Automatic Programming for Sequence Control

Hiroyuki Mizutani, Yasuko Nakayama, Satoshi Ito, Yasuo Namio-ka, and Takayuki Matsudaira, Toshiba Corporation

Industrial plants are controlled using sequence control programs running on programmable controllers. Sequence control program design has been carried out manually, and an increase in applications of programmable controllers has caused a shortage of programmers. Therefore, automatic programming systems are strongly required in this field.

Controllers receive operation signals from plant operators and current plant states through sensors, then select actions that have to be executed. Sequence control programs consist of a large amount of control logic (about 100K program steps) for such decisions. The following problems were found in previous manual designs of sequence control programs:

First, control logic is often omitted.

Second, programs might include some mutual contradictions.

Third, information that is necessary to complete one program step is distributed in several different kinds of specification document. It costs too much time for program designers to understand specifications.

Fourth, alteration of control specifications often occurs, resulting in a wide range of program modifications.

The purpose of the automatic programming system (CAD-PC/AI) described in this chapter is to reduce these difficulties to increase productivity and improve the quality of sequence control program design. Moreover, it aims to facilitate a systematic accumulation of design knowledge.

There are two kinds of design knowledge used in generating sequence control programs: One is knowledge about the environment in which the programs work. The other is the specific programming knowledge for plant control.

We found through an analysis of designers' behavior that knowledge about the environment (that is, plant) plays an essential role throughout the entire life cycle of software development: requirement analysis, specification validation, implementation, testing, and maintenance. This knowledge constitutes a model of the plant that is to be controlled and leads us to propose a model-based automatic programming paradigm. Under this paradigm, the plant model supports every task in the software life cycle.

The second significant kind of knowledge is for refining specifications to target program codes. It appears that two kinds of programming knowledge are involved: One is to find reusable program parts suitable to given specifications. The other is to select a program skeleton and refine it in a stepwise fashion, according to the specifications, into concrete programs when program parts cannot be reused.

We chose the knowledge-based approach to develop CAD-PC/AI. The significant innovations are as follows:

First, it is one of the first knowledge-based systems in the plant control program design domain in which knowledge about the environment, as well as programming knowledge, is crucial.

Second, it demonstrates a new technology for making a knowledge base widely applicable, that is, the generic-specific modeling technique and model transformation discussed later.

Problem and Approach

A plant system includes operators, operation devices, programmable controllers, plant machines, actuators, sensors, and products, as shown in figure 1.

Control programs in conventional problem-oriented languages (for example, LADDER DIAGRAM) are written at the signal level—input-output (I-O) signals of programmable controllers—as shown in figure 2.

Because these programs have become increasingly complex to implement, they are still being manually designed; as a result, the process has begun to suffer from several of the problems that were

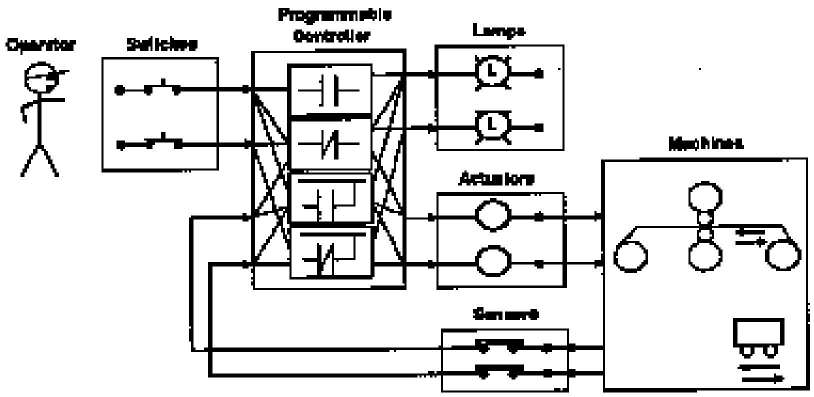


Figure 1. Conceptual Block Diagram for Plants.

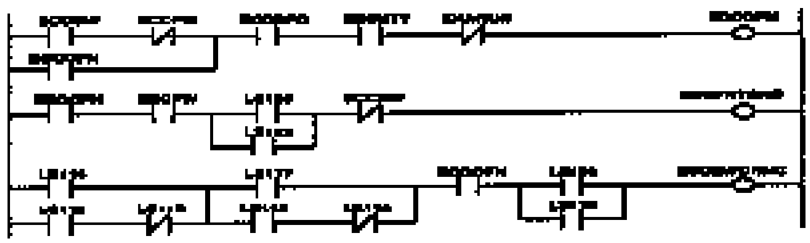
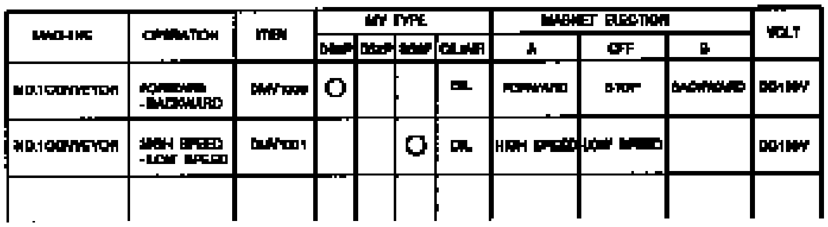


Figure 2. Example of Control Programs Written in LADDER DIAGRAM.

previously mentioned.

At the first stage of automatic programming system development, we established the software life cycle that we describe here. It was set up similar to conventional design processes so that designers would be able to easily transfer to the new system and maintain it. Because of this policy, it was necessary to simulate designers' conventional thinking on the computer system. Therefore, AI techniques were considered promising.

Previously, automatic programming research was based on the theorem-proving approach (Manna and Waldinger 1980), the program-transformation approach (Fickas 1985; Darington 1981; Green and Westfold 1982), and the knowledge-based approach (Barstow 1985; Lubars and Harandi 1987; Smith, Kotik, and Westfold 1985; Neighbors 1984). We selected the knowledge-based approach, where an informal high-level specification would be attainable, and prototyping would be easy; moreover, a conventional program-parts database could be used.



MACHINE	OPERATION	ITEM	MV TYPE				MAGNET ELECTION			VOLT
			[illegible]	[illegible]	[illegible]	[illegible]	A	OFF	B	
NO.1CONVEYOR	FORWARD -BACKWARD	[illegible]	○			OIL	FORWARD	STOP	BACKWARD	[illegible]
NO.1CONVEYOR	HIGH SPEED -LOW SPEED	[illegible]			○	OIL	HIGH SPEED	LOW SPEED		[illegible]

Figure 3. Example of Machine Specification.

Requirement Analysis Phase

Requirement analysis means deriving detailed specifications from brief requirements given in terms of the structure and operation of the plant. There are two aspects to requirement specification: One is *machine specification* (figure 3), which gives a static description of the plant in terms of actuators, sensors, operation devices, interlocks, and so on. The other is *control specification* (figure 4), which sets out the operations that the plant is required to perform.

In figure 4, a box represents an action, and a horizontal bar represents a transition. We set composite-action–level specifications as informal high-level specifications. *Composite action* is an abstract description of possible machine actions or states that can be broken down into some set of serial or parallel primitive actions or states. Detailed specifications, such as speed and subsidiary actions, are not described at this level. For example, “move forward” can later be broken down into “move forward at low speed until some conditions become true, and then move forward at high speed.” This high-level specification brings control design closer to the designers’ conceptual level, making design more natural.

In the new automatic programming system, a generic model constructs a specific model by interpreting machine specifications. These models are discussed later. The generic model determines a structural representation using the general knowledge of the functional structure of such plants. At the same time, it derives the detailed machine behavior using the general knowledge about machine operations and translates incomplete and ambiguous control specifications into detailed specifications.

Specification Validation Phase

The conventional testing method is based on a comparison of the actual behavior of the programs with the user’s intent. It is carried out using a special-purpose plant simulator after implementation is complete. If mismatches are detected, the implemented programs must be modified.

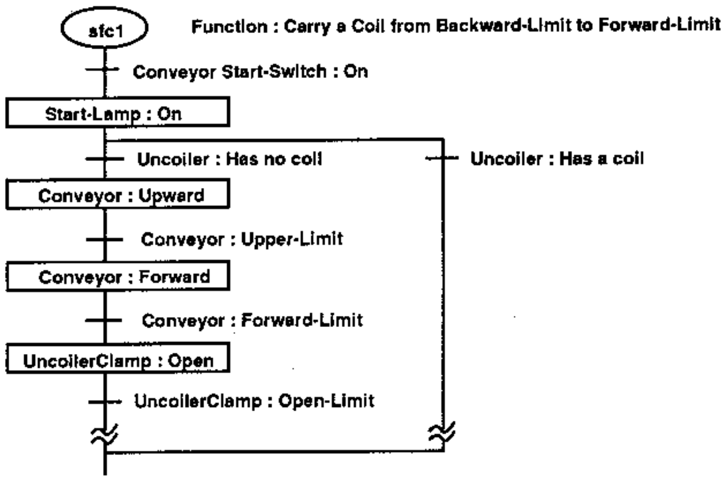


Figure 4. Example of Control Specification.

In the new system, the plant model supports specification validation. A symbolic simulation is performed using the detailed machine behavior, as represented by transitional relations between machine actions and states in the specific model.

Implementation Phase

Implementation is carried out by selecting suitable program parts and modifying them according to the specifications. Sequences that cannot be covered by program parts are refined using the program pattern in a stepwise fashion to create detailed programs. The specific model provides the knowledge necessary for these refining processes.

Maintenance Phase

Maintenance should be implemented by modifying the specifications and reimplementing them by replaying the development.

The plant must satisfy two requirements:

Task independent: The model must support the entire design process previously mentioned. There are different kinds of tasks in the design process. General-purpose modeling techniques must be developed to support every task. The knowledge-compilation technique (Chandrasekaran and Mittal 1983; Araya and Mittal 1987; Brown and Sloan 1987; Keller et al. 1989) was suggested based on a similar idea. Knowledge compilers facilitate knowledge reuse, and the same knowledge can be used for more than one purpose.

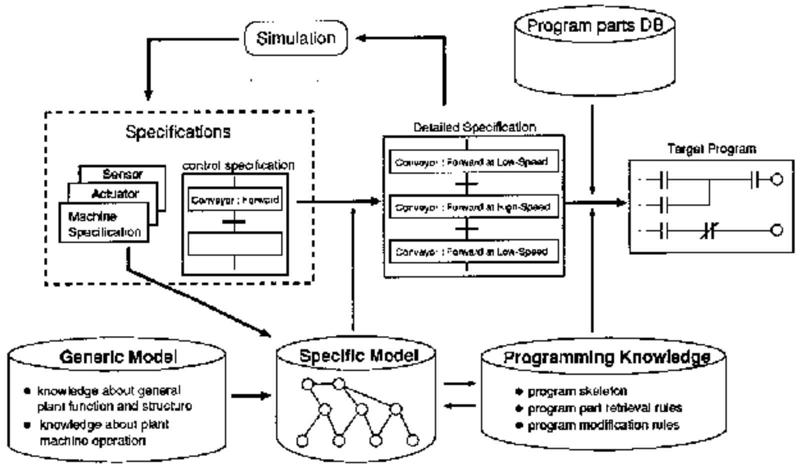

Simulation
Program parts DB
Specifications
Sensor
Actuator
Machine Specification
control specification
Conveyor : Forward
Detailed Specification
Conveyor : Forward at Low-Speed
Conveyor : Forward at High-Speed
Conveyor : Forward at Low-Speed
Target Program
Generic Model
knowledge about general plant function and structure
knowledge about plant machine operation
Specific Model
Programming Knowledge
program skeleton
program part retrieval rules
program modification rules

Figure 5. CAD-PC/AI Flow Diagram.

Application independent: The model must support general automatic programming for plant control. A common problem exists in conventional domain-specific expert systems: The knowledge base must be revised for each application because most of these systems rely on a large amount of ad hoc knowledge. To overcome this problem, modeling techniques must be developed that support every application in a specific domain, such as plant control.

System Description

Under this paradigm, we built the automatic programming system (Ono et al. 1988; Nakayama et al. 1990; Mizutani et al. 1991), as shown in figure 5. It works on the AS4000 workstation. We developed and used a knowledge description language in Lisp. It has facilities for frame representation, rule representation, and object-oriented programming. Program parts are stored in a relational database (RDB), and the knowledge description language has an SQL interface. Designers input specifications through a dedicated editor.

Model-Based Approach

We propose the modeling techniques that are outlined in the following subsections.

Generic and Specific Models

The plant model is composed of two parts: One is a generic model that contains knowledge used by system designers in the requirement analysis phase of control systems for a particular class of plants. It includes the functional structure of such installations, types of machine behavior, and expertise about plant control. The generic model is constructed by collecting the practical knowledge of experts and generalizing it. The same model is applicable to all plants of the same type; for example, the generic model of a steel plant is used for a hot-strip mill, a tandem cold mill, a processing line, and so on.

The other part is a specific model that contains knowledge used in the specification validation and implementation phases. This knowledge includes the structure, machine behavior, and constraints of a single target plant. This specific model is derived from the generic model according to the specifications of the target plant.

Extended Semantic Network

The generic model is represented in an extended semantic network that contains conditional relations in addition to the conventional semantic network. The conditional relations are associated with certain conditions. When the conditions are valid with regard to the specifications, the relation is reflected in the specific model. This representation makes the model flexibly accessible.

Furthermore, it has an object-oriented facility. The model-derivation procedures, mentioned previously, are represented as methods. Conditional relations in the generic model are instances of classes and, as such, are able to inherit the methods. As a result, appropriate specific models are built by interpreting the generic model with regard to the user-defined specifications of the target plant.

Model Transformation: The Design Process

The design process was considered as an iterative model transformation from abstract level to detailed description. In Gero (1990), a *design prototype* is a conceptual schema for representing a class of generalized functions, structures, behaviors, and relationships that are derived from alike design cases. In addition, routine design is viewed as a design prototype instance refinement.

The sequence control program design described in this chapter is a routine design, and the generic model can be considered one of the design prototypes. Figure 6 shows the model transformation in CAD-PC/AI. The refinement in the transformation is guided by input specifications. The generic model represents general knowledge about plant

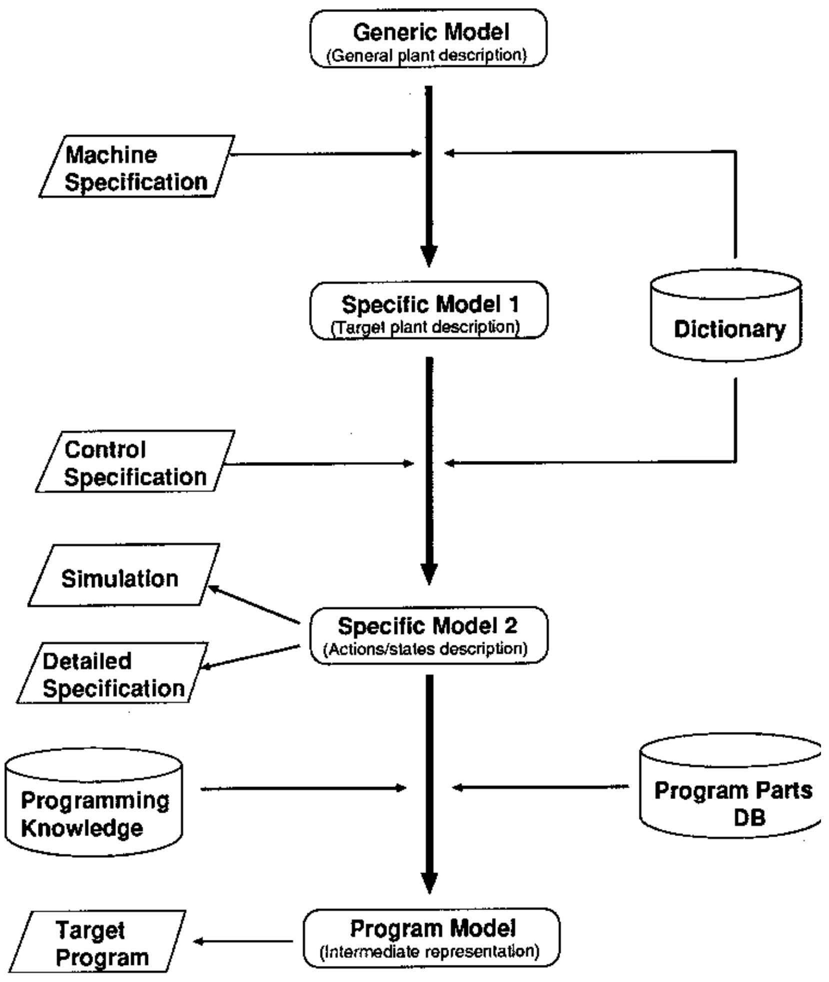


Figure 6. Model Transformation and Refinement in CAD-PC/AI.

functions, structures, behaviors, and relationships as well as expertise about plant control. The general knowledge is independent of the individual target plant. Interpreting a machine specification, CAD-PC/AI understands how the structure, represented in the generic model, is implemented in a target plant. In other words, the functions, structures, and behaviors become associated with target plant machines in the specific model 1 in figure 6, so that expertise about plant control becomes applicable to the target plant.

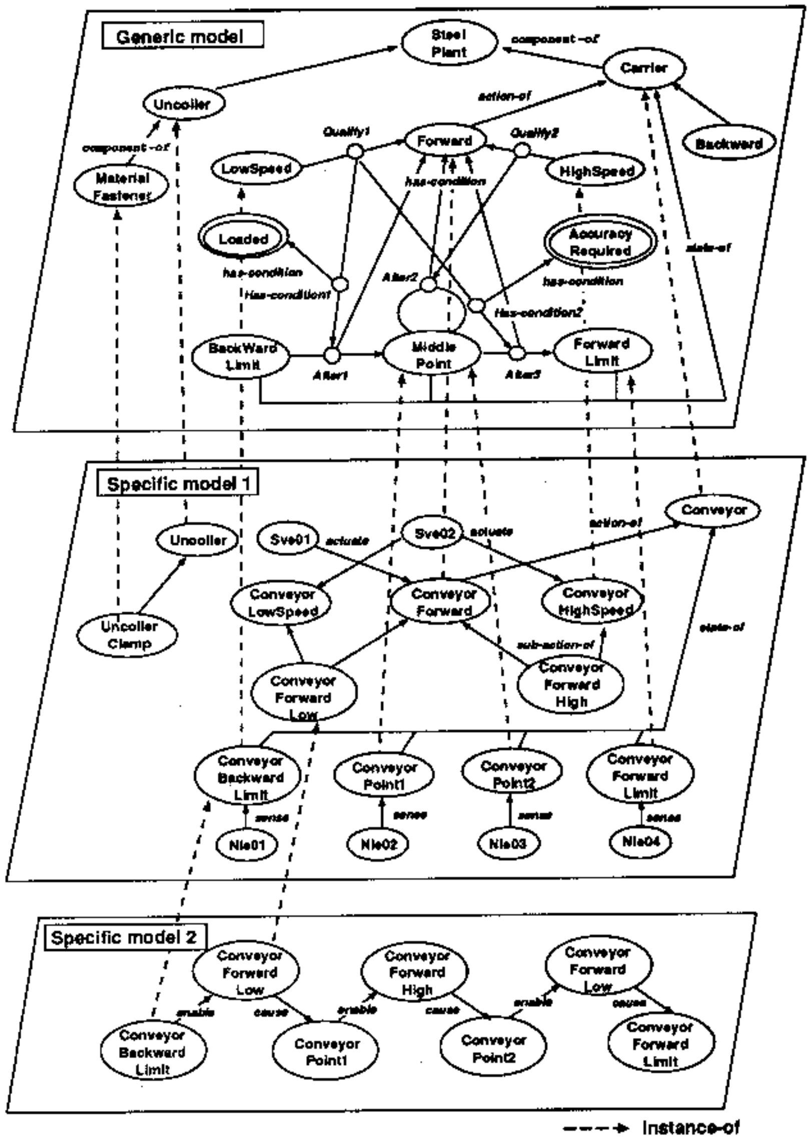


Figure 7. The Plant Model.

In the next step, specific model 1 is transformed into specific model 2 along a high-level control specification, that is, a composite-action–level specification. Detailed machine behaviors, as represented by transitional relations between machine actions and states, are specified and stored in specific model 2. They are further refined to the program model (intermediate representation) using programming knowledge and program parts.

Designers validate specific model 1 with views of a simulation and a detailed specification format. If the transitional relations between machine actions and states are not just as the designers intended, higher-level specifications are modified.

Plant Model

Figure 7 illustrates a portion of the model of a steel plant. The generic model contains general knowledge concerning the class of a plant. SteelPlant is shown as the composition of two machines, Carrier and Uncoiler. Forward is one of several possible actions of Carrier. The relation Qualify specifies the possible control speed for Forward, which can be executed at either LowSpeed or HighSpeed. BackwardLimit, MiddlePoint, and ForwardLimit are possible states of Carrier, with After specifying transitional relations conditioned by Forward. For example, a partial description of the class Forward is as follows:

```
[ Forward
          SUPER:                        MachineAction
          OPPOSITE:                     Backward
          ACTION-OF:                    Carrier
          CONDITION-OF:                 After1, After 2, After 3
          Qualified-by 1:               LowSpeed
          Qualified-by 2:               HighSpeed
          method:                       [...]
] .
```

Qualify, After, and Has-condition are conditional relations. They are defined as a class in terms of domain primitives, and they have conditions and methods for constructing specific models. Qualify1 is one of the instances of the conditional relation Qualify. Qualify and Qualify1 are as follows:

```
[ Qualify
          SUPER:           Relation
          ORIGIN:          MachineAction
          DESTINATION:     MachineAction
          Has-condition:   Relation
          method:          [...]
];
```

```
[ Qualify1
        INSTANCE-OF:    Qualify
        ORIGIN:         LowSpeed
        DESTINATION:    Forward
        Has-condition1: After1
        Has-condition2: After3
] .
```

Qualify1 is related to After1 and After3 by the conditional relations Has-condition1 and Has-condition2. Has-condition1 has the condition Loaded, and if Loaded is true, Has-condition1 is actual. Has-condition2 has the condition AccuracyRequired, and if AccuracyRequired is true, Has-condition2 is actual. Thus, when a carrier is loaded at the beginning of an action, or accuracy is required at the end of an action, it must be driven at low speed. Loaded and AccuracyRequired are condition frames that have methods to infer the actual states of the target plant. Thus, the generic model has general knowledge that is independent of the target plant.

The specific model consists of two consecutive models. The first specific model (Specific model 1 in figure 6) contains concrete descriptions of the target plant structure. After the environment of the target plant is specified, the specific model is constructed and is referred to in all subsequent phases of the software life cycle. The basic structure—for example, the physical construction, control relations, and interlocks—is generated by interpreting machine specifications using a dictionary that contains the basic vocabulary of plant control. The machine Conveyor is an instance of Carrier, and UncoilerClamp is an instance of MaterialFastener. The machine Conveyor has the action ConveyorForward driven by the actuator Sve01. The states ConveyorBackwardLimit, ConveyorPoint1, and so on, are detected by the sensors Nle01, Nle02, and so on. A partial description of ConveyorForward is as follows:

```
[ ConveyorForward
      INSTANCE-OF:        Forward
      ACTION-OF:          Conveyor
      ACTUATED-BY:        Sve01
      HAS-SUB-ACTIONS:    ConveyorForwardLow
                             ConveyorForwardHigh
      START-INTERLOCK:    UncoilerStop
      RUN-INTERLOCK:      (AND ConveyorLowerLimit
                             (OR  (NOT  ConveyorCoil Touch)
                                  (AND  ConveyorCoil Touch
                                        Uncoilerclamp CloseLimit)))
        MUTUAL-INTERLOCK:ConveyorBackward
] .
```

The second specific model (specific model 2 in figure 6) contains a transitional relationship between actions and states of machines in the target plant. Relations between actions and states are constructed by interpreting and refining a control specification using a dictionary and expertise about plant control. ConveyorForwardLow and ConveyorForwardHigh are concrete actions of Conveyor. The relations cause and enable specify the transitional relationship between actions and states of Conveyor. The cause links an action to a state. It specifies that the execution of a specified action results in a specified state. The enable links a state to an action. It specifies that a specified state enables a specified action.

Specification Validation

The symbolic simulation (Fox 1987; Reddy and Fox 1986) enables designers to validate specifications by testing for errors or omissions. The description of the machine action, the machine state, and the transitional relations between them in the specific model represent the detailed machine behavior of the target plant. The system simulates an expected machine behavior by tracing these transitional relations, that is, cause and enable relations.

Stepwise Refinement

The action-level specifications are refined into programs by referring to programming knowledge. The programming knowledge is implemented in an object-oriented style of programming, with objects representing a particular piece of programming knowledge. The programming knowledge for a machine operation sequence is:

```
[ MachineOperation
      SUPER:  ProgrammingKnowledge
      PATTERN:        (BETWEEN
                              StartOrderAcceptance
                              < StopSensor
                              : (AND RunInterlock
                                      MutualInterlock))
                      -> (ON MachineOperation)
] .
```

It has a program pattern that means “in a period between accepting a start order and detecting a stop sensor, provided the interlock conditions hold, output an on signal to the actuator that drives the target machine." The object sends a message to lower-level objects that possess their own programming knowledge (StartOrderAcceptance, StopSensor, RunInterlock, and MutualInterlock) until an intermediate representation is obtained. The intermediate representation of a part of a program is as follows:

```
[ ConveyorForward
  (BETWEEN (AND StartOrder UncoilerStop) ;  StartOrderAcceptance
       < ConveyorForwardLimit              ;  StopSensor
       : (AND (AND ConveyorLowerLimit
             (OR (NOT ConveyorCoilTouch)
                  (AND ConveyorCoilTouch
                       UncoilerclampCloseLimit))) ;  RunInterlock
             (NOT ConveyorBackward)) ;  MutualInterlock
  )
  -> (ON ConveyorForward)
] .
```

Each element is replaced by controller I-O signals, and finally, the fragment is converted to a target LADDER DIAGRAM and SEQUENTIAL FUNCTION CHART, established languages for writing control programs.

Part-Retrieval Method

Program parts are retrieved by keys that consist of the operation device type, the machine type, the actuator type, and the sensor type. The retrieval function is implemented by the production system, which uses rules in the programming knowledge base. Retrieved parts are customized in accordance with the combination of operation devices and the number of actuators.

Program parts are designed to be as small as possible, basically so that they can be widely applicable. Furthermore, macrodescriptions are provided in the program parts to enhance their flexibility.

Programmable controller languages usually use static storage allocation, and most of their variables are global. Variables in different retrieved program parts are required to be appropriately identified. This automatic programming system attaches attributes, such as machine names and operation names, to each newly created variable for maintaining identity.

Discussion

CAD-PC/AI has been in practical use since October 1990 in the sequence control program design divisions in the Toshiba Corporation. Programmable controllers are being applied to a wider range of work, and their functions are being upgraded and diversified. Thus, a design support system was strongly required in these divisions. During the first stage of development, we decided that the design processes using CAD-PC/AI should be close to the conventional ones. The sequence control program design process was considerably analyzed, and the life cycle discussed previously was established. We then decided what activities in the life cycle could be supported by AI technology. This policy was one reason that the system was deployed smoothly.

We used CAD-PC/AI to generate sequence control programs for steel plants as follows:

Wire and rod mill plant	2.5K program steps
Continuous pickling line	6.5K program steps
Continuous galvanization line	90K program steps
Continuous galvanization line	15K program steps

The first case was for validating the CAD-PC/AI prototype. The quality of generated programs was compared with those designed manually. Some problems were found with the knowledge bases, the lack of program parts, and the inconvenient human interface. After these problems were altered, three practical jobs were implemented using CAD-PC/AI. The generated programs are now running in a real plant control situation in Japan. For example, the third case breaks down as follows:

System size	
Number of frames	2900 frames
Number of program parts	190 parts
Number of part-retrieval rules	320 rules
Specification	
Number of records (machine specifications)	17K records
Number of steps (control specifications)	5.5K steps
Target program	
Target plant	Continuous galvanization line
Programmable controller	PCS-5000 (4 sets)
Program size	90K steps

It would have taken about 100 person-months to complete the target program using the conventional technique. The total cost for software development, including specifications and testing, was reduced by half using this system. The generated program was checked by both design experts and a plant simulator. The achieved quality was satisfactory. The reasons for these advantages are as follows:

First, the plant model enables designers to easily describe machine actions and states for specifying control programs.

Second, the plant model supports specification validation by explaining the expected machine behavior represented in the specific model, helping the designers notice mistakes in earlier design stages.

Third, maintenance activity much more closely parallels the original development. In this domain, plant operations are sometimes changed, which, in turn, affects the control program specifications. When machine specifications are altered, the specific model is constructed again. When control specifications are altered, the resulting programs are regenerated by replaying the development process. Thus, maintenance is performed by altering the specifications and repeating the original development process, not by patching programs.

The generic model represents general knowledge about a class of plants, and it can be used for several different applications. A single generic model was shared between the last three applications. This applicability is important for widespread use of the system.

CAD-PC/AI doubled design productivity. It took about 20 person-years to develop CAD-PC/AI: 3 person-years by the experts, 10 person-years by the system engineers, and 7 person-years by researchers. At the first stage of the development, three researchers were apprenticed to a design division for a few months to learn the design skill by themselves. It helped these researchers to communicate with the experts throughout CAD-PC/AI research and development.

The system made the quality of programs generated by the experts and others relatively uniform. However, it cannot be said that a systematic accumulation of design knowledge was accomplished because only the original developers can maintain the knowledge bases consistently. Maintenance has been continued by the original developers (researchers and system engineers) in accordance with the designers' requirements. Enabling designers to easily extend knowledge bases is the basis for further work.

References

Araya, A., and Mittal, S. 1987. Compiling Design Plans from Descriptions of Artifacts and Problem-Solving Heuristics. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 552–558. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Barstow, D. R. 1985. Domain-Specific Automatic Programming. *IEEE Transactions on Software Engineering* SE-11(11): 1321–1336.

Brown, D. C., and Sloan, W. N. 1987. Compilation of Design Knowledge for Routine Design Expert Systems: An Initial View. In Proceedings of the ASME International Computers in Engineering Conference, 131–136. Fairfield, N.J.: American Society of Mechanical Engineers.

Chandrasekaran, B., and Mittal, S. 1983. Deep Versus Compiled Knowledge Approaches to Diagnostic Problem Solving. *International Journal of Man-Machine Studies* 19:425–436.

Darington, J. 1981. An Experimental Program Transformation and Synthesis System. *Artificial Intelligence* 16:1–46.

Fickas, S. F. 1985. Automating the Transformational Development of Software. *IEEE Transactions on Software Engineering* SE-11(11): 1268–1277.

Fox, M. S. 1987. Constraint-Directed Search: A Case Study of Job-Shop Scheduling. San Mateo, Calif.: Morgan Kaufmann.

Gero, J. S. 1990. Design Prototypes: A Knowledge Representation Schema for Design. AI Magazine 11(4): 26–36.

Green, C., and Westfold, S. J. 1982. Knowledge-Based Programming Self-Applied. *Machine Intelligence* 10.

Keller, R. M.; Baudin, C.; Iwasaki, Y.; Nayak, P.; and Tanaka, K. 1989. Compiling Special-Purpose Rules from General-Purpose Device Models, Technical Report, KSL-89-49, Knowledge Systems Laboratory, Dept. of Computer Science, Stanford Univ.

Lubars, M. D., and Harandi, M. T. 1987. Knowledge-Based Software Design Using Design Schemas. In Proceedings of the International Conference on Software Engineering, 253—262. Los Alamitos, Calif.: IEEE Computer Society.

Manna, Z., and Waldinger, R. 1980. A Deductive Approach to Program Synthesis. *ACM Transactions on Programming Languages and Systems* 2(1): 90–121.

Mizutani, H.; Nakayama, Y.; Sadashige, K.; and Matsudaira, T. 1991. A Knowledge Representation for Model-Based High-Level Specification. In Proceedings of the IEEE Conference on Artificial Intelligence Applications, 124–128. Los Alamitos, Calif.: IEEE Computer Society.

Nakayama, Y.; Mizutani, H.; Sadashige, K.; and Matsudaira, T. 1990.

Model-Based Automatic Programming for Plant Control. In Proceedings of the IEEE Conference on Artificial Intelligence Applications, 281–287. Los Alamitos, Calif.: IEEE Computer Society.

Neighbors, J. M. 1984. The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering* SE-10(5): 564–574.

Ono, Y.; Tanimoto, I.; Matsudaira, T.; and Takeuchi, Y. 1988. Artificial Intelligence–Based Programmable Controller Software Designing. In IEEE AI'88 Proceedings of the International Workshop on AI for Industrial Applications, 85–90. Los Alamitos, Calif.: IEEE Computer Society.

Ramana-Reddy, Y. V., and Fox, M. S. 1986. The Knowledge-Based Simulation System. *IEEE Software* 3(2): 26–37.

Smith, D. R.; Kotik, G. B.; and Westfold, S. J. 1985. Research on Knowledge-Based Software Environments at Kestrel Institute. *IEEE Transactions on Software Engineering* SE-11(11): 1278–1295.