A COMMON CONTROL MODEL FOR VACUUM EQUIPMENT AT CERN

I.Laugier, P.M.Strubin, N.N.Trofimov, CERN, Geneva, Switzerland

Abstract

The paper presents results of the work aimed at providing a uniform interface between application programs and vacuum equipment in all CERN accelerators. The basic idea is that, despite the diversity of existing low level equipment controls, at the application program level devices of the same kind, e.g. ion pumps, can be represented by a common control model reflecting their prime operational purpose rather then specific implementations. The model is defined in a language independent form though allowing for easy mapping to specific application programming interfaces. An approach to the model implementation in the CERN accelerator controls infrastructure is described along with results of prototyping in LEP and PS accelerators.

1 OVERVIEW

The Common Control Model (CCM) aims at providing a uniform interface between application programs and vacuum equipment in all CERN accelerators. The model presents a vacuum system to the applications as a collection of logical devices.

Each logical device contains a number of functional components corresponding to physical variables in the underlying vacuum equipment, and belongs to a device class. Each functional component is uniquely identified by a name within its device. All devices belonging to the same device class have the same nomenclature of functional components. Device classes can be organized in a hierarchy; a logical device inherits all functional components defined in superclasses of its class.

2 FUNCTIONAL COMPONENTS

The functional components serve as building blocks in the logical device definition. Several classes of the functional components have been defined in the CCM (Figure 1). Each class provides a standard interface to a certain type of the physical device variables and, depending on the type (analog or discrete, input or output), specifies a number of attributes which applications can observe and, in some cases, modify.

All functional components have the following attributes defined in the *FunctionalComponent* class:

- *value*
  The current or last known value of the physical variable associated with the component.
- *validity*
  Specifies the value validity (valid, doubtful, not valid), freshness (up-to-date, last known), and indicates a specific reason for not valid or doubtful values (e.g., “out of range” or “I/O error”).
- *timestamp*
  For inputs: the time when the value was acquired from the I/O hardware. For outputs: the time when the value was last set by the applications.

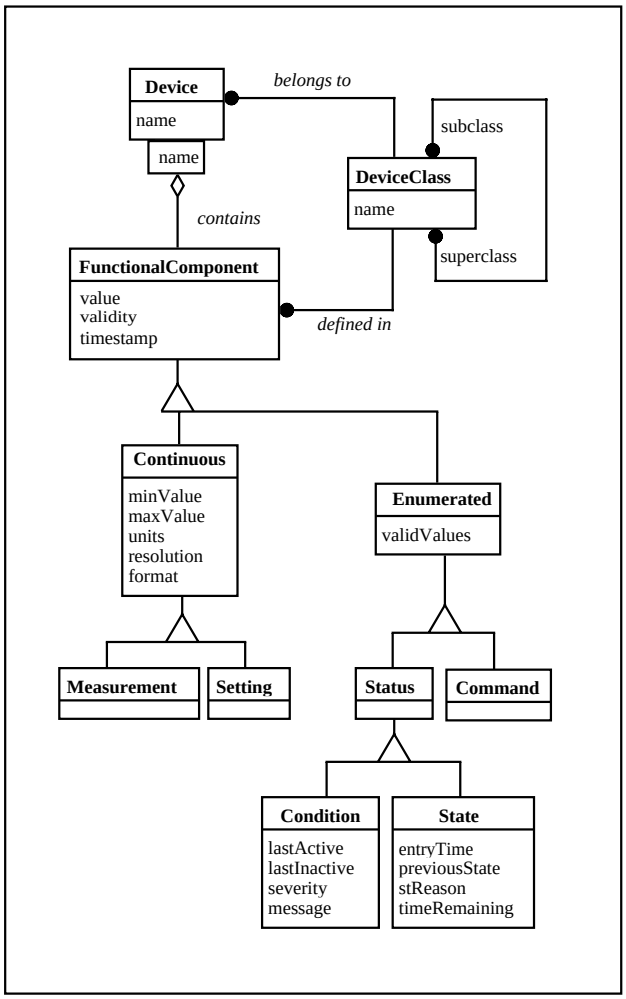

Figure 1: Functional components

Continuous components model analog physical variables. An analog variable has a value range (*minValue* ... *maxValue*), *units* of measurement, *resolution* and *format*. The latter defines the number of digits and position of the decimal point in the decimal value representation.

Measurement is a *Continuous* input component which is typically used to model an analog measurement channel. The normal use for the *Setting* component is to model an analog output channel which sends data to a digital-to-analog converter.

Enumerated components model discrete physical variables which can take one of the values from the *validValues* set. *Status* and *Command* components model general purpose discrete inputs and outputs.

Condition is a specific kind of *Status* that is associated with a certain boolean condition in a device, for example, “pressure above a limit” or “power supply failure”. Its normal use is to signal abnormal situations in a device and, in particular, serve as a front-end for alarm systems. The *value* of a *Condition* can be either INACTIVE or ACTIVE; the *lastActive* and *lastInactive* attributes store time of the most recent transitions between these values. The *severity* attribute can take one of the following values (in the increasing severity order): WARNING, INTERLOCK, FAULT. The *message* text provides detailed information on the abnormal situation signalled by a *Condition*. This attribute takes a new value each time when a *Condition* becomes ACTIVE and (as well as *severity*) may change during this state.

The *State* component provides access to the data associated with a device state machine. The *State* value corresponds to the current device state; the *entryTime* holds a timestamp of the entry into this state from the *previousState*. The *stReason* attribute specifies the reason for the last state transition and can take one of the following values:

- AUTOMATIC - an activity in a previous transient (limited in duration) state has been successfully completed.
- ACTUATION - the state transition was triggered by a user command.
- INTERLOCK - the state transition was triggered by an external interlock signal.
- FAULT - the state transition occurred due to a fault in the device.

The *timeRemaining* attribute is only meaningful in transient states, such as OPENING or CLOSING. It indicates the time expected before an automatic transition from the current state.

3 DEVICE CLASSES

In order to build a control model for a certain type of vacuum devices one has to identify the “control points” common to all devices of this type, map them to functional components and group the components into the device class definition. Major difficulties lay in the first phase: in view of the diversity of low level equipment controls and variety of device implementations in the existing system, it is often not so easy to agree on a common set of parameters which characterizes devices of some type.

The CCM approach here is based on the prime purpose and principles of operation of a vacuum device rather then on its specific implementation and the way in which this device is connected to the control system. With this approach, it is possible to present to the application software a uniform and relatively stable interface with a limited number of logical device classes reflecting core functionalities of the vacuum equipment.

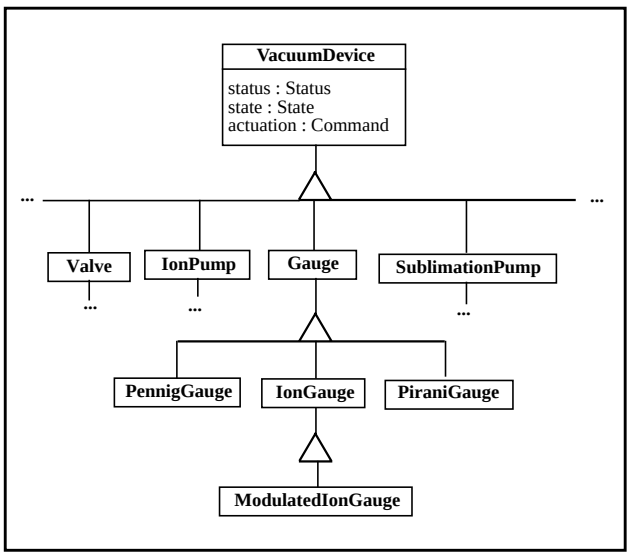

Figure 2: Vacuum device classes

For example, all vacuum gauges in the existing system are represented by the single *Gauge* class with few subclasses reflecting different principles of pressure measurement, and the number of classes is unlikely to grow until a new principle of pressure measurement will be applied in the system (Figure 2).

The functional components represent partial functionalities in a device. The behaviour of the device as a whole is described by its state model. The state model specifies activities and conditions associated with each device state and formally describes the device’s response to the user commands (actuations) and other external and internal events, such as faults or interlocks. The standard interface to the device state machine is provided by the *state* and

actuation attributes defined in the top level *VacuumDevice* class.

The *state* value is usually interpreted in conjunction with the current device *status*. The *status* value can be OK, WARNING, INTERLOCK or FAULT and is set according to the highest severity of the currently ACTIVE conditions in the device (OK if none is ACTIVE). The *status* attribute helps to qualify the state value: for example, a *Gauge* can be in the MEASURING state with status WARNING if, for some reason, the required measurement precision can not be guaranteed (the actual reason can be determined by examining relevant *Conditions*).

4 IMPLEMENTATION

The control model defines the vacuum equipment data and behaviour; however, it does not specify the software procedures and the Application Programming Interface (API) which applications can use to communicate with the logical devices. The methods part in the CCM object specifications is deliberately omitted in order to leave enough freedom in binding the model to different, accelerator controls specific or industry standard, software interfaces. The model compatible equipment access software shall support:

- device interface discovery. It shall be possible for an application to determine at runtime a set of functional components available in a device and, for each component, to determine its class.
- synchronous read and write operations with immediate data retrieval upon request. All attribute data types defined in the model shall be supported. It shall be possible to access individual attributes of a functional component as well as dynamically defined arbitrary groups of attributes, e.g, to get *value*, *validity* and *timestamp* as a result of a single read request.
- subscription to the device data with on-change data reporting. It shall be possible to specify which attributes of a functional component can trigger the reporting and, independently, the attributes which shall be included in the report.

In the prototype implementation on PS and LEP accelerators at CERN, the CDEV package [1] developed in TJNAF has been used to provide the software interface to the model.

Basic concepts of the CDEV software (device, device class, device attribute) are very close to the control model, so the binding is rather simple and straightforward. Functional components map to CDEV device attributes, and attributes of the functional components map to CDEV properties. Using standard CDEV mechanisms an application can get, set or monitor (subscribe to) values of the component attributes.

The implementation follows a 3-tier architecture where the Device Servers, based on the CDEV Generic Server framework, act as intermediaries between applications and subsystem specific controls (Figure 3) [2].

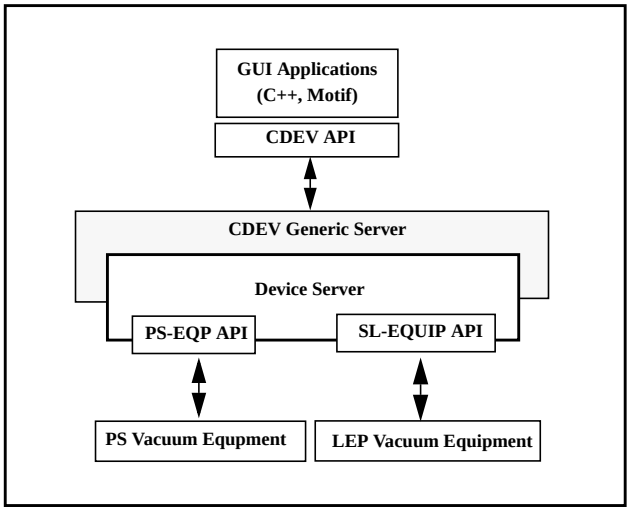

Figure 3: Prototype implementation in the CERN accelerator controls environment.

The primary function of the Device Server is to implement logical devices and to map them to the physical equipment. For each functional component in a device the server provides the “stub” functions which, using subsystem specific interfaces and protocols, link the component attributes to physical I/O points in the underlaying equipment.

A Device Server for the vacuum equipment of the new Antiproton Decelerator has been successfully running for commissioning of the machine [3]. Summarizing results of the prototyping activities, one can say that feasibility and usefulness of the proposed approach have been proved and a framework was established for further developments.

5 REFERENCES

[1] Jie Chen et al., ‘CDEV: An Object-Oriented Class Library for Developing Device Control Applications’, ICALEPCS’95, Chicago, 1996.

[2] I.Laugier, N.N.Trofimov, ‘Using CDEV as Middleware in Vacuum Equipment Controls’, ICALEPCS’99, Trieste, 1999.

[3] P.M.Strubin et al., ‘First Experience with Control and Operational Models for Vacuum Equipment in the AD Decelerator’, PAC’99, New York City, 1999.