UI-ORIENTED APPROACH FOR BUILDING MODULAR CONTROL PROGRAMS IN VEPP-5 CONTROL SYSTEM

D. Bolkhovityanov*, The Budker Institute of Nuclear Physics, Novosibirsk, Russia

Abstract

Specialists combining good programming skills with perfect physics are usually in deficit. So, often physicists prefer to write control programs themselves.

But, besides the “meat” — application control logic — a program must include some routine “spells”, like standard GUI programming, conversation with data server, etc. These lay extra burden on a developer, and, while similar in most applications, these “spells” are hard to move into a separate library, since they are usually too interwined with application logic. However, such separation is highly desirable: common parts can be implemented bullet-proof and feature-rich by professionals, leaving only tiny specific parts for particular application developers.

So, a modular plugin-based architecture was developed for VEPP-5[1] control system, which separates program data description from its implementation. Thus it allows to store this UI-oriented, but GUI-free description in a database. This description is also used by health monitoring, data archiving and web-publishing tools.

COMMONLY USED APPROACH

Large portion of most facilities’ control systems consists of “screen instruments” — meters and control knobs, which are directly mapped onto ADC inputs and DAC outputs.

To simplify development of applications which are constituted entirely of such screen instruments, most control systems provide so-called “display managers”, which take care of interaction with operator, thus eliminating a need to write any code. These are MEDM[2]/dm2k[3] in EPICS, ddd[4] in DOOCS, etc. VEPP-5 is no exception here — its control system CX[5] includes such “screen manager”, called chclient.

“Display managers” generally take some descriptions, which specify lists of display components, including their kind (text fields, menus, graphs, etc.) and positions, plus mapping of those components onto hardware channels, and build user interface “screens” accordingly.

VEPP-5 Specifics

VEPP-5 is under construction now, changes are frequent, so, instead of “visual”, “symbolic circuit” view, a “system-centric” approach is used: one window shows vacuum status of the whole facility, another one is devoted to thermostatization, etc.

MEDM/dm2k, DDD and many others use “canvas” model, which allows a user to put components into arbitrary places of a control window. VEPP-5/CX’s main approach is grid-based (see Fig.1): components (“knobs”) are laid out in a regular grid, with row/column labels added if necessary; “towers” of rows (called “stairs”) are wrapped into several sub-towers if required. (”Canvas” model is also supported, but is little used yet.)

Figure 1: A typical VEPP-5 control application — linac magnetic control. There are 3 screen “elements”, upper two are two-“tower”.

CX applications’ screens consist of 3 layers (see Fig.2):

1. Display knobs themselves.
2. Knobs are grouped into elements — also called containers, which place “streams” of knobs into grids, labeling and wrapping as required.
3. A set of elements is called grouping, which essentially is an application’s screen. A grouping performs simple automatic layout of elements — by rows or by columns, wrapping as needed.

Elements can be nested — any cell in a grid can be occupied by a sub-element instead of a knob. That sub-element can also contain sub-elements, and so on. This allows to build display hierarchies of arbitrary complexity.

Figure 2: 3-layer structure of applications’ screens.

* D.Yu.Bolkhovityanov@inp.nsk.su
So, description of an application’s screen is a tree. And these tree-like descriptions can be stored separately.

THE CHALLENGE

But: if something more than a trivial display of hardware channels is required, notably one of the following:

- Some computations beyond trivial arithmetic.
- Data must be displayed in some unusual way, so that built-in components aren’t sufficient.
- One-to-many relationship of control channels: when change of some display knob must result in modification of several hardware channels.
- Some specific data processing, feedback, etc.

— then “display manager” can’t be used and such application must be coded “by hand”.

VEPP-5 Specifics, continued

CX supports regular, scalar channels and more complex, array channels (fast ADCs, CCD-cameras, etc.; those are called “big channels” in CX) in a different manner, due to a different nature of tasks.

Moreover, in any control system, array channels often have to be displayed in some specific way — just a graph/histogram isn’t sufficient.

But it is hardly possible to equip a “display manager” with more or less unified display components for all flavors of array channels — display requirements are usually too diverse, and often a specific way of data display is the essence of a respective program.

So, all CX applications, dealing with “big channels”, had to be coded individually.

THE SOLUTION

The Idea

A similar problem had existed in many areas; the most widely known example is WWW — how to handle multimedia content — and that problem has got an adequate solution: plugins[6]. A browser doesn’t know what to do with, e.g., Flash animation, but it looks at information, describing .swf file as “application/x-shockwave-flash”, to select appropriate module, called plugin, which does know how to display such a file.

A “display manager” can work in a same manner: when it encounters a component, which isn’t tagged with one of built-in types, a search for plugin is performed, and that plugin is used just like built-in types.

Different “Classes” of Components

The initial goal was to enable placement of diverse user-supplied visualizers of array channels into “display manager’s screens. But making other kinds of display components also “pluginizable” gives many advantages. These “classes” include:

- Scalar channels. So, new kinds of knobs (like “wheel-switch”) can be added easily.
- Array channels.
- Containers. For example, a “worksheet with tabs” (see Fig.3). Or, a “virtual damping ring”, which automatically lays out its sub-components in a ring.
- Decorations — data-less components, which just appear.
- Groupings. So, a generic “vacuum system” grouping would enable switching between a regular bar graph view and a detailed numeric view of all ion pumps’ data (which is frequently requested by technicians).
- User-type components — when the task can’t be reduced to simple on-screen work with one scalar or array channel; or requires some specific processing or feedback; or just something else, not falling into one of the above classes.

“Fallbacks”

There is an important difference from e.g. web browsers: nobody except Flash plugin can handle Flash files, but since there are only few “classes” of components in control system’s displays, a fallback mechanism can be used. Text field would be a reasonable replacement for any scalar-
channel-plugin, a graph can be used for array channels, etc.

Two “parallel” trees

First, the data tree is created, which contains complete information about used channels (references to hardware channels, limits, names, labels, etc.). Then the display tree is “rendered” accordingly with that data.

In cases when only data or just a cumulative status is required (see “Additional Bonuses” below), creation of display tree can be omitted — the data tree is completely functional in its own.

Some Details

CX “display manager” — chlclient — is just a mere frontend to a set of GUI libraries: libChl (stands for “CX high-level”), providing a general application functionality; libKnobs, providing knobs API and a default set of display knobs; a tree-management library, libCdr, which can also be used in non-GUI applications (see “Additional Bonuses” below).

Internally, there’s very little difference between standard components and plugins. Both make use of the same knobs API, and when libKnobs searches for appropriate display component, it just looks through several tables, instead of one.

In fact, the “canvas” display model is implemented as plugins in CX: the “canvas” itself is a container-plugin, while various decorations (rectangles, ellipses, lines, etc.) are decoration-plugins.

ADDITIONAL BONUSES

Since display channels’ specifications are separated from code and are stored either in files or in a database, these descriptions can be directly used by various generic control system tools. These include:

- “Control center”, which works as control system’s start menu and as health monitor (see Fig.4).
- Web-publisher.
- Data archiver.

So, any control application (which is usually a reflection of some subsystem of a facility) automatically becomes fully integrated into control system’s environment.

VEPP-5 USE

As more and more VEPP-5 subsystems are put into regular operation, demand for various specialized, not-just-a-grid-of-text-fields applications, grows.

In the last two years the majority of such applications in VEPP-5 control system are implemented in a plugin-based fashion. This quickened the development and made the control system more flexible.

REGARDING NOVELTY

While tree-like descriptions of control screens aren’t new — such approach has wide use, from Visual Basic to LabVIEW — there are two significant differences in the approach presented above.

First, a data hierarchy exists separately from a display hierarchy. These trees are either “parallel”, or display tree doesn’t exist at all.

Second, plugin-architecture with “fallbacks”, where plugin displays can be provided for scalar channels, array channels, decorations, containers and “generic” user/application-specific components, enables to turn any control application to a unified form.

CONCLUSION

UI-oriented approach for building modular control programs proved to be very useful. It significantly eases development of control applications, and makes the whole control system more structured and modular. This approach will get further wider use in VEPP-5 control system.

REFERENCES