GSI

Detailed Design Specification for the SHIPTRAP Control System

Version 1.0

Distribution list:

Name (alphab.)	Department
Faouzi Attallah	Atomic Physics/GSI
Wolfgang Quint	Atomic Physics/GSI
Gerrit Marx	Atomic Physics/GSI
Christophor Kozhuharov	Atomic Physics/GSI
Klaus Poppensieker	DVEE/GSI
Dietrich Beck	DVEE/GSI
Holger Brand	DVEE/GSI
Jürgen Neumayr	SHIPTRAP/Munich

Author: Dietrich Beck Dept.: DVEE Name: Dietrich Beck Tel.: 2520 Signature:	**Inspector:** Klaus Poppensieker Dept.: DVEE Name: Klaus Poppensieker Tel.: 2782 Signature:
File: E:\TEMP\FrontPageTempDir\detspec.doc Date: 6 December 2001	Status: <doc status> File: Project file Sect. <nn.nn> File directory <directory>

Document Management

History of changes

Version	Status	Date	Person resp.	Reason for Change
0.0	created	6. Dec 2001	Dietrich Beck	new

Persons authorized to make changes

Dietrich Beck	DVEE/GSI
Klaus Poppensieker	DVEE/GSI

Document created using the following tools:

WINWORD2000

Contents

1 Introduction

1.1 Purpose of the document

The purpose of a detailed design specification is to define how the product components described in the associated architectural design specification are (to be) implemented. Moreover, these design specifications will be updated during the implementation of the control system and do finally serve as a documentation of the software.

1.2 Validity of the document

This documents describes all components of the SHIPTRAP control system. However, the emphasis is on the general part of the control system that can be used by other experiments as well.

1.3 Definitions of terms and abbreviations

- AFG: arbitrary function generator
- Device: a HW device like an AFG
- Device Driver: an independent process that is the interface of the control system to the instrument driver of a device.
- DSC: Data logging and Supervisory Control (the DSC Module is an additional module for LV replacing BridgeVIEW)
- HW: hardware
- NI: National Instruments
- OPC: OLE for Process Control: By this, lots of data items (that have simple types) can be exchanged between machines of distributed systems. Typical examples are values for voltages, DIGI I/O, …
- Static parameters: In general, “static parameters” are the parameters that can straightforwardly be implemented with the help of OPC servers.
- SW: software
- VI: virtual instrument

1.4 Relationship with other documents

- Architectural design specification which forms the basis for this detailed design specification and describe the general idea
- Software requirements specification
- Preliminary requirements and proposed solution

The documents above can be accessed via the SHIPTRAP control system home page, http://www-wnt.gsi.de/shiptrap/.

2 Core of the Control System

The core of the control system is formed by three classes. The BaseProcess, the SuperProc and the DSCIntProc class. The BaseProcess class is _THE_ base class of the system. The SuperProc and the DSCIntProc are children of and inherit from the BaseProcess class.

2.1 BaseProcess Class

The BaseProcess class is the basic class of the system and inherits from classes that come with the G++ toolkit. It has the following features.

1. Individual event, periodic action and state machine loops (three threads)
2. Watchdog (event and periodic action loop)

3. Communication between processes via calls
    a. Simple (one way)
    b. Synchronous (wait for answer)
    c. Asynchronous (answer will be sent later)
4. Trending and alarming via the DSC interface process
5. Parent class for ALL other processes. Child classes add new events, attributes and methods

2.1.1 Event, Periodic Action and State Machine Loops

These three loops serve to decouple different types of action. The event loop is used to receive events and calls from other processes. The action performed within the event loop should not be time consuming. Time consuming things should be performed in the periodic loop. The periodic loop is also used for action that has to be performed periodically, like reading a value from a ADC.

2.1.2 Watchdogs

The watchdog for the periodic loop of a process is checked by the event loop of the same process. The watchdog of the event loop is checked by another process in regular intervals. For this, the BaseProcess class provides a Ping method which is used to check the event loop via a synchronous call.

2.1.3 Calls

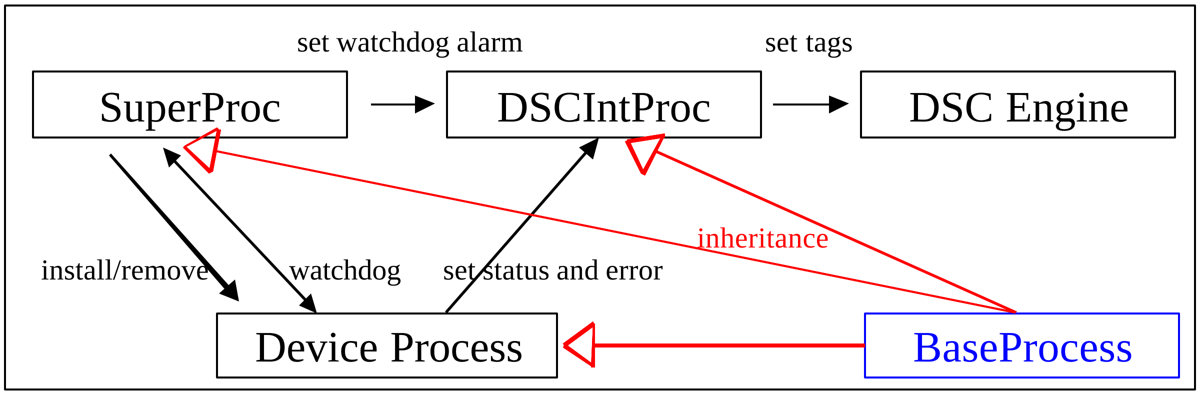
Calls can be simple, synchronous or asynchronous.

1. Simple call. The caller calls the callee. Neither does the caller wait for an answer, nor answers the callee.
2. Synchronous call. The caller calls the callee and waits for the answer of the callee. The callee receives the call, performs the required action, and answers the call of the caller. The thread of the caller will only proceed, after having received the answer or a timeout.
3. Asynchronous call. The caller calls the callee but does not wait for an answer. The callee receives the call, performs the action required, and sends the answer as a simple call to a receiver that has been specified by the original caller.

A LV cluster is used and must be filled with the information by the caller. The call cluster is defined in the Defs library (see sect. 3.4.2)

2.2 Relationship between Instances

The relationship between instances of these classes is depicted below. To make it more clear, there is a device driver name DeviceProcess.



The SuperProc installs and removes the DeviceProcess on request by other processes. The Device Process sets its status and error tags in the DSC Engine via the DSCIntProc. If the watchdog for the periodic loop is in an alarm state, the watchdog tag in the DSC Engine is updated via the DSCIntProc. The watchdog of the event loop is checked by the SuperProc. On a periodic basis, it communicates with the DeviceProcess by calling its Ping function to check the event loop of the DeviceProcess. If that fails, a watchdog alarm is set via the DSCIntProc.

All other classes for other processes (except the GUIs) will be child classes of the BaseProcess class.

3 Implemented Components

The general idea of the interplay between the components is described in the architectural design specifications. Here are the detailed design specification of the components that should also serve as a documentation.

3.1 BaseProcess Class

3.1.1 Purpose

The BaseProcess class serves as parent class for all other classes in the control system, except for GUI classes. It provides the methods for communication (see sect. 2.1.3), setting error and status, an LabVIEW object.vit template and the watchdog functionality (see sect. 2.1.2).

3.1.2 Functions

The BaseProcess class inherits from system classes of the G++ toolkit. Only public methods that are accessible via calls and that have been added are listed. Methods that have already been accessible via calls of system classes from the G++ toolkit are not listed. The names in the list below are the names of the defined events of the BaseProcess class.

1. S0. Change to the default state S0.
2. Ping. A function that does nothing and is used for the watchdog of the event loop.
3. GetDescriptors. Gets a list of the event names together with the descriptors of the data type for the events.
4. SetPdcInterval. Sets the interval of the periodic loop.
5. SetPdcPreset. Sets the preset value for the periodic loop. This value is the time interval used by the watchdog of the periodic loop.
6. SetEvtPreset. Sets the preset value for the event loop. This value is the time interval used by the watchdog of the event loop.
7. GetEvtStatus. Gets a list of the event names together with their URLs and the enable bit.
8. GetCNames. Gets a list containing the class name of the instance together with the class names of the parent classes.
9. GetIList. Gets a list containing the names of all instances of this instance class type.
10. Reset. This is the reset function. The corresponding reset method is just a dummy method that must be filled by child classes.

3.1.3 Non-functional features of the component

See also sect. 2.1. When inheriting classes from the BaseProcess class, the object.vit template should not be changed by child classes. Instead, the four methods ProcEvents, ProcCases, ProcPeriodic, ProcState and Reset should be changed for the child classes.

1. ProcEvents. Method that serves to define the events for the child classes.
2. ProcCases. Here, the methods that correspond to the defined events will be called. ProcCases is called by the event loop each time the instance receives a valid event.
3. ProcPeriodic. Here, functionality that should be performed in periodic intervals can be inserted. ProcPeriodic is called by the periodic loop in intervals defined by PdcInterval.

4. ProcState. Here, functionality that should be performed within the state machine loop can be inserted. ProcState is called by the state machine loop.

3.1.4 Implementation

The BaseProcess class is a child of the System.EvScObj class of the G++ toolkit. The watchdog for the event loop requires an instance of the SuperProc class. Instantiation should be done by an instance of the SuperProc class. Instances of the BaseProcess Class **and all child classes(!)** should get the BaseProcess.i attribute.ctl data. This LV cluster contains the following information.

1. status and error of the event loop.
2. status and error of the periodic loop.
3. status and error of the state machine loop.
4. a list of interfaces together with a list of interface addresses.
5. a list with special parameters.
6. the name of the DSCIntProc instance where status and errors will be sent to.
7. the name of the Super class that created the object.
8. the timeout for the event loop.
9. the preset value for the watchdog of the event loop.
10. the interval for the periodic loop.
11. the preset value for the watchdog of the periodic loop.
12. the alarm time for the watchdog of the periodic loop. If the periodic loop has not acted when this alarm time is reached, the watchdog alarm bit of the periodic loop is set to true.
13. the watchdog alarm bit of the periodic loop.
14. the timeout for the state machine loop.

3.1.5 Reason for detailed design decisions

The design has some ideas of a device driver of the ISOLTRAP control system. Added are the object oriented approach, the periodic loop, the state machine loop and the watchdog mechanism. The watchdog for the periodic loop is only checked internally by the event loop in order to save system resources. Thus, one must enable the watchdog for the event loop to guarantee the functionality of the watchdog of the periodic loop.

3.1.6 Technical implementations conditions

The implementation depends on LabVIEW version 6.0.2 or later and the G++ toolkit from Vogel Automatisierungstechnik. There should be no dependency on the operating system. Each instance of the BaseProcess class requires a few hundred kByte of RAM. About 50 instances of the BaseProcess class (together with one instance of SuperProc class (watchdog for event loops) and one instance of the DSCIntProc class) use about 10% CPU time on a 700MHz Pentium III. The typical rate for synchronous “Ping” events (see sect. 3.1.2) on the same machine is about 150Hz.

3.2 SuperProc class

3.2.1 Purpose

With the SuperProc class one can instantiate and destroy instances of other classes. An instance of the SuperProc class maintains a list of active instances “process table” and serves as a watchdog for instances of other classes.

3.2.2 Functions

The SuperProc class is a child of the BaseProcess class. Only public methods that are accessible via calls and that have been added are listed. The names in the list below are the names of the defined events of the SuperProc class.

1. EvtPreset. Sets the value of the event preset of an instance. That instance must have an entry in the process table. The event loop of that instance will be “pinged” within that preset value.

2. LoadProcess. Creates an instance of a class and adds it to the process table. If the instance is existing, not action is undertaken.
3. UnloadProcess. Destroys an instance of a class and removes it from the process table. The instance is only destroyed if no other process has an open connection to that instance.
4. RemoveAll. Removes all instances that are listed in the process table even if there are open connections by other processes.
5. ConnectProcess. Establishes a connection from instance A to instance B. “UnloadProcess” of instance B can not be done unless the connection is removed.
6. DisconnectProcess. Removes a connection from instance A to instance B.
7. Disconnect All. Removes all connections to all instances that are listed in the process table.
8. GetProcTable. Gets a copy of the process table.

3.2.3 Non-functional features of the component

When creating an instance “LoadProcess”, the SuperProc class must supply the BaseProcess.i attribute.ctl data. That data is obtained from a database (see sect. 3.7).

3.2.4 Implementation

The SuperProc class is a child of the BaseProcess class. When creating an instance of the SuperProc class, an instance of the DSCIntProc class is created as well (it is not clear whether this will also be done in the future). When destroying an instance of the SuperProc class all instances listed in the process table will be destroyed. The Process table is a list that has entries of a type defined in the Defs library (see sect. 3.4.2).

3.3 DSCIntProc Class

3.3.1 Purpose

This class provides an interface to the DSC engine. Setting values in the DSC engine should only be done via an instance of this class.

3.3.2 Functions

The DSCIntProc class is a child of the BaseProcess class. Only public methods that are accessible via calls and that have been added are listed. The names in the list below are the names of the defined events of the DSCIntProc class.

1. DSCStart. Starts the DSC engine.
2. DSCStop. Stops the DSC engine.
3. SetEventError. Sets the error of the event loop of a process.
4. SetPeriodicError. Sets the error of the periodic loop of a process.
5. SetStateError. Sets the error of the state machine loop of a process.
6. SetEventStatus. Sets the status of the event loop of a process.
7. SetPeriodicStatus. Sets the status of the periodic loop of a process.
8. SetStateStatus. Sets the status of the state machine loop of a process.
9. SetProcInstalled. Sets the status and errors to default for an created instance.
10. SetProcRemoved. Sets the status and error to default for a destroyed instance.
11. SetValue. Sets the value of a tag in the DSC engine.
12. GetValue. Gets the value of a tag in the DSC engine.
13. SetEvtWatchdog. Updates the tag for the watchdog of the event loop of a process.
14. SetPdcWatchdog. Updates the tag for the watchdog of the periodic loop of a process.

3.3.3 Implementation

The DSCIntProc class is a child of the BaseProcess class. The DSCIntProc.i attribute.ctl gives information about the status of the DSC engine and the name of the *.scf file used.

3.3.4 Reason for detailed design decisions

The DSCIntProc was implemented such, that the DSC engine can easily be replaced by some other database or server like a DataSocket server.

3.4 Defs Library

3.4.1 Purpose

Provide definitions of data types that are used within the SHIPTRAP control system. The data types are defined by LV strict typed controls. Given below are the names of the typedefs.

3.4.2 Data Types

1. Defs.call_cluster.ctl. The Cluster that is used when calling another process via the call process.vi.
    1. the destination object (callee)
    2. an error number
    3. a data descriptor (only needed when the data type changes during run-time)
    4. the data that is flattened to string
    5. type of call (simple, synchronous or asynchronous)
    6. event type and receiver for an asynchronous answer.
2. Defs.descriptor_type.ctl. The cluster defining the element type of the list that is obtained when calling and instance of the BaseProcess (or child) class with the GetDescriptors event.
3. Defs.error_type.ctl. An enum defining errors that are frequently used.
4. Defs.proc_table.type.ctl. A cluster defining the element type of the process table used by the SuperProc class.
    1. name of the process
    2. # of connections to that process
    3. names of the connected clients
    4. the alarm time for the event loop (when the alarm time is reached, the event loop of that process will be checked. After this check the next alarm time is set be adding the preset value for the event loop watchdog.
    5. the preset value for the event loop watchdog
    6. busy bit. True: event loop of the process is not responding. False: event loop is responding
    7. reference to the process

3.5 ProjLib Library

3.5.1 Purpose

The ProbjLib library is a collection of Vis that are used within the project and that do not really fit into a class.

3.5.2 VIs

Given below are the names of the Vis that are part of the ProjLib

1. ProjLib.call process. Calls another process. This VI should only be used by classes that are not children of the BaseProc class.
2. ProjLib.fill baseproc attrib. Converts BaseProcess.i attrib data in query format (see below: ProjLib.sql read) into BaseProcess. I attrib.ctl format.
3. ProjLib.get dscint proc. Gets the name of the DSC interface process of an instance
4. ProjLib.get error string. Gets an error string to an error number.
5. ProjLib.install process. Creates an instance. This VI should only be used by an instance of the SuperProc class. Comment: When creating a new class, this VI must be modified.
6. ProjbLib.my to LV error. Converts an error number as given by the Defs.error_type.ctl (see sect. 3.4.2) to a valid LV error number.

7. ProjLib.remove process. Destroys an instance. This VI should only be used by an instance of the SuperProc class.
8. ProjLib.sql read. Reads the information necessary to instantiate an object out of a data base. This is mainly the information of the BaseProcess.i attribute cluster. The information is given in a query format that can be converted into BaseProcess.i attribute.ctl format by using the ProjLib.fill baseproc attrib.vi.
9. ProjLib.variant to call cluster. Converts variant type data into a data type as defined in Defs.call cluster.ctl (see sect. 3.4.2).

3.6 Communication Interface

3.6.1 Purpose

Calls processes on remote machines. The communication interface makes calls of processes on remote nodes completely transparent as if the remote processes were on the local host.

3.6.2 Non-functional features of the component

The communication interface must be implemented such, that the reaction times for the CALL_PROCESS command are as fast as possible. Furthermore, the communication interface must be capable of serving more than one request at a time so that calls do not block each other.

3.6.3 Implementation

Presently, the communication interface is implemented via the VI server from NI. This is an extremely elegant approach since it does not even require additional processes on the local or remote machine but just an extension of the Bas eProcess.call process.vi. The big disadvantage is, that this is very slow (1-2Hz event rate). The present implementation is just a first solution that must be replaced on the long term.

3.6.4 Alternatives

1. Implementation with LV. TCP/IP will be used for the communication between different PCs. This is quite some work but can be fast (50-100Hz event rate)
2. Use the universal license of the G++ toolkit from Vogel Automatisierungstechnik. (100Hz event rate).
3. LabVIEW 6.1 may possibly have the feature to insert queue elements into queues of remote machines. In this case, the communication interface may become redundant.

3.7 Database

3.7.1 Purpose

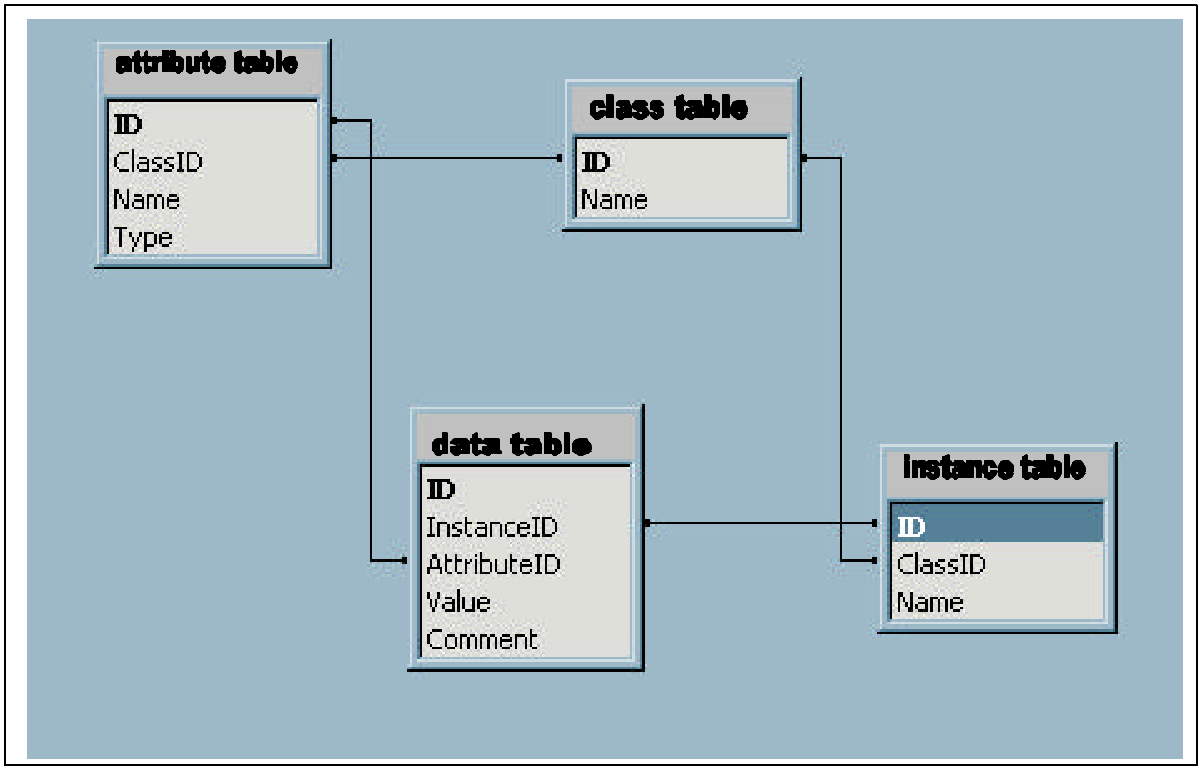
Each instance of the BaseProcess class or its child classes need to get information via the BaseProcess.i attribute.ctl cluster when created. This information is available via the database and can be retrieved with the ProjLib.sql read.vi (see sect. 3.5.2). Also the class name of an instance is needed for instantiation and is available via the database.

3.7.2 Design

The database has four tables.

1. instances
2. classes
3. attributes
4. data

The instance table contains the name of an instance, an instanceID and the class name of that instance. The class table contains the name of a class and an classID. The attribute table contains the attribute name, an attributeID, the classID to which the attribute belongs and the data type of that attribute (Note: So far, there are only the BaseProcess class attributes, since these are the attributes that are needed for instantiation of objec ts of the BaseProcess class and its child classes). The data table contains a dataID, the attributeID, the instanceID, the value and a comment. The relationships between the tables are shown below.



Assume, one would like to retrieve the information. First, one needs the class name. For this one has to search for the name of the instance in the instance table and gets its classID. With the classID one can search in the class table for the name of the class. Then one needs the data to fill the BaseProcess.i attribute cluster. From the instance table, we get the instanceID. Then one can look in the data table to find all entries for the instance via the instanceID. From the data table we get the values and the attributeID. From the attribute table one can then get the attribute name and its type via the attributeID.

3.7.3 Implementation

MS Access was choosen as a database, since it is available almost everywhere and gives easy access to edit the data. The data are only retrieved programmatically via the ProjLib.sql read.vi. The database is modified via "forms" in MS Access. Like this, it is not necessary to lock the database for different instances of the SuperProc class since it is never changed during run-time but only off-line by developers. The database is accessed via the SQL toolkit from NI.

3.7.4 Technical implementations conditions

In the present implementation, one is restricted to MS Windows. Since the database is accessed via SQL, one can easily switch to another database like Oracle, if desired.

3.8 DSC Engine

3.8.1 Purpose

Alarming, trending, safety. The processes/functions do error handling and status reporting via the DSC engine. The DSC engine provides complex process variables, the tags. The tags are also used by devices with static parameters that use the "wait for changed value" mechanism. The DSC engine and its documentation is available from NI.

3.8.2 Reason for detailed design decisions

The DSC engine can be bought and costs no development time. Access to the DSC engine is done via an instance of the DSCIntProc class or via the *classname.*wait call.vi.

The DSCIntProc allows to set or get values of tags in the DSC engine. However, one can not use the feature of "waiting for a changed value" (see LV documentation for the read tag.vi's). If this is desired, one should define an OPC event by adding the tag to the event list in the *classname.*DefEvents.vi (see sect. 3.1.3). Then, the event loop will receive an event each time the value of the tag changes.

3.8.3 Alternatives

If desired, one can easily replace the DSC engine with a DataSocket server. The only thing that has to be changed are the low level read tag and write tag methods in the DSCIntProc class. However, one looses the alarming, trending and safety features of the DSC engine.

4 Not Yet Implemented Components

4.1 Control GUI

4.1.1 Purpose

User interface for the controls but not for the on-line analysis.

4.1.2 Functions

- LOGIN: login of a user
- LOGOUT: logout of a user
- CHANGE_MODE: switch between control and monitor mode. The GUI can only switch to control mode if a) no other GUI is in control mode or b) the user is logged in as an administrator
- CREATE_CONFIG: create configuration data for static and dynamic parameters
- INIT: sends the configuration data to the data server and the init command to the central process
- DEINIT: sends the deinit command to the central process
- START: sends the start command to the central process. The start command can only be sent if a) there are no pending alarms or b) as an administrator
- STOP: sends the stop command to the central process (finish the current scan and BREAK)
- BREAK: sends a break command to the central process (stop cycle and scan immediately)
- VIEW: display the current status and pending alarms of SHIPTRAP from the DSC engine
- ACKNOWLEDGE: acknowledge pending alarms
- EXIT: sends a deinit command to the central process and exits the control GUI
- SET_PARAM_VALUE: any value of a scannable parameter can directly be set at any time

4.1.3 Non-functional features of the component

The functions of the control GUI are not really time critical. On the other hand, it would be nice to have reaction times that permit to change a parameter value fast enough, so that the parameter can be tuned "by hand". All controls and parameters must not be by more than one mouse-click away. If an alarm is pending, appropriate information always visible and never hidden behind sub-windows.

4.2 On-Line Analysis GUI

4.2.1 Purpose

The on-line analysis GUI displays the data of the current measurement.

4.2.2 Functions

This is not really complete, since the requirements from SHIPTRAP are not clear yet.

- CONNECT: the on-line analysis GUI connects to a data server
- DISCONNECT: the GUI disconnects from the data server
- GET_DATA: the GUI gets new data from the data server
- UPDATE: the GUI updates its data buffer with the new data
- CLEAR: if a new measurement has started, the data buffer and the display are cleared
- DISPLAY: the GUI displays its data buffer
- EXIT: the GUI disconnects from the data server and exits

4.2.3 Non-functional features of the component

The on-line analysis GUI displays a predefined set of graphs. It can also do fits and things like that. Basically, the only interaction by the user is to chose the data server and to exit the GUI. Of course, some interaction is allowed to move graphs around and do some fitting and so on. The on-line analysis GUI knows when to clear the display and its data buffer from the change of a file name and when getting the data element with element number 0. The on-line analysis GUI is not time-critical.

4.3 Central Process

4.3.1 Purpose

A SHIPTRAP specific process that is the heart of the system and an image of the experiment. Loading/initializing/unloading of device drivers, scanning etc. are done here.

4.3.2 Functions

- INIT: get the configuration data for the dynamic parameters, loads device drivers, initializes the hardware, creates a data module via the data server
- START: changes the state of the central process to perform cycles and scans, starts first cycle
- CYCLE_FINISHED: a cycle has finished. Now, read out data, prepare and start next cycle
- DEINIT: unloads all device drives
- STOP: sets a stop flag. Cycling and scanning will stop on completion of present scan
- BREAK: cycling and scanning are stopped immediately
- EXIT: the central process exits

4.3.3 Non-functional features of the component

The requirements of the experimentalists are directly programmed into the central process. It is very similar to the massmeas.c of ISOLTRAP.

4.4 Device Driver

4.4.1 Purpose

A device driver is the interface between the instrument driver from LV and the control system.

4.4.2 Functions

- The functions depend on the functionality of the driver and the instrument. The device driver is an instance of a device driver class which is a daughter of the BaseProcess class. The device driver has all functions of the BaseProcess class and additional events and methods. Examples for additional methods for an AFG are “SetFrequency” and “SetAmplitue”.

4.5 Data Server

4.5.1 Purpose

Provides data buffer and a interface for transferring data between processes.

4.5.2 Functions

- CREATE_DATA: creates a data module
- DELETE_DATA: deletes a data module
- LINK_DATA: creates a “fixed” connection to data module
- UNLINK_DATA: terminates a fixed connection to a data module
- RESET_DATA: resets the data module to the default state (state after RESET)
- READ_DATA: reads the data elements of a memory module and marks them for deletion
- WRITE_DATA: write a data element(s) to the memory module
- EXIT: exits the data server

4.5.3 Non-functional features of the component

Every data module can just contain a certain number of data elements. The data can be retrieved with the READ_DATA function, which marks for deletion. The data elements will be deleted (oldest element first) if a) a new WRITE_DATA is executed, b) the buffer is full and c) the have been marked for deletion. The data server allows multiple connections which (with LINK_DATA) to a data module. Connections can be of type “hard” and “soft” (see Sect. **Error! Reference source not found.**).

4.6 Data Handler

4.6.1 Purpose

Storing of data to data storage device.

4.6.2 Functions

- CONNECT: the data handler connects to a data module of a data server
- WRITE_DATA: function is executed periodically. Retrieves data from data server and writes them to the storage device(s).
- CONFIG_DISK: configures the storage device
- EXIT: exits the data handler