The use of interpretation for data acquisition and control; its impact on software development and project management

Otto Vinter

Otto Vinter, Software Engineering Mentor, Sthensvej 2F,
2630 Taastrup, Denmark
vinter@ottovinter.dk

Abstract. Over a decade I and a number of other software engineers introduced, developed, improved and expanded the principle of interpretation for data acquisition and control task descriptions; initially a simple description and execution tool to assist plant engineers; in the end a software development framework for modeling, managing and executing large, complex projects in this domain.

Keywords: Data Acquisition; Process Control; Software Engineering; Project Management; Modeling; Interpretation.

1 Introduction

On the first data acquisition and control system in 1969 for the Danish power plant Vestkraft Blok2 (Fig. 1, Appendix 1, [1][3]), we simply wanted to create a tool (simple process language) to make it easier and more flexible for plant engineers to define their measurements and calculations, and thus do away with the limited and predetermined ("hard coded") operations on process data based on flags in data tables.

At the completion of the process control system in 1978 for the Copenhagen Mail Sorting Center, the principle of using interpretation on a data-model of the system (Fig. 3, Appendix 2, [4][5]) had evolved into a software engineering framework that influenced not only the system architecture, but all phases of software development from detailed requirements, design, coding, testing, and release staging, to project management, estimation, planning, scheduling, configuration management, quality procedures, and documentation.

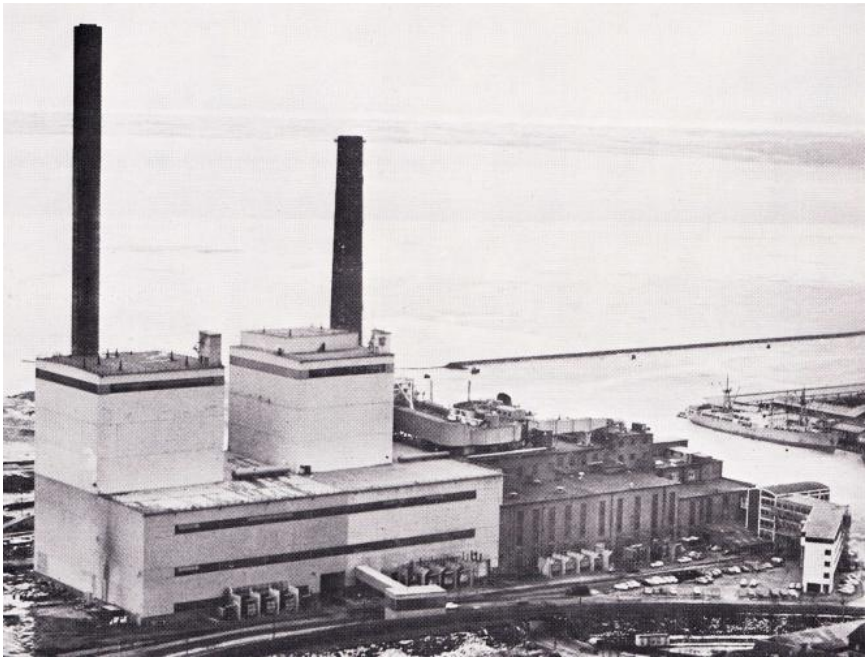

Fig. 1. The power plants at Vestkraft. Blok2 is the left tower. See Appendix 1 for details.

2 The early data acquisition systems

In the beginning of the 60's the use of computers started to spread from pure mathematical applications to the process control industry. Both buyers and suppliers, however, were very cautious about letting the computer take full control of the industrial processes. Acquisition of analog signals and their conversion to binary numbers was not very well known, and disturbances from the electrically noisy environment of high-power machinery could severely influence the low-level signals at maximum values of 24 mA and 10 V. Consequently the first computer systems were only used for logging measurement data; performing simple conversions and calculations; and present the operators of the plant with alarms and reports.

These data loggers were programmed like the hard-wired electronic instruments they were intended to supplement. The early programs were sequential monolithic structures that scanned the data acquisition channels and stored them in memory resident tables after conversion and simple alarm checks. Other programs would later read these tables, perform calculations and generate reports. Around the mid 60's the first multiprogramming monitors appeared which allowed programs to execute in parallel, e.g. data acquisition programs could execute in parallel with report printing programs. The structure of the programs did not change very much, however. They still retained their basic monolithic structure. There were just more of them; now executing in parallel.

3 The original idea of a dedicated process control language

In these early systems, in order to describe the processing to take place on the data, a number of flags (bits) were kept for each measurement variables along with its status and value. They defined what conversion routine to use, whether alarm limits should be checked, or what other calculations should be performed. When a data processing program scanned the data tables it examined the flags individually (in a specific sequence) and called the relevant routine (basically a huge case structure).

Therefore the plant engineer who designed the actual processing had the difficult task of defining the data tables and processing flags; and he had to do it in the computer's native machine code. Because of the limited amount of predefined flags and the fixed sequence in which they were scanned, it was often difficult to describe the processing that was desired.

We wanted to improve this situation by developing a data acquisition and control language closer to the concepts of a plant engineer, to gain flexibility by replacing the flags and fixed sequence of processing and allow the engineer to select the processing from a range of language commands. The introduction of such a process language was not new. Other dedicated data acquisition and control languages were developed for similar systems at that time. However, the trend was to compile such languages into (monolithic) executable programs.

4 The introduction of the interpretation principle

We could not allow ourselves the luxury of compiler for the language because our development system was the same as the executing system, and therefore had severe limitations on memory, backing store and peripherals. Furthermore, at that time compiled code was known to lack the necessary performance for real-time applications. We therefore decided to define the language in a macro-like format, which could be easily translated into command data-structures.

We made the command data-structures self-contained, e.g. the reference to the software routine to execute the macro-command and the parameters were stored together. Since each routine was designed for the specific purpose of handling its parameters, the length of each command data-structure could also be calculated and stored in the structure.

Because we did not have a file handling system either, we had to organize the lay-out of command data-structures and data variables on the backing store ourselves. At specific places the translator would insert special commands to load the next segment of command data-structures from backing store to memory, and commands to swap segments of variables that had been updated with others which would be needed next.

At predefined intervals, a simple program (interpreter) executing in one of the multiprogramming processes scanned the model containing the command data-structures. It would subroutine-jump to the routine referenced in the first command data-structure. When that routine returned, the interpreter added the stored length of the command data-structure (parameters) to point to the beginning of the next command data-structure, call that routine and so on, until an end-of-data-structure

command was encountered. This way data processing was no longer contained in a monolithic program; it had turned into an extremely flexible set of small dedicated routines in a data-model that was interpreted rather than executed.
Several routines (macro-language commands) would normally have to be called to accomplish one complete processing of a plant variable (Fig. 2), but the type of checks, conversions, calculations, and the order in which they were performed, was no longer limited or predefined by the real-time processing program.

```
; Create new value for TFd and add to sum in TFdS10
/802                    ; TFd, steam temperature for HT
 :IWR,   K=802          ; Initialize working registers (variable 802)
 :LSV,   V802           ; Load state and value for TFd (variable 802)
 :ECAV,  R1T25          ; Evaluation control of analog value (range, terminal)
   L1                   ; skip conversion and checks if compensated by operator
   L2                   ; Skip conversion in case of a measurement failure
 :CRE,   K=150          ; Convert resistance element (parameter value)
 :ILCMM, K=-200,Pih=6000 ; Instrument limit control (min, max)
2:TPC,   V802           ; Test for failures and update status (TFd)
 :TCCV,  V219           ; If compensation use value for TOH (variable 219)
1:PCM,   K=-50, Pah=5650 ; Plant status control (hysteresis, maximum)
 :SSV,   V802           ; Store new state and value (TFd)
 :SUM,   V3301          ; add to TFdS10 (variable 3301)
```

Fig. 2. Processing commands for a temperature variable at Vestkraft Blok2.

5 New opportunities because of the interpretation principle

Having one central data-model, which is interpreted rather than executed, opened up for a number of advantages in the development and customization of data acquisition and control systems. New language commands could be easily defined; a small dedicated component (routine and parameter description) designed, coded and added to the macro-translator. Nothing needed to be changed in the on-line system's processes (programs); the data-model was simply replaced.

Defects were easier to locate because they were confined to the new component (or the macro-translator), as there was no direct communication (e.g. calls) between routines, only through the data values and their status.

6 Testing in a simulated environment

The principle of interpretation allowed us to test new components in a simulated environment (e.g. off-line) using only those parts of the data-model that were needed for testing the component. Dedicated test drivers and stubs (simple test commands included in the macro-language) were inserted in the test data-model to check whether the new routine produced the correct (expected) results under different conditions of input data. For each call the drivers and stubs stepped through a list of test inputs (test cases).

A logging facility was inserted (another test component in the data-model) that could print the data values and status used by the component (routine) along with the

result data and new status it generated (stored). From this, it was only a small step to include expected results in the test lists and let the logging facility mark any incorrect results in the print. Automated regression testing in a simulated environment had now been introduced as a natural thing.

Even late in the 70's software programmers were scarce and usually we had to teach them everything: assembler language, linkers, loaders, bootstrapping, running the system, and of course basic software engineering good practices (it wasn't called that at the time). Using the principle of interpretation and simulated test environments made the introduction of rather primitively trained developers on a project much easier and safe. They were able to find and correct their errors early during unit tests in the coding phase, and quickly became seasoned developers on-the-job.

Testing in a simulated environment also meant that we were able to implement a defined process for promoting partially completed systems through several levels of environments (unit testing, system testing, and production) complete with automatic regression test data and test procedures.

7 Effects on the software architecture

The principle of interpretation of a data-model influenced all aspects of our software development. The most immediate effect was of course on the software architecture; based as it were on a comprehensive model of the industrial plant, and an easily adaptable and flexible set of software components.

All data values and their status were fetched, updated, and stored in the model. All connections and communication between the modeled physical components of the plant took place through their representations in the model. All other types of handling and control were also designed into the model and represented as "abstract" components, e.g. conversions, averages, accumulations, calculations, progress timing, storage management, plant sub systems (groups), as well as "physical" output devices and set-point controls.

Alarms, reports, logs, and other output data about the operation of the plant were generated from data in the model and communicated via a number of message buffer queues to dedicated reporting processes running in parallel to the acquisition and control process, so that processing and output tasks could perform independently of each other [2].

Input to and output from the message buffer queues were protected by semaphores, and buffer overruns were handled so they did not influence the operation of the acquisition and control process. The principle of interpretation was also used to describe the layout, contents and generation of reports.

8 Effects on project management

Project planning, scheduling and management were impacted by the data-model architecture. Because of the limited complexity of each component, it was easy to estimate how long it would take to implement it, and actual data from previously

developed components quickly created a solid basis for new estimates. Each component could be developed and tested almost independently of other components, so it was relatively easy to assign components to the available developers in the project plan and perform follow-up on development progress.

However, this did not eliminate the need for overall design of the system of components. That always involved senior developers. It sometimes turned out to be a bottle-neck and generate overruns on its estimates.

In the end we managed to deliver our projects almost on time and budget, and with very few defects in operation.

9 The applications of the interpretation principle

The interpretation principle and data acquisition and control language commands from Vestkraft Blok2 were reused and improved for another power plant (Nordkraft Sektion4) and adapted for a sugar production plant (Saxkjøbing Sukkerfabrik).

However, the comprehensive software engineering framework described above was not realized until the Copenhagen Mail Sorting Center (Fig. 3). In this system all physical components of the plant were modeled as components in the data-model.

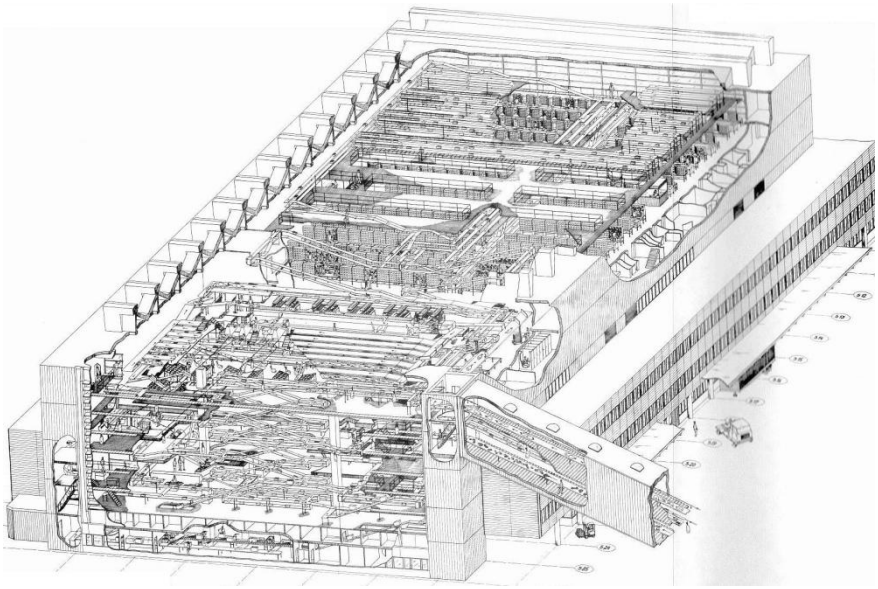

Fig. 3. The Copenhagen Mail Sorting Center. See Appendix 2 for details of the modeling.

10 Why the principle didn't catch on

The advent of new computer and software technology in the late 70's and early 80's meant a complete change in data acquisition and control systems from comprehensive centralized systems to a network of small dedicated minicomputers, microprocessors (PLCs), which required less complex software systems.

Secondly, the response time of a system interpreting a data-model is never faster than the time it takes to scan the data-model. This works for most industrial processes which only change slowly. However, direct control loops (PID) and other fast reactions to input must be handled by separate processes executing in parallel. As prices on computers went down, and hardwired instrumentation went up, the trend was to use computers to engage faster and more directly with the control of the industrial plant.

11 A final twist in the tale

In the late 80's I was product manager for a new line of automatic test equipment at Brüel & Kjær. Our goal was to develop a set of virtual (e.g. software-based) measuring instruments. On top of those we wanted to develop a comprehensive test and measurement environment, where engineers could develop their own test and measurement projects, combining the instruments of their choice with calculations, sequencing, loops and controls. Numerical results and graphs were to be combined into reports that showed whether the product under test has passed or failed.

We had many heated discussions on how to design this test and measurement environment. There was a clear divide between the experienced test and measurement engineers and the brilliant software engineers, some just out of the university. For my part, I was impressed with the advances in computer speed, compiler capabilities; and object-oriented development seemed to become an important principle for the future. Therefore we decided to base the test and measurement environment on compilation of our measurement components rather than interpretation.

We struggled several years to make this design work, but did not succeed. In the end the project was cancelled. A couple of years later a US company (National Instruments) launched a since then rather successful test and measurement environment based on interpretation of simple measurement, calculation and control components, which could be combined graphically (2D) in an easy drag, drop and connect fashion. These simple test and measurement components resemble the language commands we had used in the early days for the industrial plants, albeit in a more modern, colorful and graphic way.

The lack of speed in interpretation, that we had feared so much, was not a problem for test and measurement engineers, partly because of the increased speed of computers and partly because many test and measurement processes change at a slow rate.

In hindsight this example shows that, given the right conditions, the interpretation principle can still be the right way to solve a complex problem. And by the way,

Microsoft Excel is actually another case of a successful use of the interpretation principle.

Acknowledgments. I wish to thank Peter Kraft, who was project manager on the Vestkraft project, where the initial idea of using the interpretation principle for data acquisition and control systems was born. Furthermore, I wish to thank Bent Bagger and Ebbe Sommerlund, who were my primary supports on the Copenhagen Mail Sorting Center project, where the full impact of the principle was realized. My thanks also go to many people for their assistance in recovering our common past from our combined rusty memory and dusty archives.

References

1. Kraft, P., Vinter, O.: Rapport over proceskontrolsystemets opbygning hos I/S Vestkraft Esbjerg. Regnecentralen (1970)
2. Kraft, P., Mossin, E.: Datastrømme, elementer til kommunikation i et proceskontrolsystem. In: NordDATA72 Proceedings (1972)
3. Nedergaard, N.: Procesregnemaskinen på Vestkraft. Elektroteknikeren, 66. årgang nr. 4 (1970) (see also nr. 23 for a description of the whole plant)
4. Prag, P.: Datamatstyring af transport- og sorteringsanlæg i Centralpostbygningen i København, Rådgivning og projektering. In: NordDATA77 Proceedings (1977)
5. Vinter, O.: Datamatstyring af transport- og sorteringsanlæg i Centralpostbygningen i København, Transportdatamatstyringen. In: NordDATA77 Proceedings (1977)

Appendix 1: The Vestkraft Blok2 Power Plant

The power plant was built in 1969 (Fig. 1, Fig. 4). It had an electric capacity of 250 MW, plus a heating capacity of 160 Gcal/h that covered the needs of Esbjerg city. The turbo-group was delivered by BBC and the boiler unit by Babcock & Wilcox. All of the plant controls were handled by conventional electronic equipment. For the complete supervision of the plant a digital computer system from A/S Regnecentralen was installed [1][3].

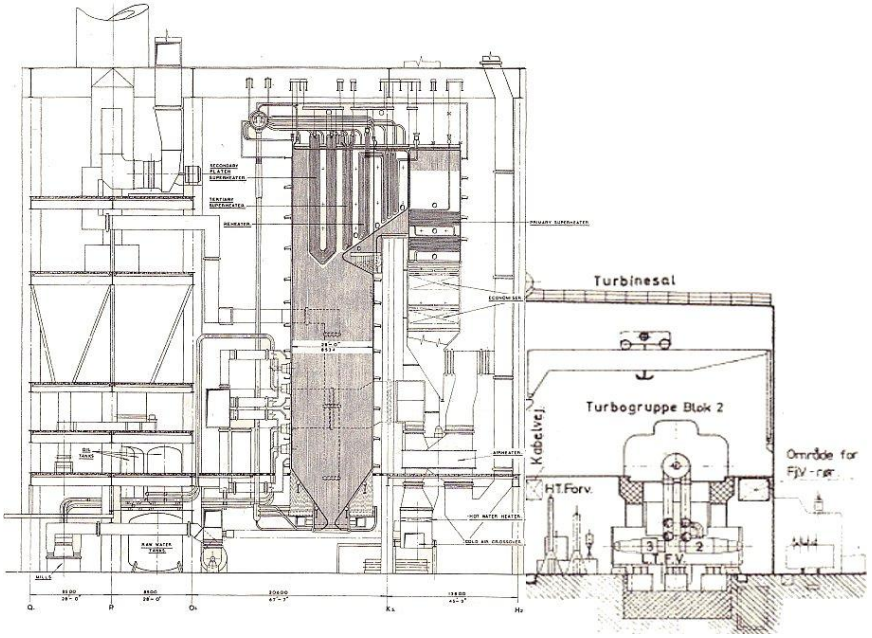

Fig. 4. A view into Vestkraft Blok2. A combination of two original drawings, matched to fit the correct proportions of the plant. The boiler section with its heating supply units to the left, and the turbine section to the right.

Every 10 seconds, all bearing and coil temperatures from major motors, pumps and generators were measured and analyzed by the computer. Special supervision of boiler drum, oil burners and air pre-heaters was also performed. Approximately 250 analogue measurements.

Every minute, another 250 analogue values were measured and analyzed; among others 170 super-heater pipe temperatures. The latter were particularly important because close supervision of these could increase maintenance intervals and prevent break-downs. All relevant measurements were accumulated over time. Performance and load calculations were performed and used to improve the management and performance of the plant.

The RC4000 computer configuration was: 32kB memory, 512kB drum storage, 512 analogue inputs, 216 digital sense inputs, 48 digital interrupt inputs (for counting), and 48 digital outputs.

Appendix 2: Modeling of the Copenhagen Mail Sorting Center

The software system for the Copenhagen Mail Sorting Center (Fig. 3, [4][5]) was developed from 1974-1978. The center was designed to handle 130.000 parcels and 3 million letters per day arriving and departing on trucks or trains following a strict schedule. The main contractor was Boy Transportmateriel A/S.

The center comprised approximately 1000 conveyor belts, which if started or stopped at the same time (especially when loaded with mail bags or parcels) would have a severe impact on the power lines supplying the building. Therefore each conveyor belt was modeled as a component in the data-model of the software system with two flags indicating its ability to receive and deliver mail respectively.

When mail is delivered at the receiving end of a belt, its predecessor component turns its able-to-deliver true, and the belt component then issues a start command (bit) to its belt's motor. While the motor is running the component calculates when mail will reach the other end of the belt, at which point it raises its able-to-deliver flag. This is detected by the succeeding component, which then starts. In case the succeeding component is not able to receive mail (its able-to-receive flag is false) the belt motor will be commanded to stop.

The same happens when mail is no longer delivered from the belt's predecessor (its able-to-deliver flag turns false). The component will allow the belt to continue to run until a calculation determines that the belt is empty. Then the belt motor is commanded to stop and the component's ability-to-deliver flag is set to false. The effect propagates down the line of conveyor belt components (Fig.5).

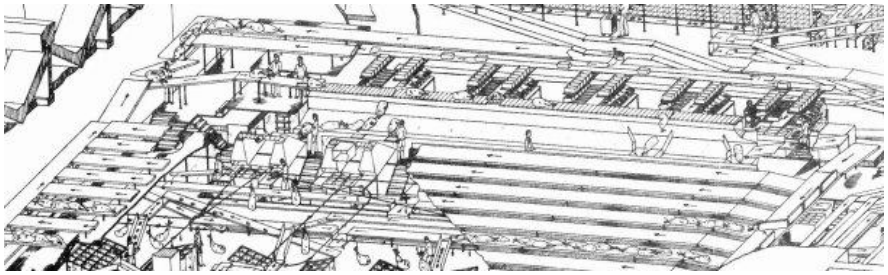

Fig. 5. Details of conveyor belt connections.

When a belt is intended for storage, the predecessor component is a photo cell component at the start of the belt, rather than another belt component. The photo cell, however, is modeled with similar flags, and the storage belt only moves as long as the photo cell component has its able-to-deliver flag true, e.g. while mail is blocking the view of the photo cell. This way mail is compacted on the belt. When mail reaches the other end of the storage belt (usually controlled by a photo cell component at the end of the belt now signaling able-to-receive false), the storage belt will indicate able-to-receive false to its predecessor (the photo cell component at the start of the belt). This not-able-to-receive flag is reflected to its predecessor (the component delivering mail to the storage belt). A storage management component will then choose another parallel storage belt to receive further mail. When emptying a storage belt, the belt component will act as a normal transporting belt, but it will still keep the able-to-receive flag false, so that no new mail will be received until the belt has emptied completely.

Thus the use of these "able-to" flags can control the progress of mail throughout the mail center irrespective of the type of equipment modeled, and only keep those conveyor belts running, that are in use. The "able-to" flags are the only way in which

the modeled components communicate, and the flags are investigated at each cycle through the data-model.

The center was controlled by 5 duplex hot stand-by computer systems for each section of the mail sorting process, a number of microprocessors, and a supervisory computer for the operators connected via asynchronous communication lines. The control computers were Control Data (CDC) Cyber 18-17 with 32-88kB memory, a memory-to-memory high-speed bus, and no backing stores.

About the Author

Otto Vinter is a consultant and mentor; managing process improvements for 15+ years; and development projects for 30 years. He was responsible for software process improvements at Brüel & Kjær and the driving force in activities on testing, requirements engineering, and agile development. He introduced defect analysis as an improvement technique and performed comprehensive analyses based on this.