSInterface will scan the configuration for every MARTe component. Whenever a field named ***MdsId*** is found for any MARTe component, its configuration is enriched with all the pairs *(<Parameter name>, <Parameter value>)* defined in the device with the corresponding MdsId identifier. The enriched CDB is then passed to the MARTe engine which will trigger again the LoadSetup method for all MARTe components. Therefore no change in any MARTe component code (GAM, GACM or Service) is required in order to let it receive parameters from MDSplus. It suffices adding a configuration parameter of the form MdsId = id and to define the desired parameter names and values in the MDSplus device instance with the same id identifier.

- ***Store***: flush signals to the MDSplus database. When MARTe is in real-time execution, signal samples are enqueued and stored in local buffers. Whenever the buffer size exceeds a given threshold, a separate thread writes that buffer as a MDSplus segment in the target data item. Therefore segments are periodically added in the pulse file during MARTe real-time execution. However, when the pulse is terminated, some buffered data may be not yet stored in the pulse file. This event, generated at the end of the pulse fill force flushing in pulse files data possibly still buffered.

The above three events are generated by three methods associated (***init, load, store***) to the MARTE_DEVICE instances. Suppose that in a given experiment model two instances of MARTE_DEVICES are defined with MdsId parameter, say, equal to 1 and 2, respectively. Then a possible sequence of device method calls could be the following:

1) Call **init** method for the first and second device instances, respectively. Two Configuration events will be generated and received by the MDSInterface MARTe service, which will receive the corresponding parameter configuration;

2) Call **load** method for any device instance (only one in this case), thus forcing MARTe to reload the current configuration;

3) At the end of the pulse, call **store** method for the first and second device, respectively to force the flushing of buffered signals.

Optionally it is possible to use the MDSInterface service to advance the MARTe State Machine. In this case, the following state machine configuration is assumed:

```
+StateMachine = {
    Class = StateMachine
    VerboseLevel = 10
    +INITIAL = {
        Class = StateMachineState
        StateCode = 0x0
        +START = {
            Class = StateMachineEvent
            NextState = IDLE
            Value = START
            +STARTALL = {
                Class = MessageDeliveryRequest
                Sender = StateMachine
                Destinations = "HTTPSERVER MARTe"
                MsecTimeOut = 1000
                Flags = NoReply
                Message = {
                    Class = Message
                    Content = START
                }
            }
        }
    }
    +IDLE = {
        Class = StateMachineState
        StateCode = 0x500
        +PRE_REQ = {
            Class = StateMachineEvent
            Code = 0x701
            NextState = PRE_PULSE
            +NOTIFY = {
                Class = MessageEnvelope
                Sender = StateMachine
                Destination = MARTe
                +MESSAGE = {
                    Class = Message
                    Content = PREPULSECHECK
                }
            }
        }
        +IDLE_REQ = {
            Class = StateMachineEvent
            Code = 0x778
            NextState = IDLE
        }
        +ABORT = {
            Class = StateMachineEvent
            Code = 0x704
            NextState = SAMESTATE
        }
    }
    +PRE_PULSE = {
        Class = StateMachineState
        StateCode = 0x504
        +PULSE_REQ = {
            Class = StateMachineEvent
            Code = 0x708
            NextState = PULSE
            +NOTIFY = {
```

```
            Class = MessageEnvelope
            Sender = StateMachine
            Destination = MARTe
            +MESSAGE = {
                Class = Message
                Content = PULSESTART
            }
        }
    }
    +ABORT = {
        Class = StateMachineEvent
        Code = 0x704
        NextState = IDLE
        +NOTIFY = {
            Class = MessageEnvelope
            Sender = StateMachine
            Destination = MARTe
            +MESSAGE = {
                Class = Message
                Content = PULSESTOP
            }
        }
    }
}
+PULSE = {
    Class = StateMachineState
    StateCode = 0x505
    +ABORT = {
        Class = StateMachineEvent
        Code = 0x704
        NextState = IDLE
        +NOTIFY = {
            Class = MessageEnvelope
            Sender = StateMachine
            Destination = MARTe
            +MESSAGE = {
                Class = Message
                Content = PULSESTOP
            }
        }
    }
    +POST_REQ = {
        Class = StateMachineEvent
        Code = 0x709
        NextState = POST_PULSE
        +NOTIFY = {
            Class = MessageEnvelope
            Sender = StateMachine
            Destination = MARTe
            +MESSAGE = {
                Class = Message
                Content = PULSESTOP
            }
        }
    }
}
+POST_PULSE = {
    Class = StateMachineState
    StateCode = 0x507
    +COLLECTION_COMPLETED = {
        Class = StateMachineEvent
        Code = 0x703
        NextState = IDLE
        +NOTIFY = {
            Class = MessageEnvelope
            Sender = StateMachine
            Destination = MARTe
            +MESSAGE = {
                Class = Message
                Content = COLLECTIONCOMPLETED
            }
```

```
            }
        }
        +ABORT = {
            Class = StateMachineEvent
            Code = 0x704
            NextState = IDLE
            +NOTIFY = {
                Class = MessageEnvelope
                Sender = StateMachine
                Destination = MARTe
                +MESSAGE = {
                    Class = Message
                    Content = COLLECTIONCOMPLETED
                }
            }
        }
    }
}
```

Device methods ***pre_req***, ***pulse_req***, ***post_r***eq and ***collection_complete*** are used to let the MARTe State Machine advance (by generating the appropriate events to the MDSInterface service). This approach is useful whenever MDSplus is used also to supervise the experiment execution using the jDispatcher and jServer tools.

4. The MDSDriver GACQM

This GACQM provides the required MARTe driver functionality to interact in real-time with MDSplus in order to generate reference waveforms and to store generated signals.

To generate reference waveforms, MDSDriver will be connected to an InputGAM instance. Following is a sample declaration of MDSDriver in the *DriverPool* section of the MARTe configuration for the generation of reference waveforms.

```
+MDSIO_IN = {
    Class = MDSDriver
    MdsId = 1
    NumberOfInputs = 3
    ReferenceWaveNames = { paraWave1  paraWave2  paraWave3 }
}
```

Parameter **`MdsId`** is required to associate the target instance of MARTE_DEVICE in the pulse file. Parameter **`ReferenceWaveNames`** defines the names of the waveform parameter definition as specified in the target MARTE_DEVICE instance (in the example above, the names used in Figute 2 are defined).

To store signals to be saved in pulse files during MARTe real-time operation, MDSDriver will be connected to an OutputGAM instance. Following is a sample declaration of MDSDriver in the DriverPool section of the MARTe configuration.

```
+MDSIO_OUT = {
    Class = MDSDriver
    MdsId = 1
    NumberOfOutputs = 3
    OutSignalNames={signal1 signal2 signal3}
}
```

Parameter **`MdsId`** is required to associate the target instance of MARTE_DEVICE in the pulse file. Parameter **`OutSignalNames`** defines the names of the generated signals. This name is only defined in the MARTe configuration file and will be reported in the pulse file.

5. Http Interfaces

Both MDSInterface and MDSDriver implement *ProcessHttpMessage()* method and therefore can offer a Web interface within the MARTe (provided the corresponding instances have been added to the *AddReference* field of the WEB definition in the configuration file).

Figure 4 shows a sample Web interface of the MDSInterface service.

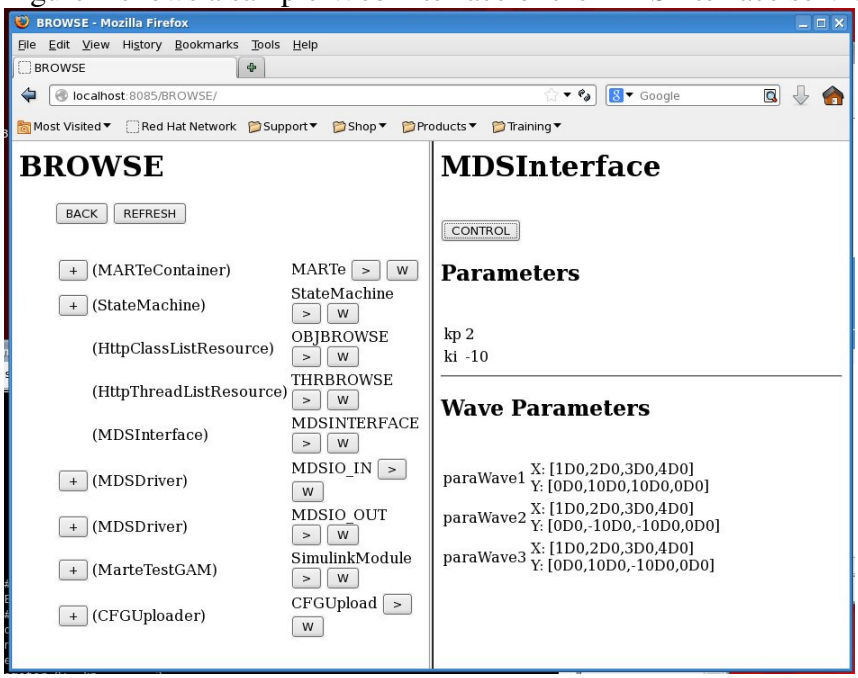

Figure 4: a sample Web interface of MDSInterface

In this example, only one instance of MARTE_DEVICE has been defined in the pulse file. In case more than one device were defined, additional buttons would be displayed, whose name corresponds to the Control Name field of the associated device. When clicked, the corresponding parameters and wave parameters are shown.

The Web interface of MDSDriver instances displays the names and the current values of the reference waveforms (input) and of the generated signals(output).