55

A perspective on a Eunice-based control system development environment

Y. Kawata, A. Yabu, M. Maekawa, T. Kawase, and H. Kozuka
Graduate School of Information Systems, University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo, JAPAN 182 Ph: +81-424-85-5477 FAX: +81-424-85-5471 E-mail: {yasuro, aki, maekawa, kawasedo, kozuka}@maekawa.is.uec.ac.jp

H. Kobayashi
NTT Data Communications Systems Corporation
Toyosu Center Building, 3-3-3 Toyosu, Koto-ku, Tokyo, JAPAN 135
Ph: +81-3-5546-8963 FAX: +81-3-5546-8990 E-mail: hisa@dc.open.rd.nttdata.jp

Abstract

The long-term objective of our Eunice Project is to identify and develop an integrated environment for Computer-Aided Control System Development, or CACSD. The most salient feature of the project is that all the development lifecycle, from specification to implementation, is to be supported by a single language, Eunice.

This paper discusses the features required of Eunice: early executability by abstract descriptions, refinement mechanism, mixture of discrete models and continuous models, combination of simulation and execution, prespecified adaptive components, support for development aspects that are not directly related to control, extension mechanism for multi-paradigm support, etc.

This paper also states the current status of the project.

Keywords

CAE, CACSD, control system development, executable specification language, simulation, refinement, real-time, Eunice

1 INTRODUCTION

Our research team has been actively pursuing the Eunice Project. This is a research program whose long-term objective is to identify and develop a solid, integrated generic environment for Computer-Aided Control System Development, or CACSD. This paper presents an overview of our approach towards this goal, addressing the technical issues involved. It summarizes the results we have obtained so far. Finally it compares our approach with relevant works.

Q. Sun et al. (eds.), *Computer Applications in Production Engineering*

2 THE EUNICE LANGUAGE AND ITS ENVIRONMENT

2.1 Full-lifecycle-support language

The most salient feature of the Eunice Project is the environment is to be built around Eunice, one single formalism or language. Eunice 'all-in-one' documents supports the full development lifecycle, i.e., from specification to implementation, and also other nontechnical activities involved in development, such as cost estimation. By so doing, in principle it can avoid the following shortcomings inherent in traditional Waterfall Development Lifecycle (Royce, 1970), a still prevalent lifecycle model of information system development in general:

- Formalisms for different phases are different and two formalisms of adjacent phases are often not completely compatible with each other. Some portion of the information obtained in a phase may not be carried on to the next phase properly.
- When an error in a phase is found at a later phase, oftentimes it is corrected only at the later phase and documents/codes of the earlier phase are not updated accordingly. In the end, nothing but source codes correctly reflects the current status of the application. This is more or less a human factor.
- In spite of the popular belief that specifications can be completely separated from implementation, Swartout and Balzer (1982) clearly showed that specification of a system and its implementation cannot be strictly separated, and inevitably they are intertwined.

2.2 Procedure of control system development with Eunice

Eunice-based development is expected to proceed as depicted in Figure 1.

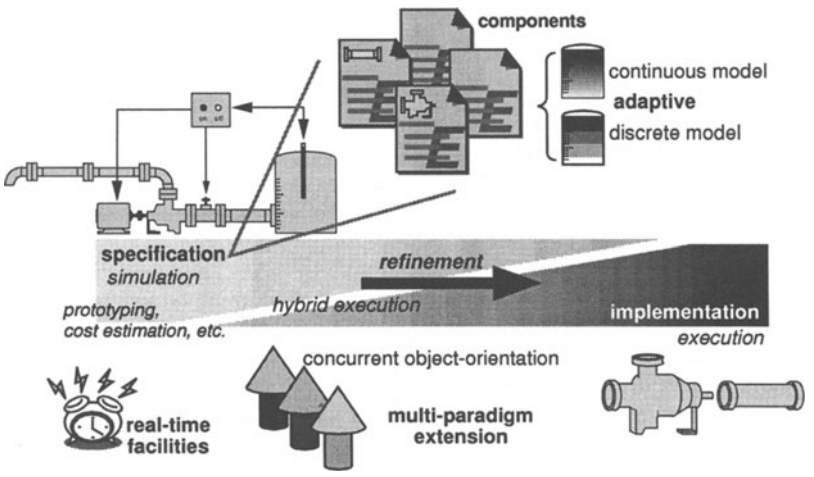

Figure 1 Envisaged development procedure with Eunice

The development begins with acquisition and specification of requirements. For writing

specifications, Eunice provides a library of prespecified components and the developer can use them 'as-is' or make necessary modifications by inheritance mechanism in the framework of the standard object-orientation.

These Eunice software components are manipulated, edited, and assembled on a user-friendly multiple window system. For a simple system, when assembled these components automatically work in concert without any adjustments or modifications.

The environment of a control system is the system minus its controller. The controller interacts with, and tries to control it. For a specification of a control system to be complete, its environment also has to be specified. Some prespecified components model objects that belong to environments of control systems: physical devices or appratuses such as motors, tanks, pipes, etc., and such objects as 'room' for a heating system.

The components that model physical devices contain the data of the physical properties of the devices they represent, e.g., nominal radii for pipes. When these components are connected, not only logical component-to-component consistency is validated but also physical component-to-component consistency is validated, e.g., whether nominal radii of connecting pipe components are the same or not.

Components also carry data that are not relevant to control, such as prices for various conditions (bulk purchase, academic discount, etc.), or iconic representation on screens.

Specifications written in Eunice are executable even when they are abstract and not detailed. By executing them, the developer sees whether he has specified what he wants to. By having the customer see the execution, optionally via a visualization tool with pseudo-animation facility, he can make sure whether he understands what the customer wants.

The developer gradually refine and incrementally detail the specifications, and make them closer to implementation, till finally all the control logic has been described, all the physical devices are installed, and the implementation is complete. At any time of this gradual transition, he can execute them.

2.3 Features of Eunice

To attain the aforementioned goal and development procedure, Eunice has the following features, each of which can be a research topic by itself:

Early executability by abstract descriptions Eunice allows more abstract descriptions than ordinary programming languages do, and incomplete descriptions as well. This leads to early executability. Missing information may be supplemented at run time by interaction with users.

This feature lets developers see what they have written represents what they have in mind and also understand correctly what the clients want. If misunderstandings between the two parties surface later in the development, it cost much to fix. Eunice considerably reduces this risk.

As you will see shortly, abstract, specification-level descriptions in Eunice are utilized till later phases, early executability in Eunice is equivalent to capability of rapid prototyping.

Refinement Automatic transformation from a specification to its implementation is indeed ideal (Zave, 1982), but it cannot be done without human assistance; thus, Eunice provides more practical manual refinement mechanism.

Eunice's refinement is not *refinement by replacement*, but instead, *refinement by addition*. In other words, in Eunice, the *pre*-refinement descriptions continue to constrain the *post*-refinement descriptions. The latter can be tested against what the former dictates. After ample testing, it is also possible to switch the semantics of a particular refinement to refinement by replacement, for better performance.

Real-time facilities Real-timeness is usually the major concern of control systems. Eunice provides real-time processing facilities such as deadline specifications on block execution, message sending, and message waiting, and exception handling including timeout handling.

Mixture of discrete models and continuous models The behavior of a controller can usually be described in a discrete model. To model objects that belong to the environment, however, discrete models are not always suitable and continuous models have to be supported. It also applies to analog devices in the controller, if any.

Since some objects need discrete models and others need continuous models, for the whole system the mixture of discrete models and continuous models is supported.

Combination of simulation and execution System descriptions in Eunice goes gradually from the stage of purely specification to the stage of purely implementation. Between these two extreme stages, the descriptions are partly specification and partly implementation, with a varying ratio. We call execution of the system at an intermediate stage *hybrid execution*. Since execution of specification can be seen as simulation, hybrid execution is a combination of simulation and execution.

The difficulty of hybrid execution is how to synchronize simulation part and execution part. The simulation part progresses according to the virtual time while the execution part progresses according to the real time. If done on a single processor, the simulation part, especially when complicated evaluation procedures of continuous variables are involved, will degrade the performance of the execution part and thus inevitably dilute the validity of the obtained results.

Component-based construction with prespecified components Object-orientation, fundamental paradigm adopted in Eunice, is reputed to be effective in class- or object-based reuse. The 'vanilla' object-orientation, however, is not sufficient for component-based construction. For example, name of the messages a server provides have to be known to clients *beforehand* to make them work in cooperation. Eunice provides channel interface between components to enhance component-wise independence.

Adaptive components To facilitate component-based construction, components have to be *adaptive*. It adapts to the context in which it is used. Take a Eunice component that models a tank for example. It might be used to 'pour ten liter of water in' (discrete) or 'pour water in at the rate of one liter of per second for ten seconds' (continuous). Prespecified components such as the one for this tank cannot know beforehand how it will be used in conjunction with other components. Eunice provides framework with which such adaptation is possible.

Support for development aspects not directly related to control Accurate modeling of controllers and their environments is the crucial part of control system development, but there are other factors involved in it. Physical constraints (part of them may be implicitly incorporated into models), cost issues (not likely to be incorporated into models at all), etc., also have to be given consideration from the specification phase. It is well probable that developers weigh their choices in control methods and devices against costs they incur (Swartout and Balzer, 1982).

Since all descriptions are in Eunice, in combination with other Eunice features, developers can try connection of components with other components, overall behavior of the system being constructed, cost estimation, etc., on a user-friendly multiple window system. Thus, Eunice, along with its environment, supports all aspects of decision-making in an integrated manner.

Manufacturers of (physical) components for control systems supply Eunice (software) components because the Eunice components replace conventional catalogues printed on paper. Because of this, Eunice components are circulated at a cheap price, or maybe even distributed for free.

When this perspective is realized, both control system developers and component manufacturers can benefit. Developers can save much of their work in specification validation, design, prototyping, and implementation by utilizing supplied Eunice components. Component manufacturers can convince developers of advantages of their products in an objective manner—Developers can easily see choice of a certain component will improve the overall efficiency of control up to 20 %, or reduce the total cost by 5 %, etc., by executing the corresponding Eunice component with the rest of the system.

Extension mechanism for multi-paradigm support Control systems have vast varieties from 'large' ones in size such as chemical/pharmaceutical plants to 'small' ones such as home appliances. Each minor field has its own paradigm fitted for it, and also long-lived conventions, traditions, etc., specific to that minor field. It is not realistic to try to provide a single paradigm that is suitable for all the minor fields. Instead Eunice provides a fundamental paradigm, namely, concurrent object-orientation, and lets extension of minor-field-specific paradigm.

2.4 Supporting environment for Eunice

The supporting environment for Eunice has the following features:

- It runs under a multiple window system. Information is presented visually, by icons, graphs, etc., in an intuitively easy-to-understand manner. Various kinds of manipulation can be done in a point-and-click mouse interface.
- To facilitate quick and easy searching and browsing of library components, components written in Eunice are stored in a database, not just as files.

3 CURRENT STATUS

Our Eunice project is still in its infancy. As of this writing, Kawata, Kobayashi, Yabu, Onogawa, Kawasaki and Maekawa (1994) discusses gradual, incremental development in Eunice and its supporting visual environment. Kawata, Onogawa, Yabu, Maekawa, Kawasaki and Kobayashi (1994) discusses component-based development, incorporation of physical properties into components, interactive validation of component-to-component connection, and its visual manipulation environment. Kawata, Yabu, Kobayashi, Onogawa, Kawase, Kozuka and Maekawa (1994) discusses adaptive software components that automatically select their behavior from continuous descriptions and discrete ones according to the context in which they are used.

4 RELATED WORK

We have not yet seen any equivalent or similar approaches to ours as a whole. Regarding our individual ideas, however, some parallels can be found.

Open architecture of integrated CACSD environment is discussed in (Andersson, Mattsson and Nilsson, 1992), (Barker, Chen, Grant, Jobling and Towensend, 1993), etc.. They, however, are too concerned with the traditional concept of control system development, i.e., 'control first, other things later', and other aspects of development as pointed out in this paper are not considered. For example, (Barker et al., 1993) bases its discussion on the ECMA/NIST's reference model of integrated CASE environment (so-called 'toaster model') (NIST ISEE Working Group and the EMCA TC33 Task Group on the Reference Model, 1991), and simply replaces its data services with 'modeling services'. Data services in ECMA/NIST model handles *metadata*, i.e., processes involved in development, but the replacing modeling services do not support this.

In the field of simulation, some general-purpose simulation languages support combination of continuous models and discrete models (e.g., SYSMOD (Smart and Baker, 1984), and COSMOS (Kettenis, 1992)), and also those dedicated for chemical engineering (e.g., (Barton and Pantelides, 1994) and (Zentner, Elkamel, Pekny and Reklaitis, 1994)). Their one and only goal is simulation itself and the models that have been built for simulation cannot directly be used for later development processes.

In the field of Computer-Aided Design, or CAD, STEP/EXPRESS (Schenck and Wilson, 1994) (ISO TC184/SC4/WG PMAG, 1992) (ISO TC184/SC4/WG5, 1991), is now an ISO draft of product specification, and thus physical properties can well be described in this framework; however, because of its goal, the EXPRESS language lacks ability to express dynamic, behavioral nature of products. FDL (Imamura, 1992) supports interactive component assembly on computers, guided by constraints specified for each component. Some works in so-called intelligent CAD incorporate description of dynamic behavior into traditional CAD (e.g., (Nayak, Joskowicz and Addanki, 1992)).

Our hybrid execution is in line with 'hybrid simulation' in the works of Arano *et. al.* (Arano, Chang, Mongkolwat, Liu and Shu, 1993; Arano, Chang, Aono and Fujisaki, 1993). Our approach and theirs differ in that they treat specification and implementation two completely different phases, and use different formalisms for each.

5 CONCLUSIONS

This paper has stated the objective of the Eunice Project, which is to identify and develop an integrated environment for Computer-Aided Control System Development, or CACSD. The most salient feature of the project was that all the development lifecycle, from specification to implementation, is to be supported by a single language, Eunice.

This paper has discussed the features required of Eunice: early executability by abstract descriptions, refinement mechanism, mixture of discrete models and continuous models, combination of simulation and execution, prespecified adaptive components, support for development aspects that are not directly related to control, extension mechanism for multi-paradigm support, etc.

This paper has presented the current status of the project and compared our approaches with others' works.

REFERENCES

Andersson, M., Mattsson, S. E. and Nilsson, B. (1992), On the architecture of CACE environments, *in* H. A. Baker, ed., 'Computer Aided Design in Control Systems', IFAC Workshop Series, International Federation of Automatic Control, Pergamon Press, pp. 41–46. Selected Papers from the IFAC Symposium, Swansea, UK, 15–17 July 1991.

Arano, T., Chang, C. K., Aono, H. and Fujisaki, T. (1993), A new simulation technique in prototyping development, *in* 'Proceedings of Summer Computer Simulation Conference (SCSC) '93', pp. 990–995.

Arano, T., Chang, C. K., Mongkolwat, P., Liu, Y. and Shu, X. (1993), An object-oriented prototyping approach to system development, *in* 'Proceedings of the Seventeenth Annual International Computer Software & Applications Conference (COMPSAC 93)', IEEE Computer Society Press, pp. 56–62. November 1-5, 1993, Phoenix, Arizona, USA.

Barker, H. A., Chen, M., Grant, P. W., Jobling, C. P. and Towensend, P. (1993), 'Open architecture for computer-aided control engineering', *IEEE Control Systems* **13**(2), 17–27. Special Issue on Computer-Aided Control Systems Design.

Barton, P. I. and Pantelides, C. C. (1994), 'The modelling of combined discrete/continuous processes', *AIChE (American Institute of Chemical Engineers) Journal* **40**(6), 966–979.

Imamura, S. (1992), FDL: A constraint based object oriented language for functional design, *in* G. J. Olling and F. Kimura, eds, 'Human Aspects in Computer Integrated Manufacturing', IFIP Transactions B: Applications in Technology, North-Holland, pp. 227–236. Proceedings of the IFIP TC5/WG 5.3 Eight International PROLAMAT Conference, Man in CIM, Tokyo, Japan, 24–26 June 1992.

ISO TC184/SC4/WG PMAG (1992), STEP part 1: Overview and fundamental principles, ISO CD 10303-1, International Standard Organization. Owner: Howard Mason/John Rumble.

ISO TC184/SC4/WG5 (1991), EXPRESS language reference manual, Document N. 5, International Standard Organization. Release Draft. ISO 10303-11. Owner: Philip Spiby.

Kawata, Y., Kobayashi, H., Yabu, A., Onogawa, K., Kawasaki, A. and Maekawa, M. (1994), Eunice/ITRON: A control system development environment for ITRON machines, *in* 'Proceedings of the 11th TRON Project International Symposium', IEEE Society Press, pp. 91–105. December 7–10, 1994, Tokyo, Japan.

Kawata, Y., Onogawa, K., Yabu, A., Maekawa, M., Kawasaki, A. and Kobayashi, H. (1994), 'EVE: A graphical specification environment', Submitted to *the Transactions of the Institute of Electronics, Information and Communication Engineers*. In Japanese.

Kawata, Y., Yabu, A., Kobayashi, H., Onogawa, K., Kawase, T., Kozuka, H. and Maekawa, M. (1994), Eunice adaptive components: Modeling external objects in control systems for better construction and validation of specifications, *in* C. Mingins and B. Meyer, eds, 'Technology of Object-Oriented Languages and Systems TOOLS 15', Prentice-Hall, pp. 45–55. Proceedings of the fifteenth International Conference TOOLS Pacific 94, Melbourne, Australia, November 28–December 1, 1994.

Kettenis, D. L. (1992), 'COSMOS: A simulation language for continuous, discrete and combined models', *Simulation* **58**(1), 32–41.

Nayak, P. P., Joskowicz, L. and Addanki, S. (1992), Context-dependent behaviors: A preliminary report, *in* D. C. Brown, M. B. Waldron and H. Yoshikawa, eds, 'Intelligent Computer Aided Design', IFIP Transactions B: Applications in Technology, North-Holland, pp. 237–250.

NIST ISEE Working Group and the EMCA TC33 Task Group on the Reference Model (1991), Reference model for frameworks of software engineering environments, NIST Special Publication 500-201, NIST (National Institute of Standards and Technology, United States of Commerce). Technical Report ECMA TR/55, 2nd Edition.

Royce, W. W. (1970), Managing the development of large software systems: Concepts and techniques, *in* 'Proceedings of the IEEE WESCON', IEEE Press, pp. 1–9. Los Angeles, CA, USA, August 25–28, 1970. Reprinted in 'Proceedings of the Ninth International Conference on Software Engineering', pp. 328–338, IEEE Press, 1987. Monterey, CA, USA, March 30–April 2, 1987.

Schenck, D. A. and Wilson, P. R. (1994), *Information Modeling: the EXPRESS Way*, Oxford University Press.

Smart, P. J. and Baker, N. J. C. (1984), SYSMOD—an environment for modular simulation, *in* 'Proceedings of Summer Computer Simulation Conference', North-Holland, pp. 72–82.

Swartout, W. and Balzer, R. (1982), 'On the inevitable intertwining of specification and implementation', *Communications of the ACM* **25**(7), 438–440.

Zave, P. (1982), 'An operational approach to requirements specification for embedded systems', *IEEE Transactions on Software Engineering* **SE-8**(3), 250–269.

Zentner, M. G., Elkamel, A., Pekny, J. F. and Reklaitis, G. V. (1994), 'A language for describing process scheduling problems', *Computers and Chemical Engineering*. To appear.

BIOGRAPHY

Yasuro Kawata received the M.S. degree in computer science from the University of Tokyo and is currently a research associate at the Graduate School of Information Systems, University of Electro-Communications. His research interests include specification languages and methods and environments for computer-aided control system development.

Akifumi Yabu is currently a graduate student at the Graduate School of Information Systems, University of Electro-Communications. His research interests include control system specification, reusable adaptive components, and continuous/discrete combined system descriptions.

Mamoru Maekawa received the Ph.D. degree in computer science from the University of Minnesota. He is currently a professor and the head of the Department of Information Systems Design, Graduate School of Information Systems, University of Electro-Communications. His research interests include software engineering, distributed systems and multimedia.

Hisahiro Kobayashi received the M.S. degree in computer science from the University of Tokyo and is currently employed at NTT Data Communications Systems Corporation. His research interests include specification and programming of control systems, and architecture for distributed online transaction systems.

Both **Tomohiro Kawase** and **Hitoshi Kozuka** are currently undergraduate students at the University of Electro-Communications.