Process Visualization with Oberon System 3 and Gadgets

E. Templ, A. Stritzinger, G. Pomberger

Institut f. Wirtschaftsinformatik
Ch. Doppler Laboratory for Software Engineering
Johannes Kepler University Linz, Austria

e-mail: (templ | stritzinger | pomberger)@swe.uni-linz.ac.at

Abstract

In process automation, visualization systems are used to visualize the state of a plant (in real time) and to enter control commands for interaction. The proliferation of graphical user interfaces has improved design opportunities as well as user interface quality considerably. Nevertheless, the state of the art in visualization systems is rarely satisfying for end users (operators) or for application developers (automation engineers). Concrete visualization systems often vary considerably with respect to their user interface components and their process control system interface. Hence, adaptability and extensibility are utmost crucial in visualization systems. This was the reason for choosing Oberon and the graphical user interface kit Gadgets as the system platform for the realisation of a new visualization system, called VISION. In the following, we present short introductions to Oberon System 3 and Gadgets as well as to the field of process visualization systems. Furthermore, some interesting implementation aspects are described. A rough rating of Oberon's suitability for practical applications based on our experiences is given in the conclusion.

1. Introduction

Oberon System 3

Oberon System 3 (V3) is an evolutionary continuation of the original Oberon operating system [WirGut]. One of the newly introduced, basic concepts of V3 is a type Object in the system kernel which provides a mechanism for making objects persistent. In order to handle quite a large number (>1000) of persistent objects, V3 manages libraries of named objects and provides a small set of generic operations (load, store and copy, as well as get, set and enumerate attributes). V3 exhibits a quite object-oriented architecture where application and user interface components are strictly separated. In general the application itself works exclusively on abstract, i.e. non-visible objects which form the data model. For viewing and controlling purposes, concrete graphical objects called gadgets are employed. In contrast to the original Oberon system where the display space is strictly hierachically structured by means of nested frames, V3 allows frames which can be shared when an objects has multiple representations. In addition, V3 frames are objects, too. (Frame is an extended type of Object.) Thus they can be stored easily in libraries. Whenever messages are sent to display objects, the message is passed through the entire frame hierarchy to allow any superordinate object to perform some specific operations. This message broadcast mechanism into the display space forms the basis for a generalized model view scheme adopted from Smalltalk, where models should not know their views [KraPop].

Gadgets

Gadgets is a graphical user interface kit containing prefabricated, reusable interface components of low granularity for constructing graphical user interfaces of arbitrary complexity and sophistication [Gutkne]. The look and feel of the user interface is defined by several gadgets like buttons, sliders and checkboxes as well as a set of guidelines for responding to mouse commands in general. Gadget user interfaces can be adapted to individual needs on the fly by the end user. An important Gadget type is a panel, which are containers for other gadgets and manage editing operations on their children. Gadgets can have several attributes consisting of a name and a (typed) value. Typical attributes are colors, labels and, in particular command strings, which can be executed by the gadget. In order to access visible and invisible attributes interactively, an instrument named inspector is provided.

Process Visualization Systems

As in many domains, personal computers have had a great impact on the field of process automation. Standard industrial PCs increasingly often are replacing conventional techniques like control panels, dashboards, flow and mimic diagrams for controlling tasks. In contrast to widely used, specialized process computers in embedded real-time systems, visualization systems usually do not control an associated plant in an independent automatic manner, but serve for monitoring and interacting with the facility. Typical applications of visualization systems are chemical plants, assembly lines, bulk goods mixing facilities, and power plants. Visualization systems are most often unique implementations or employed in a rather small count. Hence, the individual construction costs of a single system are a deciding cost factor, so that excessive programming would never pay off.

Developing a visualization system application usually involves interactively drawing, assembling, configuring and parameterizing a generic visualization system by an automation engineer. Advanced programming skills should not be required. A high-quality visualization system should be characterized by a purposeful functionality and a simple, user-friendly interface for end users as well as application developers.

Our experiences with commercial visualization systems have shown, that even heavy-weight systems often lack features which would be quite natural and desirable in a specific application. On the other hand, a lot of rather exotic functionality often burdens the automation engineer. Hence, the desire for a visualization system with extensibility as an intrinsic attribute became apparent. In the VISION project we tried to create a lean kernel system which can easily adapted and extended to specific requirements.

2. The Process Visualization System VISION

A visualization system usually does not control the associated process directly. Instead, a process control system (PCS) is used for interacting with the process, controlling automatic execution, and communicating with the visualization system. We use a standard serial interface (RS 232) as communication medium with the PCS. This is sufficient, because visualization systems can accept a certain time delay (1-3 seconds) beween real-time and displayed time. This less-critical timing requirement is also prerequisite for the sufficiency of a few nonpreemtive tasks provided by the Oberon system. A typical configuration of a VISION system coupled with a Siemens Simatic S5 PCS using the protocol standard RK512 is shown in Fig. 1.

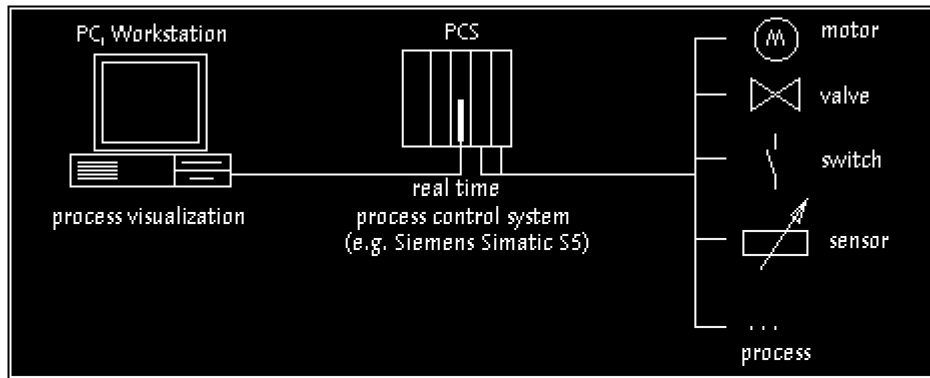

Fig. 1: Typical configuration of a VISION system

Monitoring and Operating (Operator Aspects)

Before the arrival of computerized visualization systems, mimic diagram panels with integrated or separate control elements were used. One may regard such panels as early predecessors of graphical user interfaces (built in hardware). In order to smooth the transition to the new technology for the operators, it is common practice to provide the familar static and active elements as used in hardware panels. Sometimes even both classic and computerized visualization systems are used together.

For displaying the state of plant devices, (colored) lamps, digital and analog instrument displays, and trend plotters are used in most cases. Important or dangerous state changes have to be reported to the operater in a very striking manner, which is usually a blinking lamp. These events, called alarms have to be acknowledged by the operator; then the warning signal changes into a steady light and disappears when the alert condition is resolved. Sometimes text displays are used to report the reason for the alarm. Alert stages are often protocolled textually in a journal file.

VISION also provides lamps, digital and analog displays, and trend plotters. Alerts can be collected in a list containing time, date and a descriptive text (error message). The alert list can be displayed on demand in a separate panel or continuously shown in an integrated panel. Printing and storing of the list is also supported. Another way for visualizing states is by means of displayed (short) texts with arbitrary fonts and colors. For example, a motor state can be displayed by either a green ON or a red OFF text. Colorchanges may occur on foreground texts as well as on background areas. Workable controller gadgets for real-time controlling tasks are not supported by VISION; only their measurement values can be displayed (in digital or analog form).

The typical actuating elements in hardware control panels are pushbuttons. Furthermore, toggle buttons, illuminated buttons, rotary switches, turn knobs, as well as thumb wheel and dip switches for set point value input are realized as hardware gadgets.

In VISION buttons with a 3D look, with or without toggle behavior, can be actuated by a mouse click. Several of them can be combined to a radio button block with mutual exclusion. To input numeric values, digital displays with increment and decrement buttons or keypads can be assembled and, of course, the keyboard can be used. At the moment VISION does not support rotary switches; because of their less intuitive mouse operation, we suggest using radio buttons instead. But their is no principal problem preventing their realization.

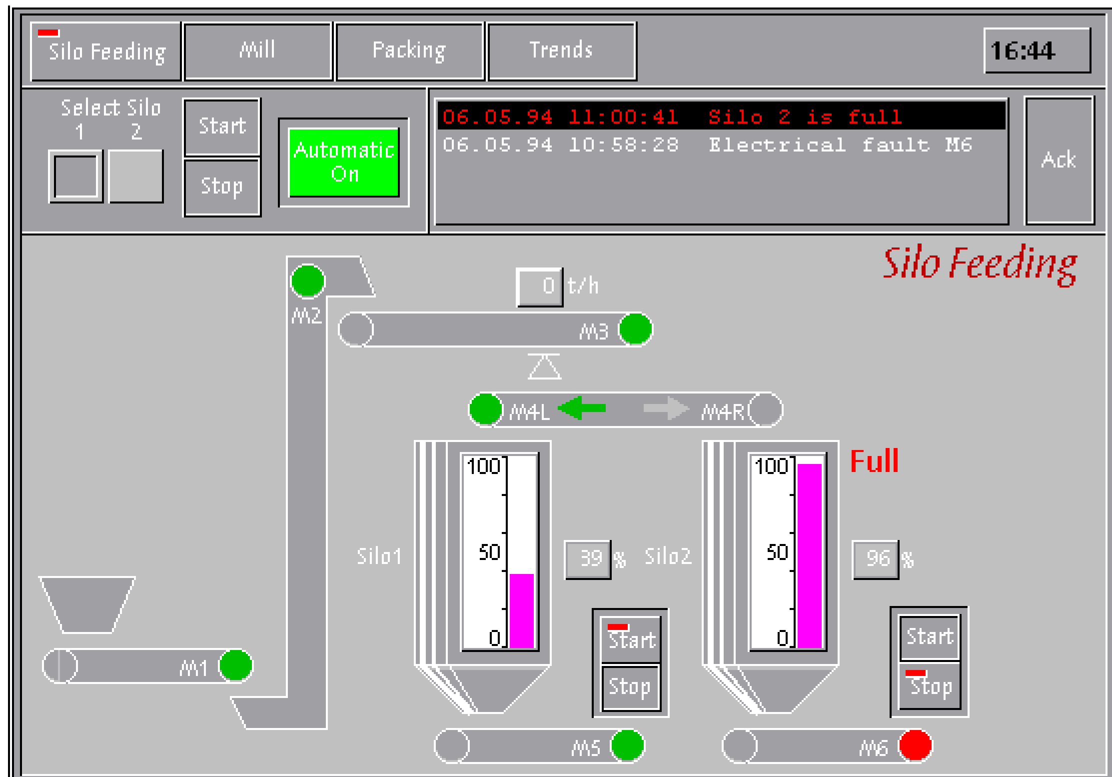

Fig. 2: A mimic diagram of a cement plant as VISION panel

Editing and Constructing (Developer Aspects)

The core Gadgets system itself already provides all means for interactivly producing the user interface for a concrete visualization system. The tools for this task are the Illustrator and the Inspector. The Illustrator enables the developer to produce basic graphic elements like lines, circles, squares and polygons with individual frames and fill colors. Furthermore, it is possible to integrate scanned bitmap images into a panel. Active and VISION-specific gadgets are also arranged by means of the Illustrator. In many applications it is not possible to visualize the entire process facility in a single panel; instead, several switchable or popup panels are combined. Copying gadgets or groups of them produces a precise copy of all views which share the same model and thus mirror the very same process state. All VISION elements as well as standard gadgets possess a number of specific attributes, at least a name attribute which can be inspected and modified by the inspector tool.

Interface to the process control system (PCS)

The user interface components (gadgets) of a VISION application have to be logically connected to memory blocks of the associated PCS. Whereas the PCS organizes its data in words (or items) and blocks, VISION reflects the information in appropriate attributes of certain gadgets. All informations which are neccessary for connecting the gadgets front-end with the PCS are described textually in an application file. This ASCII file is structured into several sections:

- constant declaration
- mapping declaration
- PCS declaration
- GadgetLink declaration
- alarm declaration

- cyclic communication description
- initialization

The constant declarations serve primarily as a vehicle for improving readability. They are represented as integer values and are heavily used for naming colors, eg.:

```
white = 0; red = 1; ...
```

The mapping declarations describe enumerations of at most 8 different values, e.g.:

```
StringMap = ("OFF","ON","FAULT")
```

PCS declaration: The VISION system uses channels for communication with (possibly more than one) PCS. For each channel the type of interface and protocol of the device have to be specified, e.g.:

```
PROTOCOL RK512; PORT=COM1; BAUD=9600;
```

Furthermore, each received or transmitted data block has to be described by a name and a number of items. An item is specified by its address (word number), a PCS data type, and a name, e.g.:

```
BLOCK Inputblock = DB10;
    DW 1: INTEGER SCALE(2.0,10) = Temperature;
    DW 2: BCD = Countervalue;
    DW 3: INTEGER = Motor1;
END Inputblock;
```

The gadget-link declaration defines where and how state changes are visualized. An item of a data block is associated with one or more gadgets. Each value can be transformed by a map before assigning it to an attribute, e.g.:

```
PumpMotor = Motor1.Value (StringMap), Motor1.Color (ColorMap);
```

PumpMotor is the name of a data block item; Motor1 is a unique gadget name. Value and Color are attributes of the Motor1 gadget. Every time a data block containing the pump motor item is sent from the PCS, the Value and Color attributes of the Motor1 gadget are updated if neccessary.

Alarm declaration: Each data item can be checked, whith respect to an alarm situation. Each alarm knows its alarm group, which is an object that typically serves as model of an alarm panel, e.g.:

```
ALARMGROUP SiloAlarm;
    Silo1Content > 99: "Silo 1 overflow";
    Silo1Empty = 1: "Silo 1 is empty";
    ElevatorM2 BIT 1: "Conveyor M2 electrical fault";
    ElevatorM2 BIT 2: "Conveyor M2 mechanical fault";
END SiloAlarm;
```

The cyclic communication description specifies which data blocks are transmitted in which direction at which intervals. The shortest time interval is one second because of the limited throughput of the serial port. Usually only cyclic reading of PCS data blocks is needed; cyclic writing allows the PCS checking whether the visualization system is still responding, e.g.:

```
EVERY 1 SEC READ DigitalInput;
EVERY 5 SEC WRITE LiveSignal;
```

Initialization: In some applications it is meaningful to read a number of data blocks at visualization system start up. This can be specified in the initialization section, e.g.:

```
BEGIN
    READ DigitalInput, AnalogInput
END
```

3. Implementation

The functionality of VISION can be separated into two parts: Part 1 are user interface components. The components for establishing and supplying the data models form part 2. For the construction of the user interface several new VISION-specific gadgets were implemented (part 1):

- BarGraph for displaying an analogous value
- DipSwitch for incrementing or decrementing a number
- OutputField for displaying an alphanumeric text
- Lamp to signal a boolean state
- Trend for showing a course of an analogous value over a period of time
- Logger for storing a trend
- AlarmPanel for displaying a list of triggered alarms
- AlarmGroup for managing triggered alarms

Because of Oberon's and Gadget's extensibility, it is easy to build new gadgets and to extend or modify existing gadgets. All attributes, even those unknown at design time, can be employed for visualizing a process state by just establishing a suitable link in the application file.

Model-related components (part 2) are responsible for the interpretation of the interface description and for communication handling. At the top of VISION's module hierarchy resides the command module Vision. It provides all commands for controlling the system:

- parsing an application file
- installing and removing communication tasks
- opening a startup panel
- starting and stopping communication
- sending and fetching data blocks and items

Like other visualization systems, VISION can be adapted to different PCS devices. All these devices have their own notion for addressing data blocks and items. Furthermore, depending on the kind of communication port (serial port, process field bus, etc.) a number parameters may vary. In order to achieve the required flexibility, the module VChannels defines an abstract base class for all possible PCS devices which may be used in cooperation with VISION. This class must be specialized for each concrete PCS.

Every time a data block from the PCS is fetched by VISION, the data are converted into an internal representation and compared with the stored and displayed state. When the state has been changed, the corresponding gadget attributes (as specified in the link section of the application file) are updated with regard to optional mappings. Since we do not know whether or where or how often our object is visualized (models do not know their views), we broadcast an update message into the display space. So all affected gadgets get a chance to redraw themselves. Broadcast messages are sent to all visible objects, regardless of whether they are concerned. Although it causes some overhead, this policy is justified because it offers a lot of flexibility, and state changes are typically rather rare events (a few per second). If this were not the case, the operator would totally lose the overview and control over the facility.

4. Experiences with Oberon and Gadgets

The implementation of an extensible visualization system was also motivated by the desire to gain insights into Oberon's and Gadget's suitability for practical applications in the field. The following comments are based on a single project and are of course subjective.

Language

Oberon is a simple, easy-to-learn language, especially for programmers already familiar with Pascal or Modula-2. The static type safety, good readability, automatic garbage collection, and short turnaround cycles are great benefits for programmers. The concept of type extension supporting polymorphic variables is the most important feature, and one can hardly imagine programming without it. The concept of implementing messages as (record) objects and interpreting them via a handler procedure as employed in the original Oberon system as well as in System 3 has proven to be an extremly flexible communication mechanism. By this mean it is for instance possible to send messages through a chain of objects which have no idea of the type or semantics of the message; they just pass it through until another object is willing to respond to the message. The drawbacks of this untyped message passing (in the sense that any message can be sent to any object) are a performance penalty and, what is more painful, that message objects are quite clumsy because records have to be declared, filled and interpreted by the programmer. Methods or type-bound procedures, as they are called in Oberon-2 [MösWir] could circumvent these troubles, but in certain cases they are just not flexible enough.

Operating System

Reusability and extensibility are deciding factors in order to be able to adapt a lean kernel system to new requirements of future applications which never can be foreseen by a tool designer. Oberon System 3 supports these requirements to a very high degree at low resource costs. The concept of persistent objects and libraries as introduced in System 3 was invaluable and worked without any flaws. The lack of a sophisticated (preemtive) multitasking system caused some questions and doubts about the feasibility of such a visualization system. The cooperative tasking concept of the Oberon kernel was sufficient for our purpose, as it is for many single-user applications. But more advanced embedded systems will require a preemtive multitasking kernel. Nevertheless, we do not advocate for such an extension, because we doubt that in general Oberon can or should compete with powerful commercial operating systems from the market place. Instead, we would welcome a shift in perspective in viewing Oberon as a universal, generic application running in commercial environments rather than a standalone system.

User Interface

Oberons textual user interface of viewers and command tools frees users from superfluous modes and states. Eliminating almost every kind of confirmation imposes somewhat more responsibility to the user, which sometimes may cause misguided actions by infrequent or novice users. In particular, users familiar with graphical user interfaces, need considerable time to readjust to the strange, nevertheless efficient user interface of Oberon. The graphical user interface management system Gadgets is a very fascinating and nice-looking attempt to polish Oberon's surface, and its flexibility is striking. The need for graphical user interfaces may sometimes be a matter of taste; for applications like VISION it is a must. Thanks to Gadgets, we could implement our system in a very short time.

5. Conclusion

We have the impression that from a technical point of view VISION could compete quite well with commercial visualization systems. Although it was not our goal to develop a product-like system, Oberon and Gadgets provided such an outstandingly flexible and yet rich basis that our effort was considerably less than estimated.

Acknowledgments
We owe special gratitude to J. Templ and J. Marais for their valuable suggestions and technical support.

6. References

Gutkne J. Gutknecht, Oberon System 3: Vision of a Future Software Technology, Software Concepts and Tools, Springer International, Vol. 15, No. 1, 1994

KraPop G. E. Krasner and S. T. Pope, A Cookbook for Using the Model View Controller User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, Aug./Sep. 1988

MösWir H. Mössenböck and N. Wirth, The Programming Language Oberon-2, Structured Programming, Springer-Verlag New York Inc., Vol. 12, 1991

WirGut N. Wirth and J. Gutknecht, Project Oberon, Addison-Wesley 1992.