Bridging the gap between control theory and its application to complex industrial processes requires translating process requirements into economical, reliable software—a task control engineers can now undertake.

Control Software Specification and Design: An Overview

Cláudio Walter, Universidade Federal do Rio Grande do Sul

Although much remains to be done in control theory, it has already developed a significant body of knowledge, composed of models and identification and optimization methods. This knowledge is often very sophisticated and, when applied to relatively rapid and complex industrial operations and design, it may require substantial engineering support. While operations can already be performed by computers, design remains largely a human domain where trial and error, experience and intuition come into play.

Current control system design for industrial applications corresponds essentially to stating the process requirements, specifying the functions to be performed, then implementing them in software. This article surveys techniques in the field of control software and design, noting particularly the need for clear specifications and engineering methods. It provides the basis for predicting the general direction of improvements in control system design allowed by the developments in computer technology and artificial intelligence.

Control system design needs

A glance at computing systems and process control applications reveals a significant reduction of the hardware cost-to-function ratio, leading to a much broader range of applications than imaginable 20 years ago. This trend is expected to hold during the eighties, due to higher scales of integration of electronic components. The resulting sophistication of the applications and the increased complexity of their control algorithms turn software, which includes system analysis, engineering, programming, and testing, into the most expensive item of a control system and often a significant part of the overall investment in an industrial plant.

Unfortunately, one cannot consider software production without thinking about software errors, which stem from several causes. To start with, the customer who orders a software system may not know precisely what he wants. When discussing his ambiguous, incomplete, and sometimes contradictory specifications with the supplier, both customer and supplier may make diverging assumptions, thinking that they understand one another. The supplier then proceeds to introduce his own errors through internal communication problems within his staff and through the very nature of computer programming.

The fault does not lie entirely with the people involved. The culprits are the traditional means for specification and development. Free-syntax, informal specifications are clearly inadequate to define software and lead to unforeseeable costs, delays, and often unsatisfactory results. It follows that practical formalisms and software production tools and controls are needed. Some have already been introduced. The existing efforts and their contexts merit consideration.

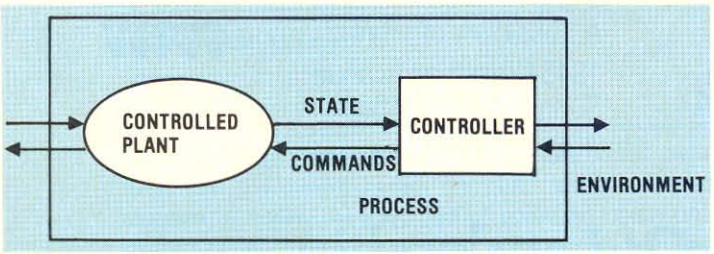

Figure 1. Partition of a process into controller and plant.

Control and software engineering

A control system, or process, can be divided into the controller and the controlled plant, as shown in Figure 1. A control design job can be broadly defined as specifying and implementing the functions that drive the inputs so that a plant performs a specified process. Design can be broken down into two successive, but often overlapping

0018-9162/84/0200-0020$01.00 © 1984 IEEE

tasks—control engineering and software engineering—as illustrated in Figure 2.

Given a plant and the process that it is expected to perform—the requirements specifications—control engineering establishes the functions of the controller. CE is supported by control and automata theories. A significant part of its task is performed during the user-designer interaction at the beginning of the project. It is straightforward when applied, for instance, to monovariable, constant-parameter, linear systems, where a satisfactory feedback can be quite easily established from the given plant transfer function and from the overall process-time and frequency-domain requirements. Unfortunately, most real processes are not so simple, and besides their mathematical complexity, the uncertainty of the incompleteness and inconsistency of the process specifications become sources of difficulty. Although such problems are typical of control engineering, tools borrowed from software engineering usually assist in structuring, consistency checking, and providing documentation facilities to solve them.

Software engineering, or more precisely control software engineering, implements the specified control function as an executable computer program, accommodating such criteria as safety, readability, and flexibility. While control software engineering tools may help in stating the function of the software to be generated, they generally provide only very broad guidelines—structural rather than functional—for the actual design of the software specifications from process requirements. Here control and software engineering may overlap because the process/ plant specifications are included in the control software specifications, instead of just leading to them. The overlap can be explained by:

- Designer psychology: It is convenient for the designer to simulate the process on paper and in his mind while developing the control specifications.
- Observer theory: It is often necessary for the controller to include a model—the observer—which reconstitutes the state of the process from limited captor information. This modeling occurs in some classes of adaptive control functions and very often with the sequential aspects of processes, which can be represented by finite automata.

Comparing control software and general-purpose software engineering, we see that both have software functional specifications as the starting point and an executable program as a product. CS engineering extends GPS engineering in the same way as, say, real-time Fortran extends common Fortran. CS can be specified using GPS specification languages, but the result will probably be harder to write and read because CS specification languages incorporate as primitives some constructs that would be expressed as composite instructions of GPS specification languages. These constructs—real-time, syn-

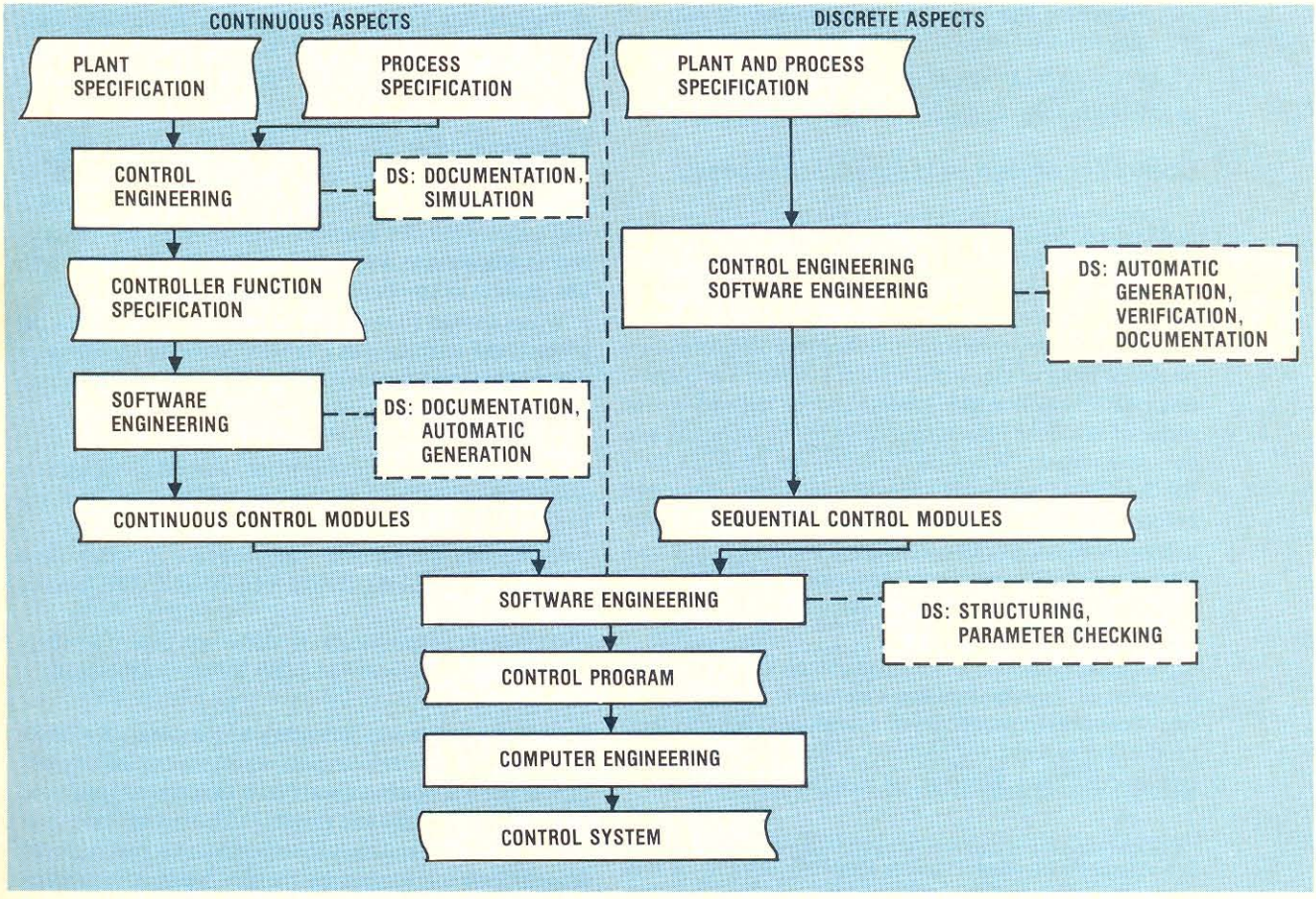

Figure 2. Specification and design of the controller. DS identifies functions of the development system.

chronization, communication, etc.—are related to control-system applications in environments with real-time constraints and often with a high degree of parallelism. Finally, the control program has to be executed on a computing system with appropriate hardware and software support, whose design/configuration is a computer engineering task.

During the last decade, software designers have tried to create control-software development systems that provide the designer-analyst-programmer with tools for the cost-effective generation of dependable software. The tools ease the designer's task by

- guiding his steps as he develops software requirements,
- providing parameter and range checking when continuous and sequential control modules are knit together into a control program, and
- automating the documentation.

Among the most significant design tools are

- the EPOS[1] specification system, with several modules now operational and others in development; its starting point is a formal model of control software requirements, which specifies the controller functions for each state and event;
- PCSL,[2] which is essentially a language based on the PSL general software specification system,[3] and its improved offshoot, Espreso[4];
- the SARS[5] system, inspired by R-nets[6]; and
- the Mascot[7] system, a software specification, development and management tool based on "channel and activity" nets.

These systems are based on graphical and/or textual specification formalisms.*

Controller design of industrial processes

The function of a development system is determined largely by the different aspects—sequential and continuous—presented by industrial processes. An industrial process, for instance, is composed of concurrent sets of sequential activities, each corresponding to a transformation on one or more of the attributes of the objects it acts upon. Each activity within the process can be defined by the range of attribute values acceptable for its incoming and outgoing objects. When these objects are observed from the outside, an activity is a step among others performed before, concurrently, or afterwards. But when an activity is considered intrinsically, it is always continuous and controlled by continuous control algorithms, which may be interpreted continuously or sampled for computer control.

The aspect that carries the most complex component of a process or its main function determines whether a process is considered sequential or continuous. A paper pulp production process, for instance, is generally regarded as continuous. It can, however, be regarded as a set of clearly identifiable sequential steps, each with its own input and output object specifications. In batch processes, such as steel production, the operation on a batch within each activity (such as melting or refining) is continuous, while the transfer of batches between furnaces and other equipment can be regarded as an interval in the sequence of events.

Sequential and continuous aspects differ considerably in difficulty of analysis, controller specification, and design techniques. The difference lies in the resolution characteristics produced when continuous phenomena are partitioned into discrete steps. Partitioning reduces resolution. On the other hand, increased resolution results in an increased bandwidth; the order of the significant derivatives also increases, and the system becomes harder to describe and to treat mathematically.

The sequential aspects of physical phenomena can be represented by finite automata, which are easy to construct from the informal understanding of the process. We have to consider only the successive states and the events that indicate the state transition of the process. These transitions are subject to few disturbances, and their consequences can be foreseen. At most, a disturbance might modify the expected behavior by taking the system into an exceptional, but represented state or by altering the timing of the expected transition.

The main formalisms derived from finite automata are

- state graphs, generally used in switching applications[8,9];
- finite interpreted Petri-nets,[10] which facilitate the specification of parallel activities; they are used as theoretical models for the process specification and control language Grafcet[11]; and
- LL (1) grammars, used for the specification of the process behavior.[12]

The grammar that specifies a process can be translated by a grammar transformer into a control program. Process specification grammars are rarely used, possibly because they require a bulky grammar transformer. Still the concept is elegant and may become of practical interest with the increase of computing power.

When applied to the sequential aspects of an industrial process, the operation of a control software specification and development system can be expressed by the following steps:

(1) Specify the controlled plant: generally, the plant is defined by the physical transformations it is expected to perform.
(2) Specify the process—the sequence in which these transformations should occur—and their parameters.
(3) From the specifications, derive the controller function and the corresponding software analytically.

Besides the automatic derivation of the controller, these formalisms allow the verification of aspects of interest, such as the boundedness and deadlock potential provided by analysis packages like Ovide.[13] The overlapping control engineering and software engineering tasks can be automatically performed by the development system, as shown in Figure 2.

*A detailed discussion of these systems falls beyond the scope of this article; refer to articles in the May 1982 issue of *Computer* (Vol. 15, No. 5) on requirements specifications for a survey covering several approaches and systems.

Continuous aspects of industrial processes are often difficult to represent accurately because of the quantity and resolution of the variables involved. Their behavior is affected by noises and other environmental disturbances. This complexity can be partially handled by adaptive control concepts and techniques, but as a whole, the automatic derivation of continuous control algorithms in practical applications is still very limited. As a consequence, the control engineering of continuous processes remains largely an iterative process, in which successive simulations and human interaction combine to yield satisfactory results. Development systems are used mainly to provide documentation and simulation support to the designer. And once the control function has been established, it is the task of software engineering to support the implementation by means of programming languages that allow the control function to be specified with a syntax related as nearly as possible to the nature of the application (Figure 2).

Over the long term, significant advances can be expected. One contribution will certainly come from the development of software engineering. Another, which is more significant from the control point of view, originates in the development of better process models and artificial intelligence. In the early nineties, fifth-generation computers are expected to provide the computing power necessary to develop expert systems—sophisticated programs and interpreters with predicate calculus and learning capacity. One of the most important results of this development will be an explanation of why designs are as they are; it will clarify a discipline that is now hidden behind largely intuitive and/or empirical decisions.

According to reports about present experimental systems,[14] control design expert systems will operate along the following lines:

(1) A human "expert" team will introduce and improve technology by developing mathematical, physical, chemical, economic, and ergonomical rules, as well as application-oriented process and control models and rules.
(2) With the assistance of an application-oriented dialog, users will introduce environment information, such as raw materials, human resources and equipment availability, and cost, in addition to requirements specifications that state desirable features, acceptable ranges of process, and plant variables.
(3) Users will extract controller specification and software and process simulation data from application-oriented dialogs.
(4) The systems' learning capacity will improve the quality of results and knowledge introduced in (1), by analyzing data provided by operating plants.

Present-day computer-aided specification and design tools already emulate particular classes of applications of these "true" expert systems. The pressing need for faster design of more dependable software, together with the computing resources now at our disposal, promise a rich and stimulating field of software research for the next 10 to 20 years. ■

Acknowledgments

This work is partially supported by grant 30.1461/81 of the CNPq (Conselho Nacional de Desenvolvimento Cientifico e Tecnológico).

References

1. J. Biewald et al., "Real-Time Features of EPOS: Formulation, Evaluation and Documentation," *Proc. 10th IFAC/IFIP Workshop Real-Time Programming,* 1980, pp. 95-100.
2. J. Ludewig, "Process Control Specification in PCSL," *Proc. 10th IFAC/IFIP Workshop Real-Time Programming*, 1980, pp. 103-108.
3. D. Teichroew and E. A. Hershey, "PSL/PSA: a Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Trans. Software Eng.*, Vol. SE-3, No. 1, Jan. 1977, pp. 41-48.
4. J. Ludewig, "ESPRESO—A System for Process Control Software Specification," *IEEE Trans. Software Eng.*, Vol. SE-9, No. 4, July 1983, pp. 427-435.
5. W. K. Epple and G. R. Koch, "SARS: a System for Application-Oriented Requirements Specification," internal report, Univ. of Karlsruhe, 1983.
6. M. V. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Trans. Software Eng.* Vol. SE-3, No. 1, Jan. 1977, pp. 50-69.
7. *The Official Handbook of Mascot,* Mascot Suppliers Association, London, 1980.
8. J. V. Landau, "State Description Techniques Applied to Industrial Machine Control," *Computer,* Vol. 12, No. 2, Feb. 1979, pp. 32-40.
9. B. Taylor, "A Method for Expressing the Functional Requirements of Real-Time Systems," *Proc. 10th IFAC/IFIP Workshop Real-Time Programming,* 1980, pp. 111-120.
10. J. L. Peterson, "Petri-nets," *Computing Surveys,* Vol. 9, No. 3, Sept. 1977, pp. 223-252.
11. AFCET, "Pour une representation normalisée du cahier de charges d'un automatisme logique," *Automatique et Informatique Industrielles,* No. 61, Nov. 1977, pp. 27-32.
12. F. Anceau and J. Bordier, "A Syntactic Method for Programming Simple Industrial Control Applications with Microprocessors," *Euromicro Symp. Large Scale Integration,* North-Holland, Amsterdam, 1978, pp. 324-328.
13. E. Le Mer, "OVIDE: A Software Package for Verifying and Validating Petri-nets," *Proc. Third IFAC/IFIP Symp. Software for Computer Control,* 1983.
14. "The Concept of Expert Systems," *Infotech State of the Art Report,* Series 9, No. 3, 1981.

Cláudio Walter is a professor for computer control at the Universidade Federal do Rio Grande do Sul in Porto Alegre, Brasil. He has done research in the field of requirements specifications and worked with microprocessor applications to instrumentation, besides consulting in industry. He holds a BSEE and a MSc in computer science from UFRGS. From 1978 until 1981, he was at the Institut National Poly-technique de Grenoble, France, where he earned a Docteur-Ingénieur degree.

His address is Pós-Graduação em Ciencia da Computação/Dep. Eng. Elétrica, Universidade Federal do Rio Grande do Sul, 99 Av. Oswaldo Aranha, Caixa Postal 1501, 90.000 Porto Alegre, RS, Brasil.