# Model-based Automatic Generation of Sequence Control Programs from Design Information

**T. Sakao and Y. Umeda and T. Tomiyama**
Graduate School of Engineering, the University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan
Email: {sakao, umeda, tomiyama}@zzz.pe.u-tokyo.ac.jp

**Y. Shimomura**
Mita Industrial Co., Ltd.
Tamatsukuri 1-2-28, Chuo-ku, Osaka 540, Japan
Email: simomura@mita.co.jp

## Abstract

This paper proposes a new model-based technique to automatically generate sequence control programs for mechatronics machines from design information. This technique solves one of the bottlenecks in developing mechatronics machines by reducing burdens of software development. In this technique, a sequence control program is generated from a model of the design object represented in a mechanical CAD by searching the physical causalities described by Qualitative Process Theory and by providing geometric information. Based on this technique, we implemented a prototype system named the Sequence Control Program (SCP) Generator which integrates a mechanical CAD and a software generation system. An example of control program generation for a photocopier with the SCP Generator is described.

## Introduction

Design of mechatronics machines involves both that of hardware and software. It is well known that one of the bottle necks in developing mechatronics machines is software development, even if the control is based on simple sequence control. Thus, a system to automate this process is required.

Although many researchers have studied methods to automatically generate control programs, most of them are based on transformation knowledge from formal specifications to codes (*e.g.* (Fickas 1985)). For instance, Nakayama (Nakayama 1990) has developed a technique to automatically generate sequence control programs based on a model of object from the requirements for the object represented in the conceptual level of the designer. Graves (Graves 1992) also succeeded in automatic programming through generation of actions which should be activated based on functional connections among the components of a plant.

These techniques have the following problems in common:

1. Specifications for the control and those for the mechanism are often developed independently and inconsistently.
2. The designer is not able to make good use of information about the mechanism in the software design stage.

In other words, the scope of previous studies is limited to design of software and such techniques are not fully integrated with a mechanical CAD, thus calling for concurrent engineering problems. That is why these techniques are not useful for supporting the whole process of the design of mechatronics machines effectively.

This paper proposes a new model-based technique to automatically generate sequence control programs from models of a design object in the "Knowledge Intensive Engineering Framework" (KIEF) (Tomiyama 1994). Within KIEF, the conceptual design stage is supported by the "Function-Behavior-State (FBS) Modeler" (Umeda *et al.* 1990) dealing with a model of a design object that preserves designer's intentions in the form of functions and physical causalities based on Qualitative Process Theory (QPT) (Forbus 1984). This method integrating a mechanical CAD and a software generation system solves the problems described above. Moreover, this method is applicable to situations in which sequence control programs must be dynamically changed; consider a truly flexible manufacturing system that must be flexibly controlled or a function redundant self-maintenance machine that dynamically changes its behaviors to maintain its functions (Umeda, Tomiyama, & Yoshikawa 1992; Umeda *et al.* 1994).

In chapter 2, KIEF is is briefly illustrated and our strategy for generating sequence control programs is described. Chapter 3 describes the prototype system named a Sequence Control Program (SCP) Generator and its algorithm. Chapter 4 illustrates an example of generation of a sequence control program for a photocopier with the SCP Generator. It also demonstrates that the SCP Generator can generate sequence control programs for different operation conditions from a

normal one. This is useful for dynamic software generation during operation. Chapter 5 and 6 describe discussions and conclusions respectively.

## Knowledge Intensive Engineering Framework

### Architecture

Our group at the University of Tokyo has been conducting research on the development of KIEF. KIEF is an integrated computational framework for engineering activities in various product life cycle stages, including design, production, operation, maintenance, and recycling (see Figure 1). Using various modelers, simulators, and a very large-scaled knowledge base (VLKB) in a flexible manner allows an engineer to create more added-value.

Within KIEF, the designer conducts conceptual design with the FBS Modeler to be described in Section . The FBS Modeler decomposes functional design specifications into physical behaviors of mechanisms. The SCP Generator, whose algorithm is described in Chapter 3, takes this qualitative behavioral information of the mechanisms of a design object and generates qualitative control sequence. The Qualitative Process Abduction System (QPAS) (Ishii, Tomiyama, & Yoshikawa 1993) reasons about appropriate physical phenomena which realize the required qualitative changes of the parameters based on physical causalities managed by the Qualitative Reasoning (QR) System (Kiriyama, Tomiyama, & Yoshikawa 1991). The SCP Generator further incorporates quantitative geometric information of the mechanisms and generates a sequence control program. The 2-D bit map modeler is a tool for handling geometric information of the design object and the Spatial Reasoning (SR) System (Matsumoto *et al.* 1993) reasons about kinematic properties of the mechanisms.

All of these systems are integrated in KIEF that allows flexible use of knowledge about design objects (see Figure 1) and enable the following:

1. The system can generate sequence control programs from the results of conceptual design.
2. This results in easy modifications of sequence control programs, contributing to increasing productivity of software development.

### The Function-Behavior-State Modeler

The FBS Modeler deals with *functions*, *behaviors*, and *states* occurring on mechanism. Behaviors and states are managed based on physical causalities that are represented and reasoned by the QR System, which implements QPT. It is important that the FBS Modeler can represent and reason about the trichotomy of function, behavior, and state, because control is a means to obtain the desired function through physical behaviors of the mechanism. Figure 12 shows an example of screen hardcopy of the FBS Modeler.

In QPT, a physical world is modeled with three components; *i.e.*, individuals, individual views, and processes. An individual represents an entity, such as a gear and water. An individual view represents a way of seeing an entity, such as seeing a gear as a rotatable entity. A process represents a physical phenomenon, such as rotation of a gear pair and boiling of water. In the QR System, individuals, individual views, and processes are represented in a uniform framework named a *view*.

Table 1 shows the scheme of function prototypes used to model functions in the FBS Modeler. *Decomposition* describes feasible candidates for detailing this function in the form of networks of subfunctions which include representation of needed temporal transitions among subfunctions. *F-B relationship* describes behaviors that can perform this function in the form of networks of *views* in the QR System.

Table 1: Definition of a Function Prototype

| Item | Contents |
|---|---|
| Name | *verb + objectives* |
| Decomposition | subfunction networks |
| F-B Relationships | a network of views |

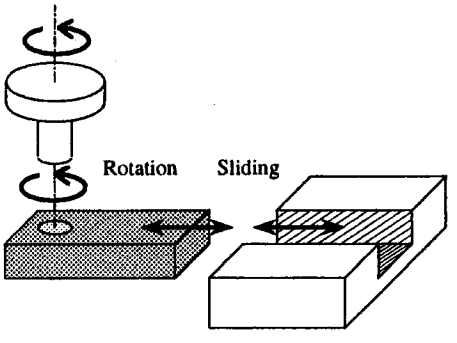



Figure 2: Examples of the Kinematic Pairs

### Managing Geometric Information

Geometric reasoning and kinematic reasoning symbolically represent and reason about mechanisms (*e.g.* (Faltings 1987; Joskowicz & Sacks 1991; Gupta & Struss 1995)). KIEF deals with geometric information about mechanisms using a geometric modeler plugged into the Metamodel System (Kiriyama, Tomiyama, & Yoshikawa 1991) through the Pluggable Mechanism (Yoshioka *et al.* 1993) and the SR System can reason about the spatial characteristics of the design objects such as kinematic pairs. Figure 2 shows simple examples of the kinematic pairs. The Metamodel System maintains consistency among different design object models (such as qualitative model and geomet-

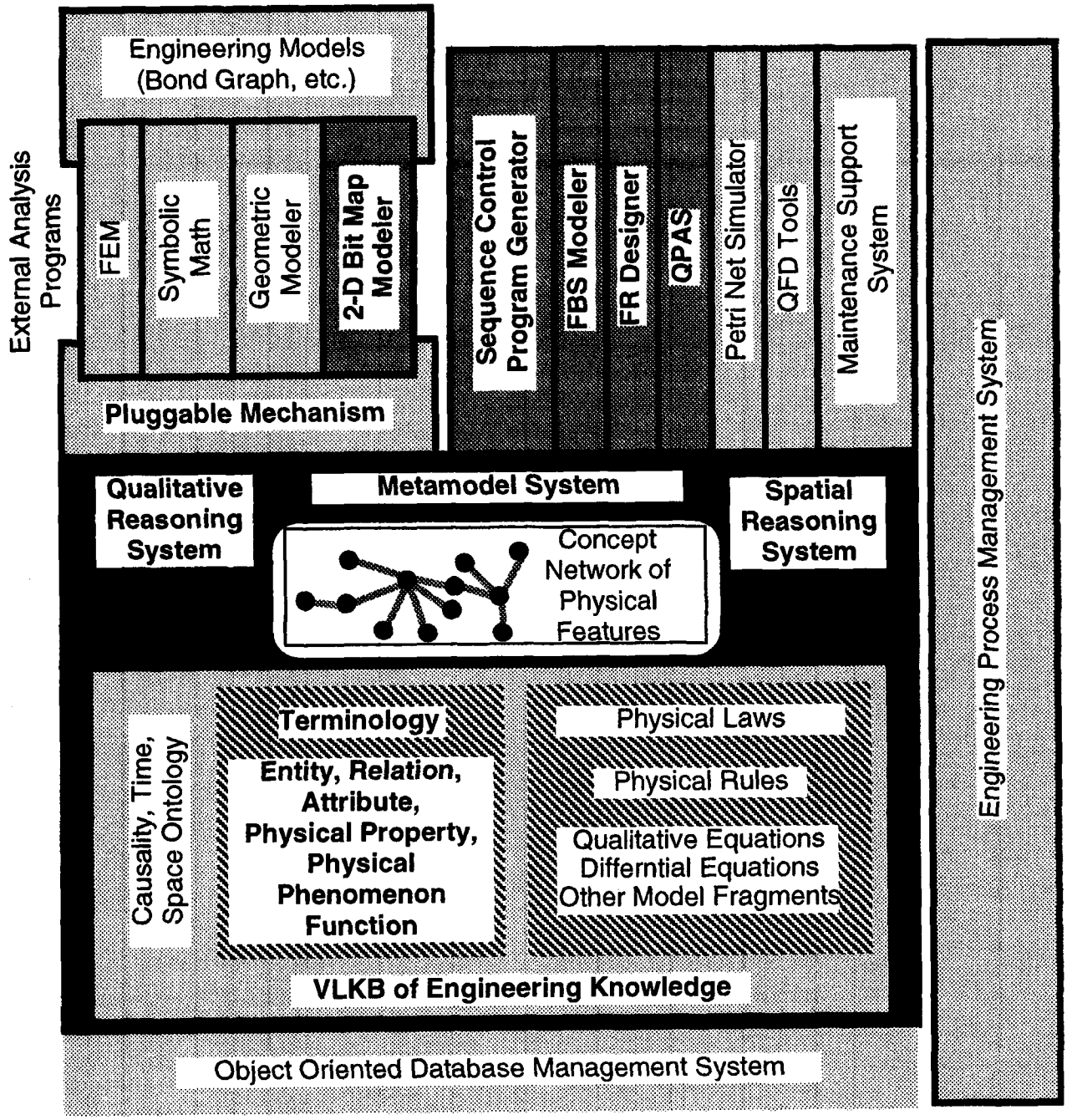



Figure 1: Knowledge Intensive Engineering Framework

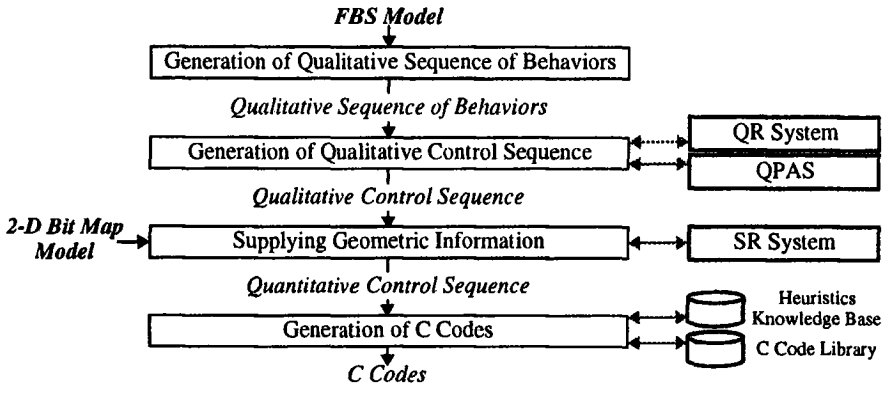



Figure 3: Algorithm of the Sequence Control Programs Generation

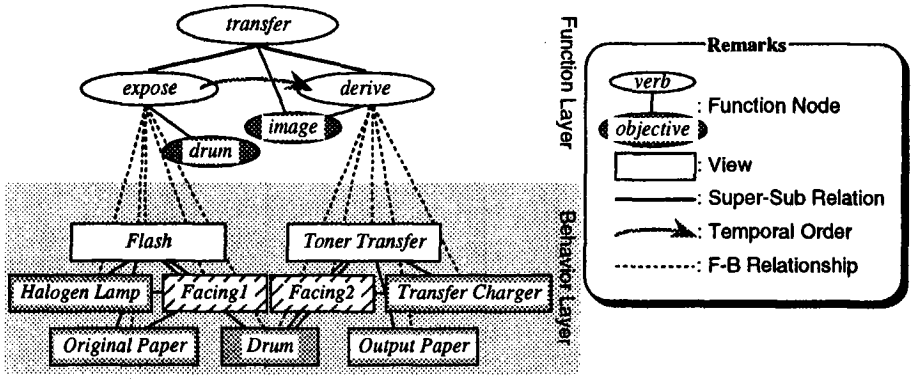



Figure 4: The Input FBS Model

ric model) by symbolically representing relationships among them. In this study, a 2-D bit map modeler handles geometric information about design objects. However, alternatively a solid modeling system can be used, if geometry of the design object has enough details.

The designer makes the correspondence between the geometric information of the design object represented on the 2-D bit map modeler and the symbolic information of the physical states on the FBS Modeler, and the Metamodel System manages these correspondences.

This architecture allows the SCP Generator to access to geometric information for generating quantitative control sequences.

## The Algorithm of the Sequence Control Program Generation

After conceptual design and parametric design are finished, the SCP Generator generates a sequence control program that satisfies the designer's intentions represented as a sequence of needed functions on the FBS model. Figure 3 shows the algorithm.

In Figure 3, a 2-D bit map model represents the structure of the design object with its quantitative data and the Heuristics Knowledge Base contains the heuristics for controlling devices. A *qualitative* control sequence means a sequence of states in temporal order, while a *quantitative* one is a sequence that includes time information. The quantitative control sequence is converted to a C program using respective C functions stored in the C Code Library. Although we used the C language, our technique can generate programs in any other language by modifying this library.

In this chapter, we describe the algorithm to generate sequence control programs with the SCP Generator using an example FBS model depicted in Figure 4. Sequence control is a way to control according to the conditions and the sequence prepared in advance. Since this example satisfies those, the algorithm described here does not lose its generality. Note that we do not deal with quantitative control, nor feedback control here.

In the FBS model of Figure 4, oval nodes in the upper half denote functions and represent as a whole the functional hierarchy while rectangular nodes in the lower half denote views and represent as a whole be-

haviors and states of the design object.

## Generation of Qualitative Sequence of Behaviors

The SCP Generator derives a sequence of behaviors from an FBS model that represents views to satisfy a transition sequence of subfunctions in request.

In the example of Figure 4, the top function is *transfer image* that can be decomposed into *expose drum* and *derive image*. The arrow from *expose* to *derive* indicates their temporal order; *i.e.*, first *expose drum* should happen and next *derive image*. The example FBS model also shows their F-B Relationships (see Table 1). Namely, *expose drum* and *derive image* are performed by activating *Flash* and *Toner Transfer*, respectively. From this, a qualitative sequence of behaviors (see Table 2) is derived from the initial FBS model as a temporal sequence of *state1* and *state2*.

Table 2: Qualitative Sequence of Behavior

| state | *state1* | *state2* |
|---|---|---|
| required function | *expose drum* | *derive image* |
| views | *Flash*<br>*Halogen Lamp*<br>*Original Paper*<br>*Drum*<br>*Facing1* | *Toner Transfer*<br>*Transfer Charger*<br>*Output Paper*<br>*Drum*<br>*Facing2* |

## Generation of Qualitative Control Sequence

In this stage, the QR System and QPAS reason out required parametric changes from the qualitative sequence of behaviors generated in the previous stage. The QR System performs envisioning to see whether or not the input model has a possibility to arrive at the required states under feasible conditions regarding parameters, entities, and relations.

To do so, first, the designer specifies the initial state and the final state by giving the values of the parameters. The system creates a parameter transition map for activating views needed for the sequence of behaviors and completes a consistent transition map based on the procedure below. For this example, the designer sets the values of the parameters included in the design object of *state1* and *state2* at the initial state and at the final state as follows:

initial state: *Surface = charged, Angle = x0, Position = y0, Velocity1 = 0, Velocity2 = 0, Lamp = off, Charger = off, Developer = off, Motor1 = off, Motor2 = off,*

final state: *Surface = nothing, Angle = x3, Position = y2, Velocity1 = 0, Velocity2 = 0, Lamp = off, Charger = off, Developer = off, Motor1 = off, Motor2 = off,*

where *Surface, Angle,* and *Velocity1* belong to *Drum*, and *Position* and *Velocity2* to *Output Paper*. Also, *Surface* is a non-controllable, sensory state parameter, *Lamp, Charger, Developer, Motor1, and Motor2* are controllable state parameters, and *Angle, Position, Velocity1, and Velocity2* are variable parameters. Controllable state parameters are associated with controlling methods described in the Heuristics Knowledge Base.

1. Based on the knowledge shown in Table 3 maintained by the QR System, a specification transition map (see Figure 5) is created by providing information about the initial state and the final state. Also, new states, *state1* and *state2*, at which *Flash* and *Toner Transfer* are supposed to occur, are recognized. Table 3 denotes if the conditions of a physical phenomenon are satisfied, its influences occur on the design object. Note that at this stage, we only focus on uncontrollable parameters in conditions and influences of phenomena (see Figure 5). In Figures 5, 6 and 8, *ch*, *ex*, *dv* and *no* are abbreviations for *charged, exposed, developed* and *nothing.*

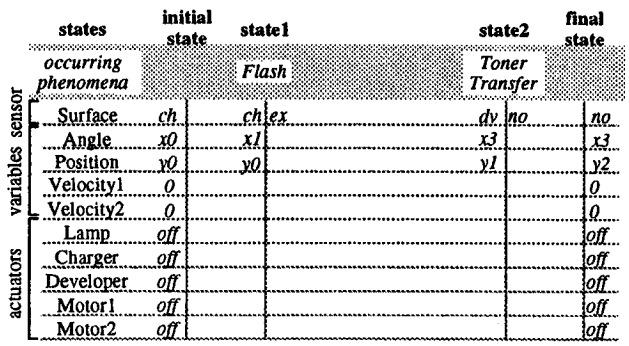



Figure 5: Specified Transition Map

2. QPAS reasons out the physical phenomena that activate the changes of the uncontrollable parameters reasoned out in the first step, because uncontrollable parameters should be controlled indirectly by activating an appropriate physical phenomenon. In the example, QPAS reasons out that *Develop, Drum Rotation,* and *Paper Moving* should be activated, between *state1* and *state2*, at the initial state, and at *state1*, respectively, by considering parametric changes. The system also adds a state named *state 3* between *state1* and *state2* to get *Develop* that changes the value of parameter *Surface* from *exposed* to *developed* (see Figure 6).
3. The QR System completes the conditions of controllable parameters for all the physical phenomena reasoned out. In the example, the transition of the values of five actuators is generated to activate those five phenomena (see Figure 6).

Table 3: Definition of Each Physical Phenomenon in the QR System

| phenomena | *Flash* | *TonerTransfer* | *Develop* | *DrumRotation* | *PaperMoving* |
|---|---|---|---|---|---|
| conditions of the parameters in the views | *Angle=x1*<br>*Position=y0*<br>*Surface=charged*<br>*Lamp=on* | *Angle=x3*<br>*Position=y1*<br>*Surface=developed*<br>*Charger=on* | *Angle=x2*<br>*Surface=exposed*<br>*Developer=on* | *Motor1=on* | *Motor2=on* |
| influences | *Surface=exposed* | *Surface=nothing* | *Surface=developed* | *Velocity1=v0*<br>$\frac{dX}{dt} = Velocity1$ | *Velocity2=w0*<br>$\frac{dY}{dt} = Velocity2$ |

After these steps, the consistent transition map that represents a qualitative control sequence shown in Figure 6 is generated.

However, this algorithm has some problems and limitations. First, there is no guarantee that qualitative control sequences as solutions exist nor they are optimal. When there are multiple solutions, the designer has to choose one. Second, this algorithm cannot deal with situations of inter-dependent phenomena. For instance, phenomenon $P_1$ assumes parameter $X$ be *high*, and $P_2$ assumes $Y$ be *low*. If $P_1$ has an influence that $Y$ becomes *low* and $P_2$ has that $X$ becomes *high*, and if there is no other phenomena that involve $X$ or $Y$, $P_1$ and $P_2$ are inter-dependent phenomena.

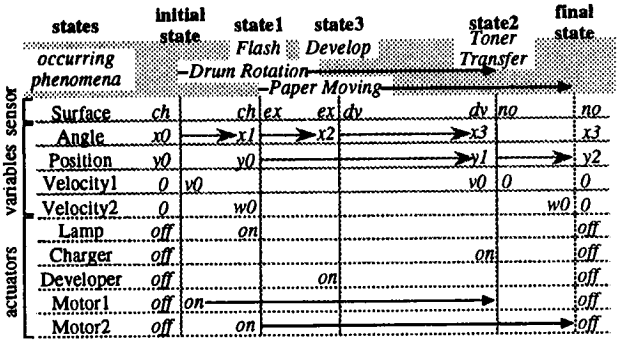



Figure 6: Transition Map with Qualitative Control Sequence

## Supplying Geometric Information

In this stage, a quantitative control sequence is generated from the generated qualitative control sequence. This is performed by supplying time information between states in the qualitative control sequence and sensory information needed for the control. These two types of information are created from geometric information obtainable through the Metamodel System.

For the example, the designer specifies geometric information of the mechanism by the 2-D bit map model shown in Figure 7. The following describes the algorithm we developed to supply geometric information

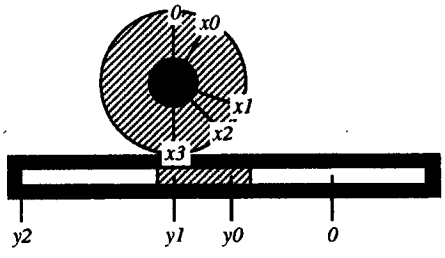



Figure 7: The Input Bit Map Model

to qualitative control sequences.

1. *Giving Correspondences between Geometric Objects in the Bit Map Model and Entities in the Physical Model.*

   The designer gives correspondences between geometric objects in the 2-D bit map model and entities in the physical model maintained by the QR System. For the example, the designer makes correspondences between the round object shown in the bit map model and *Drum* in the FBS model of Figure 4, and the rectangular object and *Output Paper*.

2. *Reasoning about Kinematic Pairs.*

   Motion takes place at kinematic pairs. To clearly recognize kinematic causality for motion control, the SR System finds out kinematic pairs in the design object from its 2-D bit map model and its physical model maintained by the QR System. This can be done by comparing the necessary spatial conditions for each kinematic pair in the kinematic pair data base with the spatial conditions represented by the physical model such as *fixed*. In the example, a rotational pair and a sliding one are found.

3. *Giving Correspondences between Parameters Associated with Kinematic Pairs and Parameters in the Physical Model.*

   The SR System also sets a parameter associated with each kinematic pair. The designer gives its correspondence to a parameter of the physical model. In the example, the designer gives correspondences between *angle* set for the rotational pair and parameter

*Angle* of *Drum*, and *position* set for the sliding pair and *Position* of *Output Paper*.

4. *Supplying Time Information.*

   Since variable parameters change their values with respect to time according to differential equations of the parameters, time intervals can be calculated by integrating the equations assuming that the variable parameters are controlled precisely. Namely, if $\alpha$ and $\beta$ are related by equation $d\alpha/dt = \beta$ and $\beta$ is constant between two states, the length of time $\Delta t$ between the two states is calculated by equation $\Delta t = \Delta\alpha/\beta$. The QR System searches this relationship and the length of time between two states is calculated using the geometric quantitative data from the bit map model.

   In case that plural variables change during one interval, another state has to be added between those two states, because each transition time is not necessarily equivalent. Here, the heuristics that those variables should simultaneously start changing is used.

   In Figure 8, time information is added to the qualitative control sequence. Each time period is calculated as follows:

   $$t1 = (x1 - x0)/v0,$$
   $$t2 = (x2 - x1)/v0,$$
   $$t3 = (x3 - x2)/v0,$$
   $$t4 = (y1 - y0)/w0 - t2 - t3, and$$
   $$t5 = (y2 - y1)/w0.$$

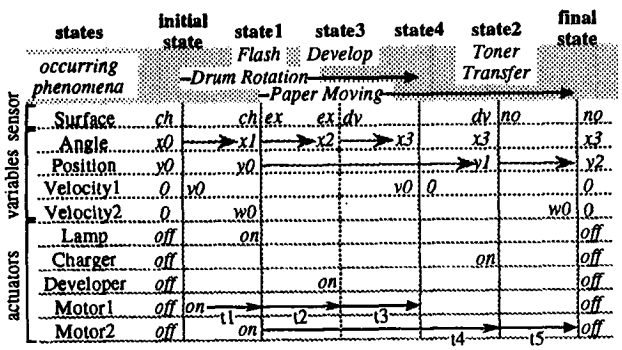



Figure 8: Modified Transition Map Including Time Information

5. *Supplying Sensory Information.*

   Some variable parameters are difficult to control precisely only based on time information. In order to control those parameters, sensory information should be added. It is determined by the way of control represented on the physical model and the kind of a kinematic pair whether the control of a parameter with a kinematic pair is achieved precisely or not. As to kinematic pairs for which precise control is difficult, the condition of such a variable parameter is exchanged to that of a sensor value if there is a respective sensor[1].

   In the example, sensory information is added to control parameter *Position*, by exchanging the conditions of *Position* = $y1$ and *Position* = $y2$ to *Sensor1* = *on* and *Sensor2* = *on*, respectively, where we make the following assumption:

   A super-class phenomenon of *PaperMoving* is to slide an object by friction[2], and therefore, parameter *Position* on the sliding pair is difficult to control precisely. On the other hand, a super-class phenomenon of *DrumRotation* is to rotate an object by gear transmission, and therefore, parameter *Angle* on the rotational pair is controlled precisely.

   Then, the quantitative control sequence is generated. It is in the form of the "if-then" rule temporally ordered. Here, "charged(Surface)" means that the value of *Surface* is *charged*. "make(Lamp,on)" means to set the actuator value *Lamp* to "on." The "if" part specifies conditions for activating the rules. The "then" part designates the "after $t$ [ms]" timing information and actions the rule has to fire. Some rules can specify waiting conditions in the "*SensorX : vol1 → vol2, action*" format, which means that *action* should be taken waiting for *SensorX* becoming *vol2* from *vol1*.

   From the example above, the following quantitative control sequence is generated.

   1. **if** charged(Surface) & off(Lamp) & off(Charger) & off(Developer) & off(Motor1) & off(Motor2) **then after** 0 [ms], make(Motor1,on)
   2. **if** charged(Surface) & off(Lamp) & off(Charger) & off(Developer) & on(Motor1) & off(Motor2) **then after** t1 [ms], make(Lamp,on), make(Motor2,on)
   3. **if** exposed(Surface) & off(Lamp) & off(Charger) & off(Developer) & on(Motor1) & on(Motor2) **then after** t2 [ms], make(Developer,on)
   4. **if** developed(Surface) & off(Lamp) & off(Charger) & off(Developer) & on(Motor1) & on(Motor2) **then after** t3 [ms], make(Motor1,off)
   5. **if** developed(Surface) & off(Lamp) & off(Charger) & off(Developer) & on(Motor1) & on(Motor2) **then after** *Sensor1*: off → on, make(Charger,on)
   6. **if** nothing(Surface) & off(Lamp) & off(Charger) & off(Developer) & off(Motor1) & on(Motor2) **then after** *Sensor2*: off → on, make(Motor2,off)

[1] This information is included in the Heuristics Knowledge Base.

[2] This kind of hierarchy is described in VLKB (see Figure 1).

## Generation of C Codes

Finally, the quantitative control sequence is translated into a C program. Each rule in the quantitative control sequence is converted to a sentence in C by translating conditions and actions into respective C functions stored in the C code library. The generated C program which consists of translated functions above and a main function executing them sequentially is compiled to an executable program. Figure 9 shows part of the generated C codes corresponding to the fourth rule and the fifth in the quantitative control sequence above.

```
if(count[4][1]&&countertimepassed(4))
{
  makeoff(C4);
  setcounterinactive(4);
};

if(count[5][1]&&sensorchanged(5))
{
  makeon(C2);
  sensorinactive(5);
};
```

Figure 9: Part of the Generated C Codes

# Applications

We implemented the SCP Generator and developed an experimental photocopier controlled by a personal computer with generated sequence control programs (see Figure 10). Figure 11 depicts the structure of the photocopier.

## Sequence Control Program Design

In this section, an example of sequence control program design on the SCP Generator is described. Figure 12 and Figure 13 show the FBS model and the bit map model of the photocopier, respectively. From these models, the SCP Generator generates a sequence control program.

An output image in A4 size by the generated program is Figure 14 (b), while one with the embedded program in ROM of the photocopier is (a). Although the SCP Generator succeeded in automatic generation of a sequence control program, Figure 14 shows the top positions of the outputs (a) and (b) are slightly different.

The SCP Generator considers slipping of paper and the lag time between the start of control of an actuator and its actual start, which are not included in the object model, by including sensory information. However, it still needs minor adjustments to incorporate empirical know-hows in controlling real actuators, which are considered in the embedded program. This is the major reason for the difference of the top positions in Figure 14 (a) and (b). The embedded program also has some routines to handle errors, such as paper jamming, which cannot be reasoned out from the physical knowledge discussed in this paper.

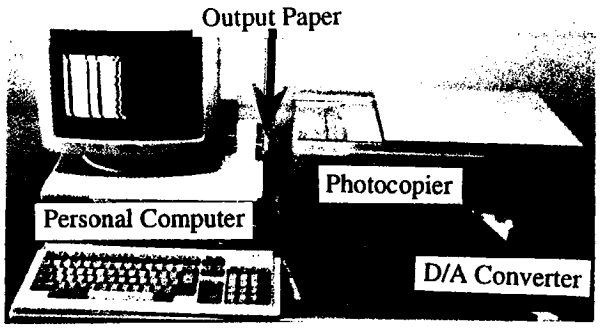



Figure 10: Overview of the Experimental Photocopier

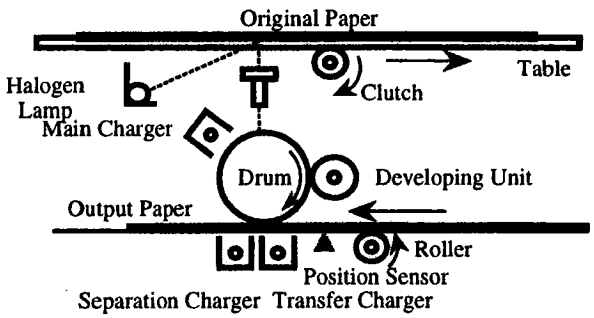



Figure 11: Structure of the Photocopier

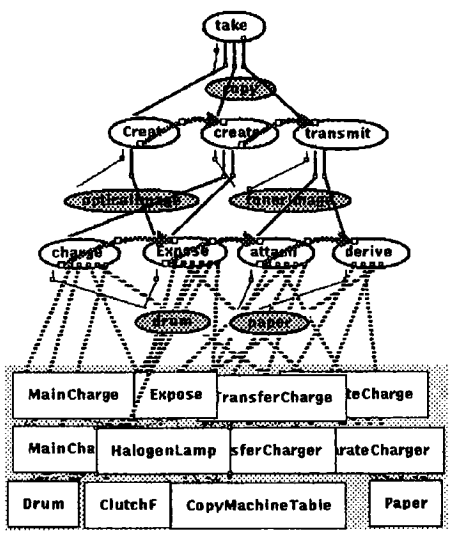



Figure 12: The FBS Model of the Experimental Photocopier

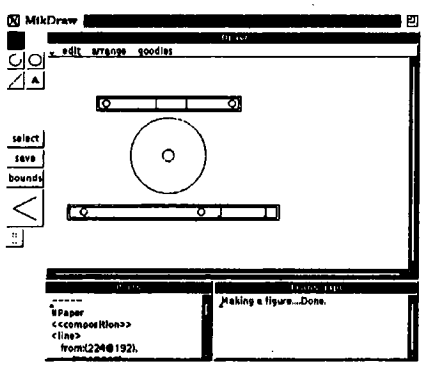
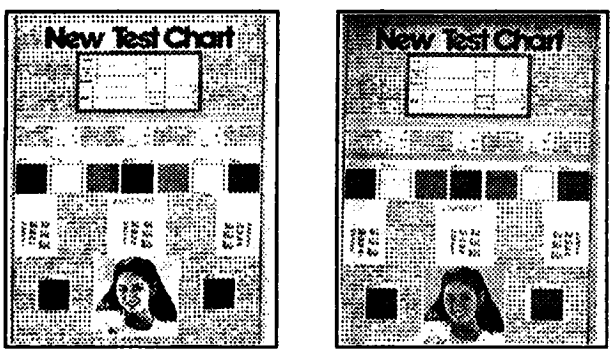



Figure 13: The Bit Map Model of the Experimental Photocopier




(a) The Original Program

(b) The Program Generated with the SCP Generator

Figure 14: Outputs

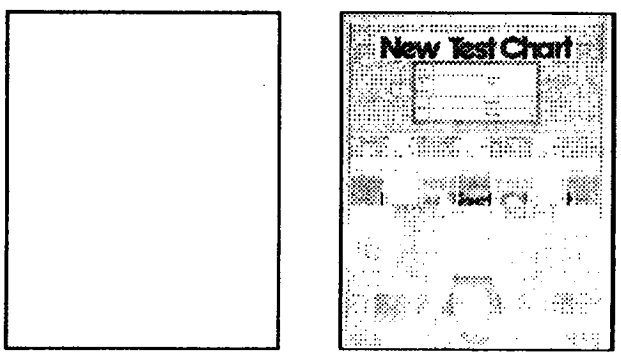
## Dynamic Generation and Modification of Sequence Control Programs

The technique proposed in this paper is also applicable to dynamic generation and modification of sequence control programs during operation, if computing speed is fast enough. This is useful for, *e.g.*, flexible manufacturing systems of which control programs must be dynamically generated or modified in tomorrow's responsive manufacturing environment. It is also useful for a function redundant self-maintenance machine (Umeda, Tomiyama, & Yoshikawa 1992; Umeda *et al.* 1994) that reconfigures its behaviors to maintain its functions even when a critical fault occurs. Since this reconfiguration is performed by rearranging control software, its dynamic generation and modification is extremely critical. This section demonstrates an example of dynamic generation of sequence control programs for function redundant self-maintenance machines.

To conduct function redundant design during the conceptual design stage, we developed a tool called an FR Designer (Umeda, Tomiyama, & Yoshikawa 1992; Umeda *et al.* 1994). This system allows the designer to find out alternative behaviors that can still maintain a target function even when some parts are malfunctioning or losing their function, by using other parts in a slightly different way from the original. The SCP Generator can automatically generate sequence control programs from an FBS model that represents such situations.

Figure 15 (a) shows an output image with the original program before the maintenance (*i.e.*, faulty image) and one with the generated program (b). We can point out the following:

1. The SCP Generator succeeded in functional maintenance based on function redundancy by dynamically generating a sequence control program.
2. The lower half of (b) is basically faulty, not because of self-maintenance, but because the radius of the drum is too small to print out a full A4 image in a function redundant mode that requires the drum to rotate twice. It is peculiar to this experimental photocopier.
3. It took more time to take one copy in (b) than in (a), because the drum has to be rotated twice in (b).




(a) Faulty

(b) The Program Generated Dynamically

Figure 15: Outputs in Faulty States

## Discussions

It is possible to automatically generate sequence control programs from design information. The program generation method with the SCP Generator signifies that integrating various kinds of design knowledge can create more added-value such as automatic software generation. This method, therefore, advocates *knowledge intensive software design.*

This technique allows dynamic generation and modification of sequence control programs, so that the machine can adjust itself to environmental changes, faults, and changes of the user's requirements. This is also

applicable to frequent changes of sequence control programs for manufacturing systems of, particularly, small volume or even one-off production, because the SCP Generator is applicable not only to a photocopier but also to mechatronics machines in general by replacing the knowledge base. We call such machines *soft machines* in that they can flexibly reconfigure and self-organize themselves (Tomiyama *et al.* 1995).

The FBS Modeler represents a function in the form of "*verb + objectives.*" Therefore, other designers are able to easily understand and modify an FBS Model, which results in easy modifications of sequence control programs.

As described before, the control methods are limited to on-off control. Namely, neither methods to control quantity directly nor feedback control can be handled by the algorithm proposed in this paper. The controls either change the values of discontinuous state parameters or make the differential values of continuous variable parameters plus or minus. That physical knowledge on control is managed by the QR System.

## Conclusions

This paper presented a method for a model-based automatic generation of sequence control programs from design information including causalities of physical phenomena and geometric information of mechanisms. The SCP Generator, which implements this technique, succeeded in automatically generating a sequence control program. This results from integration of a CAD for functional design (the FBS Modeler), a synthetic reasoning system (QPAS), a geometric modeling system, a spatial reasoning system, and a sequence control software generation system on an integrated framework (KIEF) with the Metamodel mechanism. This method is applicable to sequence control software design of mechatronics machines in general. By allowing dynamic generation and modification of sequence control programs, these machines can become more flexibly reconfigurable, self-organizable, and robust, adapting themselves to environment changes, faults, and changes of the user's requirements.

## References


Faltings, B. 1987. Qualitative kinematics in mechanisms. In *Proceedings of IJCAI-87*, 436–442.

Fickas, S. F. 1985. Automating the transformational development of software. *IEEE Trans. on Software Engineering* 11(11):1268–1277.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* 24(3):85–168.

Graves, H. 1992. Lockheed environment for automatic programming. *IEEE Expert* 7(12):14–25.

Gupta, V., and Struss, P. 1995. Modeling a copier paper path: A case study in modeling transportation processes. In *Proceedings of the 9th International Workshop on Qualitative Reasoning*, 74–83.

Ishii, M.; Tomiyama, T.; and Yoshikawa, H. 1993. A synthetic reasoning method for conceptual design. In Wozny, M. J., and Olling, G., eds., *Towards World Class Manufacturing 1993*. Amsterdam: North-Holland. 3–16.

Joskowicz, L., and Sacks, E. 1991. Computational kinematics. *Artificial Intelligence* 51:381–416.

Kiriyama, T.; Tomiyama, T.; and Yoshikawa, H. 1991. The use of qualitative physics for integrated design object modeling. In Stauffer, L. A., ed., *Proceedings of Design Theory and Methodology - DTM'91-*, 53–60. ASME.

Matsumoto, M.; Kiriyama, T.; Tetsuo, T.; and Yoshikawa, H. 1993. Qualitative reasoning using a spatial modeler. In *Proceedings for the Annual Meeting of the Japanese Society of Artificial Intelligence 93 (In Japanese)*, 267–270.

Nakayama, Y. 1990. Model-based automatic programming for plant control. In *IEEE Proceedings of the Sixth Conference on Artificial Intelligent Applications*, 281–287. IEEE.

Tomiyama, T.; Sakao, T.; Umeda, Y.; and Baba, Y. 1995. The post-mass production paradigm, knowledge intensive engineering, and soft machines. In Krause, F.-L., and Jansen, H., eds., *Life Cycle Modelling for Innovative Products and Processes*. London: Chapman and Hall. 369–380.

Tomiyama, T. 1994. From general design theory to knowledge-intensive engineering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 8(4):319–333.

Umeda, Y.; Takeda, H.; Tomiyama, T.; and Yoshikawa, H. 1990. Function, behaviour, and structure. In *AIENG'90 Applications of AI in Engineering*, 177–193. Computational Mechanics Publications and Springer-Verlag.

Umeda, Y.; Tomiyama, T.; Yoshikawa, H.; and Shimomura, Y. 1994. Using functional maintenance to improve fault toleneance. *IEEE EXPERT* 2(3):25 – 31.

Umeda, Y.; Tomiyama, T.; and Yoshikawa, H. 1992. A design methodology for a self-maintenance machine based on functional redundancy. In Stauffer, L. A., ed., *Proceedings of Design Theory and Methodology - DTM'92-*, 317–324. ASME.

Yoshioka, M.; Nakamura, M.; Tomiyama, T.; and Yoshikawa, H. 1993. A design process model with multiple design object models. In *Design Theory and Methodology (DTM '93)*, 7–14. New York: ASME.