1 Basics

1.1 Overall Design

Devices in the renovated control system will be realized as independent objects on the master processor (GµP). Each device (in the view of the operators of the facility) will be represented by a separate object. They can be accessed by unique names (nomenclatures).

The specifics of the devices are modeled by properties, which correspond to methods in object oriented software. Properties are identified and can be accessed by names. Properties interact with the connected hardware, at the present stage, via the existing device controller (EC) layer.

Access to the properties is by direct communication with the device objects, without central access point. The CORBA standard is chosen for networking. The existing Userface access mechanism is replaced by a wrapper around the CORBA access, providing the existing functionality.

Management of the device objects (creation, deletion) is done by a system process, the device manager (N-MOPS). It too provides the data for the operation of the EC control system components.

1.2 Notation

Access interface (Zugriffs-Schnittstelle):

Interface for access to devices in the control system

Realized as CORBA-interface (IDL-specification)

Device object (Geräte-Objekt):

Software-object in the control system which represents a device (primarily a hardware device) which can be handled to operate the accelerator facility.

Device objects are located on the GµP (MP) at first, but can be realized on other network computer too.

Each device object can be identified by an associated unique name, the nomenclature.

Device manager (Geräte-Manager, N-MOPS):

Management process for device objects, realized on the GµP.

Device manager is the system component which creates and deletes device objects and provides the EC layer of the control system with data needed for operation (like nomenclatures of the devices).

Device locator (Geräte-Lokator):

Management process on the application level.

The device locator finds and returns the (CORBA-) reference of a device object for a given nomenclature. The background of this mechanism is to hide name server from applications.

2 Requirements

2.1 Fundamentals

- The control system, as seen by the developer of control system applications, must not change significantly during the renovation process.
    - Functionality of the actual system has to be provided in the future too.
    - Using the control system must be possible in the same way like today.
    - Modifications in functionality or handling can be tolerated only when they outbalance the effort for adapting the existing applications.
- Extensions in functionality, are of course, possible.

        - Extensions should be limited, the project must not be complicated unnecessarily.
        - Extensions in functionality should thus be postponed to a later project.
- To skip a feature of the current control system and to not support it in future requires extreme care. It is most likely that a functionality provided by the current control system will be used somehow somewhere by some application.
    - Generally the need for a already forgotten feature appears when a crucial application no longer works correctly when it is used after some period. Too often this happens Friday afternoon, when the expert to correct it has left for the weekend.
- Care has to be taken to support also features which are not obvious at first sight. The likelihood of not supporting or, worse, of not being able to add an option which was forgotten during design, has to be reduced by using similar concepts as in the original system.
- Concepts to be kept comprise:
    - A generic (narrow) interface has to be provided for device access.
    - All data items which relate to the pulse-to-pulse modulation must be identified by the so called number of the virtual accelerator (actually 0..15).
    - The number of the virtual accelerator has to be provided in each device access.
    - Each data item, read by a device access, provides so called stamps to identify the data set.
        - Stamps comprise at least cycle time and event pattern.
    - The actual rejection mechanism of unauthorized device access, based on device type and VME node where the device is hosted, should be supported in future.
        - Additionally, more sophisticated methods can be implemented.
    - Error handling: Execution status of device access can be concentrated to one major error message and one optional (default value is OK) secondary error message.
        - Both, major (primary) and secondary error messages are in the OpenVMS message code format.
- Focus has to be on portability and modularity.
    - It should be possible, with moderate effort:
        - Port the system to other hardware platforms, other operating systems.
        - Exchange components, even replace CORBA by some other mechanism.
    - To achieve this, care should be taken to
        - Encapsulate interfaces that depend on implementation details
            - Do not use CORBA interface directly in applications, use a wrapper instead
            - Use control system data types (use typedef) rather than C++ types or CORBA types, at least in applications or device specific software.
        - Never use unions in interfaces, unless correct evaluation of the contents is guarantied for both sides of the interface (e.g. like CORBA handles unions).
        - Avoid complex structures in interfaces. Use arrays of basic types instead.
            - If different types have to be mixed, provide means for conversion of the used types, e.g. from little to big endian.
        - Data used in Interfaces which may some day connect different hardware platforms must contain information of the coding (little/big endian) used in writing.

2.2 Device Access

- All existing properties of all devices have to be provided in the future too, each property in the current

system must be represented by some item in the system after the renovation.

    - Properties of system components which are used for system purposes only may be modified, like MOPS's DOWNLOAD property.

- It must be possible to map each property in the current system to some corresponding property in the future system, providing the same functionality.
- Equivalents of the existing asynchronous access methods, especially connected commands, must be provided.
- Devices and Properties must be accessible by the existing 8-character naming scheme, plus an access class indication (i.e. read/write/call).
- Providing means for using mixed data typed in device access, e.g. combining floats and ints, is highly desirable.
- Each actual data item, which depends on the pulse-to-pulse switching, must be identified by stamps, i.e. cycle time and event-pattern.
- An access rights management is required. Functionality has to cover the functionality of the existing access rights management:
    - It must be possible to restrict user's requests to selected equipment models or selected VME nodes.
    - It must be possible to adjust, for each equipment model, on a property base whether it is affected by the rights management or whether free access is allowed. The default setting is to allow all users free access to read properties and to request special rights to access write and call properties.

2.3 Software Implementation

- It must be possible to re-use existing device specific software, that is the existing USRs, without significant modifications.
    - It must be possible to adapt the existing code of the USRs for re-use without understanding implementation details.
    - Adaption must be possible by executing some formal recipe, maybe even automatically by some script.

3 Implementation

3.1 Front-End Software

3.1.1 Device Manager

- The device manager is a (the) system process on the VME master processor
    - At least one instance of the device manager has to run on each VME master processor.
    - It has to be checked whether using several device managers on each VME master processors may be advantageous (i.e., one device manager per EC or one per device type)
- The main task of the device manager is to handle the device objects, that is
    - Create and delete device objects
    - Provide initial data to the objects (device specific constants)
    - Register the device objects to the name service
    - Record lists ot the existing device objects, with name and type, which can be read from the device manager.
- The device manager has to keep the device object's status up to date. This means

    - Monitor the SE’s status and the device SE’s device list.
        - SE’s actions are monitored by heart beats from the SEs.
        - Appropriate actions have to be done when a SE’s heartbeat is missing (“offline”)
    - Create a device object for devices newly found on a SE
        - Do consistency checks, e.g. for the device type.
    - Update all device objects of a SE when the SE performs a cold reboot.
        - E.g. Update pointers to the device data in DPR
        - Collisions have to be avoided with actually performed device access, e.g. Pointers used in execution of a device access should not be modified during the device access. Appropriate synchronization should be done.
- The device manager is not involved in the communication with the device objects, accessing a device object is by direct communication with the device object.
- The device manager provides the EC tier with the data needed for operation.
    - Data needed by the EC tier are mainly the nomenclatures for the devices found.
- The device manager handles the data needed to manage the devices on the node. These data, which correspond to the devices and constants tables of the actual VME data base, comprise
    - Device names, device types, initial data (device specific constants), ...
    - Data for the EC tier, like physical device number.
- Some management tool supplies the device data for the VME master processor.
    - This tool corresponds to the actual dbsgen tool.
    - Data should be derived automatically from the existing dbsgen input files, while in future other mechanisms to handle the source data may be used.
- The device manager is involved in the alarm handling.
    - If promotes alarms send by the SEs.
    - Changes in the configuration which affect the device manager have to be signaled by alarms.

3.1.2 Device Objects

- Each device object is associated with (has) a unique name, the nomenclature.
- Device objects are accessed from the application level directly, that is without involving some central instance like the device managers
    - The access interface for the application layer is part of the device objects
    - The device object's access interface handles asynchronous access too, e.g. connected commands.
- Device objects model the specifics of the devices (generally the hardware) to be controlled.
    - The specifics of the device are represented by a set of properties.
- Implementation of the properties:
    - Properties are implemented as independent objects, attached to the device objects.
        - All properties of the same access class (read, write, call,...) implement a common interface, an execution method which will be called when a device access is issued.
        - Calling the property’s execution method is by a dispatcher in the device object.
    - Alternatively: Properties could be realized as methods of the device objects, or, as in the actual implementation, as procedures acting on the device objects.

        - This rather procedural attempt will be used only if an object oriented implementation could not be realized.
- Existing USR code is modified to form the major part of the device specific code for the implementation of the properties.

3.2 Device Access

3.2.1 Access Interface

- The access interface will be implemented as a semi generic (semi narrow) interface.
    - Each device object has some access methods (read, write, ...) by which the complete functionality is available.
    - Selection of the functionality is done by a parameter in the access methods, based on the name of the property.
    - This is, the access interface is neither fully specific (no pure wide interface) nor fully generic (no pure narrow interface), it is a semi generic interface:
        - Each device is accessed directly, this is specific.
        - The complete functionality of one device object, i.e. all properties, is accessed by few interface methods in which the property has to be selected, this is generic.
- The access interface of the device objects provides also the existing asynchronous access methods:
    - Asynchronous single access
        - Main use of asynchronous single access will be reading data.
        - Besides reading also write access and function calls have to be supported.
        - Asynchronous access, also for write and call, returns an execution status. For asynchronous write and call it should be possible to skip any response. In this case users will not be informed of errors during execution on the front end.
    - Periodically connected commands.
    - Event connected commands.
    - Time-of-day connected commands.
    - Intentionally the existing interrupt connected command will not be implemented again.
        - There is some hope that interrupt connected commands are used for IPS devices only. It has to be checked whether more general mechanisms can be implemented to support IPS functionality.
        - That is, some update mechanism which can be triggered from the EC layer. This update mechanism may depend internally on interrupts but should hide this form the application.
- An additional asynchronous method “modification connected” command” is foreseen
    - A response is send, when a value changes his value.
    - Check for modified values may be made periodically or once per execution of a virtual accelerator, that is once per accelerating cycle.
        - In case of once per execution of a virtual accelerator, the message rate has to be limited in environments with high repetition rates like the Unilac.
- No distinction is made between properties which access single data (class R, W) and multiple data (class RA, WA).
    - It has to be verified that no properties exists which differ only by single and multiple access class.

        - It had to be confirmed that no application depends on the class information.
            - Userface, for some years now, no longer depends on this information. If a property entry is not found, an entry with the other type is searched (RA/WA instead of R/W and vice versa).
            - With this extension of Userface, Nodal too no longer depends on this information.
        - Skipping the distinction between single and multiple data access means an alteration to the actual implementation. This alteration is accepted because of the benefits in clarity.
- Device access is realized by CORBA.
- Only one command will be specified in an device-access.
    - Some mechanism has to be implemented to handle the multiple commands supported by Userface.
    - This can be implemented completely in Userface.
    - Alternatively, a command dispatcher can be foreseen in MOPS.
        - The list of commands is passed to the MOPS command dispatcher which distributes the commands to its devices.
        - The commands are simply forwarded to the device object's access interface.
        - Commands are handled sequentially, in the same way as in the old MOPS's command dispatcher.
- The device access interface is implemented as part of the device objects.
    - Based on the name of the property., a dispatcher in the device object calls the execution method of the addressed property object.
    - The dispatcher evaluates tables in which the properties are registered.
- The CORBA device access interface is not provided directly for the application layer
    - A wrapper is used, to take care for a possible future change in the access mechanism which can not be excluded today. This would also ease a transfer to another CORBA based control system.
    - The existing Userface access routines are rebuild as a wrapper around the new device access.
- CORBA specific mechanism and items have to be kept away from the device specific software, that is the property implementation.
    - Some sort of shell may be used, if needed, to separate the device specific software.

3.2.2 Device Access Data

- Data is exchanged with the device objects by container objects
- These data container can transport different data types, like
    - Single long items, signed and unsigned
    - Single double items
    - Multiple char items (strings)
    - Multiple octet items (arrays of 8-bit data)
    - Multiple long items (int arrays), signed and unsigned
    - Multiple double items (double arrays)
    - Other types
- For greater clarity not all data types of the current Userface will be implemented, 8-bit and 16-bit

integers (both signed and unsigned) will be embedded in the corresponding 32-bit data.

- Support for mixed types has to be discussed still.
    - When using mixed data types:
        - Adding descriptive information about the sequence of types used is preferred about using fixed combinations of different types
- The data container also provides additional stamp informations
    - Stamps are only provided on reading the data, that is for out data.
    - Stamps comprise cycle time, the date (day/month/year) and event pattern.
    - Stamps are provided by the device specific software.
        - Access data is initialized with default stamps, which can be overwritten in the device specific software, to ensure proper data in all cases.
        - Defaults are date and time of the call and zero event pattern.
    - It has to be checked to use at least the number of the virtual accelerator also as stamp.
        - This would make sense also for in data.
- Data containers provide methods for data conversion. Data can be converted to other supported data types, e.g. int data to float and vice versa.
    - Data range is checked, trying to convert float values bigger than the maximum possible int value result in an error.

3.2.3 Access Rights

- Since access is no longer handled by some central service, access rights management has to be implemented in the device objects.
- Information about the user's access rights is provided by a list of rights for each user.
    - It has to be taken care that unique idents are used to identify the users.
    - The login name of a user may differ, at least for different operating systems.
- Using a client side wrapper around the CORBA device access interface facilitates implementation of the rejection of unauthorized device access.
- Access rights should be implemented in several levels.
    - A 'system' level should be foreseen which is restricted to a very limited group of persons which are responsible for proper operation of the control system.
    - The system-level access right is foreseen to clear unforeseen deadlocks in the system. It should not be available for the operators of the facility but for the persons in charge of the control system.
    - A separate level should be foreseen to give the operators full access to all devices in the facility.
- Carefully implemented access rights handling can then be used to restrict access to highly critical functionality to a carefully selected group of people. This high level of security is for example needed for interventions in the fully automated medical cancer therapy operations. Although operation is fully automated, interventions should be possible to be able to clear deadlock situations.

3.3 Other Aspects

3.3.1 Device Locator

- Purpose: Deliver the object reference of a device object to a nomenclature.

- First step implementation: CORBA name server and a wrapper for access.
    - Keep in mind that implementation may be changed in future, e.g. to a data base system.
- The device locator is the location to check for access control, that is to reject unauthorized device access.
    - In the client for each device to be accessed, flags are set during device instantiation.
    - These flags are checked in the device access.
    - Check is implemented in the client side wrapper around the CORBA device access interface.

3.3.2 Alarm System

- Functionality of the alarm system is the same as in the actual system.
- A standard transport method is used for networking of the alarm messages
    - This can be based directly on UDP
    - or the CORBA Notification service can be used
        - Is it available for VMS?
        - Redundancy? Otherwise it will be a single point of failure.

3.3.3 Surveillance

- It has to be evaluated which surveillance tasks are implemented in the actual system processes.
- It has then to be decided whether they are still needed, in which case they have to be implemented somehow newly.
- No need is seen to implement heartbeats on the VME master processor.
    - An offline status is signaled when an attempt to interact with a device fails, not earlier.
- Applications must be able to identify different kind of interruptions in the communication path, the different reasons actually commonly named “offline” must be identifiable:
    - Device offline: The communication between the VME node and the device breaks
    - Access error: The device object is not accessible

3.4 Interfaces

- Interfaces in this environment mean interfaces between components on different platforms (like the DPR, the interface between the GP and the EC) .
- This includes all interfaces between components which may be split to different platforms some day.

3.4.1 General Aspects

- Always use “simple” types for data exchange, no complex records. This means
    - All elements should be of the same size
    - Arrays of 1, 2, 4, 8 byte data
    - Unless otherwise requested, use the biggest data type needed for all other items too. That is, if the interface needs a 4 byte item and 3 2 byte items, use 4 byte data for all items.
- Align data items according to their natural alignment, that is 2 byte items to even addresses and 4 byte items to addresses which are multiples of 4.
    - Use explicit padding by introducing fill bytes, do not depend on automatic alignment of compilers. Alignment may vary from compiler to compiler.

- Always provide an item to allow detection of sender's little/big endian encoding and float formats (at least if other than IEEE) at receiver side.
- Take care not to influence other interfaces if data structures of one special interface have to be adjusted. Following the above rules should avoid the need of adjusting data structures of interfaces anyhow.

3.4.2 Interface GµP/EC (actually DPR)

- No direct access to ECM-data
- Always use SW interface (procedures, methods) for access
    - Preferably via an access method using command descriptor / data
- No unions shall be used
- No packed elements shall be used

4 Not to Forget

- Hidden Properties: The existing control system implements some features which are highly critical to the reliable operation of the facility as 'hidden properties'. Access path information is not inserted in the access path data base: These properties are not visible to the access interface, access to these properties is not possible with the common tools provided by the control system. Access is possible on the network level only by explicitly sending a network packet, containing the required access information.

  A method has to be implemented for fully controlled access of dedicated properties. It must be ensured that these properties can be accessed by persons who own the highest level of access rights.