Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

Y. Yamaguchi, F. Kimura, P.J.W. ten Hagen

Interaction management in CAD systems with a history mechanism

Computer Science/Department of Interactive Systems Report CS-R8756 November

Bibliotheek
Centrum voor Wiskunde en Informatica
Amsterdam

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Copyright © Stichting Mathematisch Centrum, Amsterdam

69H12, 69K36, 69K70, 69D58

Interaction Management in CAD Systems with a History Mechanism

Yasushi Yamaguchi, Fumihiko Kimura
Department of Information Engineering,
University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan

Paul J.W. ten Hagen
Department of Interactive Systems,
Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

User friendliness is one of the unresolved problems in CAD systems. There are many possible directions for improving user friendliness. Understanding of the modeling process is one of the most important directions. It is natural for a user to describe the model in terms of its evolution. We call this concept *model derivation*. To construct and use model derivation, we propose a *history mechanism* which keeps and manipulates the history of the modeling process. The history mechanism manages high level interactions by introducing powerful symbolic computation to manipulate the history. Since the history representation is based on the operation's syntax and separated from the internal model representation, it is easy to apply the history mechanism to any modeling system which uses established techniques. Thus the system designer can easily introduce model derivation without reducing efficiency of the implementation.

1982 CR Categories: H.1.2, I.3.5, I.3.6, J.6, D.4.8.
Note: This report has been published in the proceedings of the Eurographics '87 Conference.

1. INTRODUCTION

In recent years a lot of mechanical CAD systems have been developed. At this moment, however, they are not useful tools for product designers to create the CAD data that CAM systems use. A desirable CAD system provides helpful functionality for the designers to construct product models during the modeling process. There are many difficulties to obtain such CAD systems. For instance, relevant data model for the product models is necessary, and many studies have been achieved.[1, 2, 3] However, on the contrary, little study has been done on user friendliness. The user friendliness of CAD systems still remains a difficult problem. Many people have ambiguously said CAD systems must be more interactive and intelligent in this sense.

There are several ways to develop user friendliness. One of them is to maintain the history of modeling processes. It is natural for a user to describe a model in terms of its evolution or derivation, however this is not possible in a conventional modeling system. Let us explain with an example. See figure 1. The top figure is usually represented by one arc, two segments, and two points in a conventional modeling system. We have no idea how the figure has been defined by the user. The round corner could have been created by either rounding a sharp corner or drawing two tangent lines to a circle. No sooner has the modeling system constructed an internal model, than the user's idea is lost. Understanding of the modeling process is important to improve user friendliness of modeling systems.

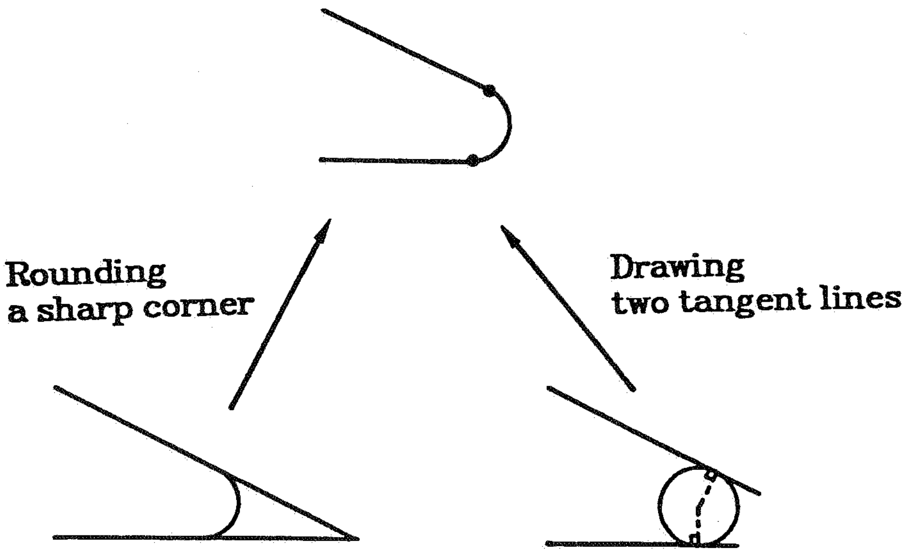

Figure 1. Two Ways to Create a Round Corner

We think it is important to provide a mechanism to keep *model derivation* by focusing attention on the modeling process. Modeling process is carried out in an exploratory way involving experimentation and backtracking. The process steps have to be captured into the model derivation that users can use to have a perspective view over all the modeling process. The model derivation must be managed in such a way that users will still feel unhindered while building models. Model modification is easily realized by means of obtaining a new model derivation. A mechanism of this sort will have to be developed if users are to create new derivations by analogy with existing ones. The concept, model derivation, reflects the interpretations of the internal models from the user's point of view, while conventional modeling systems provide no knowledge of the user's intention. Therefore, the mechanism managing the derivations must deal with the semantics of modeling as well as its syntax. It is essential that powerful semantically based modeling tools change the way to create and modify the models.

In order to satisfy these required conditions for the model derivation management, we propose a *history mechanism* which maintains the model derivation by keeping the history of the modeling process and manipulating the history symbolically. This symbolic-computation capability is the heart of the history mechanism. Our premise is that it is possible to construct model derivation from a sequence of modeling steps, i.e., operations. The history representation is based on the syntax of procedural operations. This syntax base representation is useful in introducing the axiomatic method of formal semantics.[4, 5] The history mechanism also guarantees the independence of model derivation from model representation. It is easy to apply the history mechanism to conventional modeling systems by the separation between model interpretations and representations. Thus system designers of modeling systems need not worry about maintaining an interpretation and ensuring an efficient implementation.

The input history of user's commands has been utilized during the interaction process.[6, 7] It is, however, impossible to maintain the model derivation only with the input history in the form of text composed of character strings (i.e., a log file). An approach to keeping the user's idea of a model has been to provide such a flexible data representation that the resulting model contains an interpretation of itself.[8] In other words, it has tended to allow several representations of the identical object to exist according to the user's intentions. However, this kind of method leads the modeling system to less efficiency in calculation. Furthermore, the data representations are so unsuitable for applications that judging the equivalence of models is quite difficult.

Section 2 describes the representation of the history which is required to represent the model derivation. Section 3 shows how the history manager traces the dependency network of the history in order to utilize the model derivation and points out some other characteristics of the history mechanism. Section 4 contains the examples that were applied to a simple drafting system. By applying the history mechanism, the drafting system has been improved to handle variational geometry.[9] Section 5 summarizes this study and indicates future plans.

2. HISTORY REPRESENTATION

The history mechanism is realized by the *history manager* module situated between the user interface and the kernel of the modeling system. The history manager supervises the modeling process in terms of monitoring the interactive process. Figure 2 shows the system configuration. The history manager stores the history of a user's interaction in its own database. This is the characteristic point of this study. Studies on data models such as geometric reasoning have tended to store model derivation in the database of the modeling system itself. The database of the history manager also keeps several general rules concerning the interaction. The history manager is able to complete the interaction at a high level by applying those rules to the history.

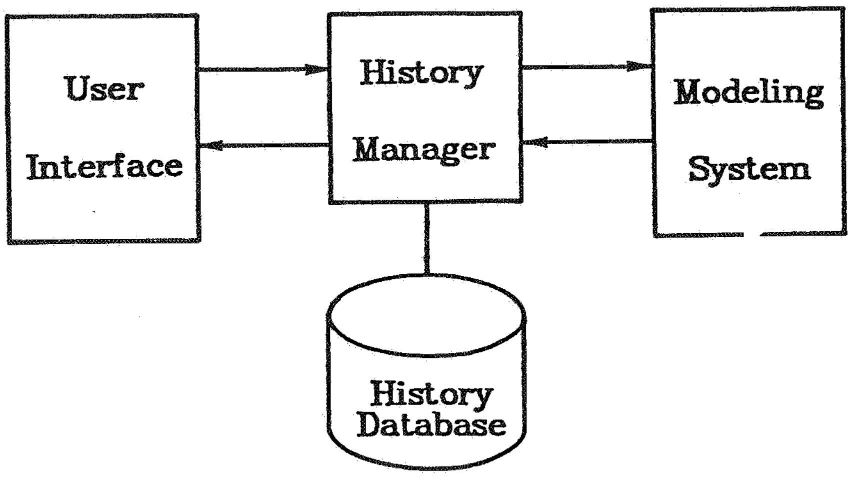

Figure 2. History Manager in History Mechanism

2.1. History Item

Our premise is that an operation (i.e., a command) corresponds to a step in the modeling process. This means that the history of the interaction can be expressed in records of the operations executed by the user. The history manager adds a *history item* to the history database every time an operation is executed. The history item is a framework to represent the history corresponding to each operation.

A history item consists of the two predicates of the *input* and *output* relations. Both predicates assert the relations between operations and data objects related to the operations (i.e., entities in the modeling system). An input relation shows the list of the entities and numerical values that the operation took as its arguments. An output relation tells the list of entities generated by the results of the operation. The history manager treats each operation as an instance object. Let us consider an operation X which takes N arguments and generates M results. The history item of this operation would be:

input (operationX, arg1, arg2, ..., argN)
output(operationX, res1, res2, ..., resM)

2.2. Rules in the Operation Class

Each operation is treated as an instance object within a history representation. We can categorize the same kind of operations as a class and represent their characteristics using rules about their syntax and semantics. Syntactic rules specify the syntax of both input and output. The input syntax means the number of input arguments with their data type and range. In the case of the output syntax, the number might be variable but the maximum number is fixed. By introducing axiomatic methods of formal semantics, the operation's semantics can be represented in predicates based on the syntactic rules. Semantic rules show the logical relations between the arguments and the resulting outputs of the operations.

Let us explain a simple example for a drafting system. Consider an operation called *'Intersect_curves'*. used to determine the intersection points between curves. The operation takes two curves, and calculates the intersect points of these curves and creates their data. Assuming that the curve types handled in the system are only a straight line and a circle, the number of resulting points should be less than three. Figure.3 shows an example of this operation for a circle and a line.

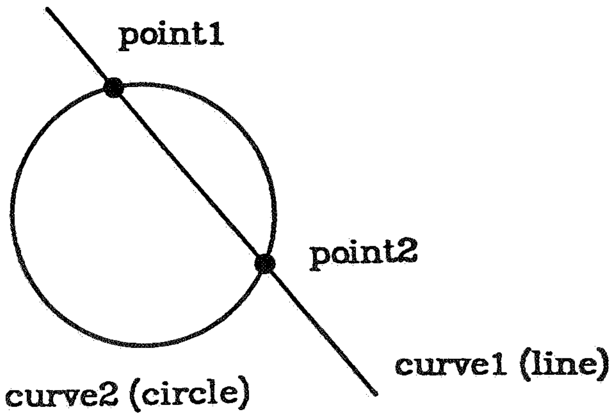

Figure 3. Intersect_curves

The history item for a certain operation of this kind looks like this:

input (operationX, curve1, curve2)
output(operationX, point1, point2)

This history item implies some syntactic specifications of the operation. The operation requires two input arguments and produces two output results. There are some other syntactic rules about arguments and results such as data type and range. In this example, we restrict the type of curves to lines and circles. The results must be points. We can also represent the semantics of the operation as rules. The intersection points are the points which both input curves pass through. The syntactic and semantic characteristics of this operation can be represented in the following rules:

OPERATION_CLASS
    class_of(Operation, 'Intersect_curves')

SYNTAX
```
    input (Operation, Arg1, Arg2)
  & output(Operation, Res1, Res2)
  & (type_of(Arg1, 'line') | type_of(Arg1, 'circle'))
  & (type_of(Arg2, 'line') | type_of(Arg2, 'circle'))
  & type_of(Res1, 'point')
  & type_of(Res2, 'point')
```

SEMANTICS

```
     coincide(Res1, Arg1)
   & coincide(Res1, Arg2)
   & coincide(Res2, Arg1)
   & coincide(Res2, Arg2)
```

The operator '|' means OR, while '&' means AND. The predicate *class_of(i, c)* defines the class of the operation, with *i* being an instance of the operation class *c*. The predicate *type_of(d, t)* specifies the type of the data object *d* as *t*. *Coincide(p, c)* asserts the geometric relation that the curve *c* passes through the point *p*.

3. HISTORY USAGE

Modeling activity is processed in an accumulative way. A model is constructed incrementally such that a user generates an entity with an operation and he can use the entity in future operations. The former operation affects later operations by the entity. In other words, later operations depend on the entity and the operation. The history of the modeling process expresses dependencies that are closely related to the model derivation. The history manager uses the history by tracing these dependencies.

The dependencies in the history mechanism are expressed in the following manner. Suppose there are two consequent operations *X* and *Y*. The latter operation *Y* uses the entity *H* that has been generated by the former operation *X*. According to the definition of the history item, both the output relation of *X* and the input relation of *Y* contain the entity *H* :

output(operationX, ..., entityH, ...)
input (operationY, ..., entityH, ...)

These two relations point out the dependency between the operation *X* and *Y* mediated by the entity *H*.

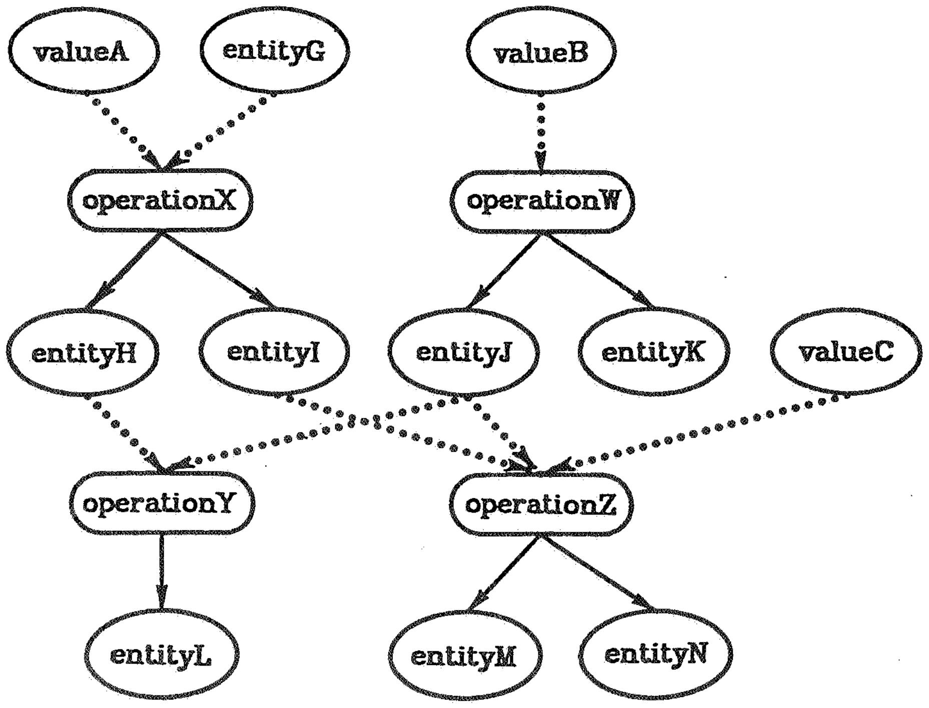

Figure 4. Dependency Network of History

Since many operations are executed and many data objects are related to those operations during the modeling process, there are a great number of dependencies generated. The dependencies among operations and data objects form a complex network propagating throughout the whole history, figure 4. Solid line arrows stand for output relations, dotted line arrows stand for input relations. The history manager traces this dependency network in two directions, in the direction of the arrows and in the opposite direction. Since each node is connected to multiple arrows, the tracing in either direction might be propagative. We named these characteristics of tracing history as the downward and upward propagation of the history. Roughly speaking, a downward propagation implies that an operation affects later operations and resulting entities. An upward propagation implies that an entity is derived from former operations and related entities.

3.1. Downward Propagation of History

One goal of this history mechanism is to build an experimental environment for modeling activity. The history mechanism allows modification of models by replaying the modeling process according to the modified history. Let us explain the replay mechanism without any modification before discussing history modification. To replay the whole modeling process is easy, if the history keeps the contents of past operations in sequence. The history items are stored in the same order as the operation sequence. The history manager takes the input relation one by one from its database and reinvokes respective operations. It can be said that replay is accomplished by sequential tracing which has a downward tendency. This however is linear rather than propagative.

The problem in the replay process is that the history items are represented with only entities valid in the original modeling process. Whenever the input relation includes some entities, the new operation must be invoked with the new entities having been generated in the replay process. The original input relation cannot be used alone. The history manager has to adjust it to the context of the replay process. The correspondence between the original entities in the history and the new entities is necessary for the adjustment. To make this correspondence, the history manager matches the results of a new operation with those of the original operation each time. In this sense, the history can be seen as a scenario in the replay process.

The history mechanism provides the capability of model modification by means of modifying the history. The history is treated as a scenario including some parameters in the modification process. By featuring arguments in operation rules, the history manager knows which arguments are changeable. For instance, consider an operation to make a circle with its center point and radius. The class rules will be like this:

```
OPERATION_CLASS
        class_of(Operation, 'Make_circle')

SYNTAX
        input (Operation, Arg1, Arg2)
    &   output(Operation, Res1)
    &   type_of(Arg1, 'point')
    &   type_of(Arg2, 'floating_point_number')
    &   type_of(Res1, 'circle')

SEMANTICS
        center_of(Arg1, Res1)
    &   radius(Res1, Arg2)

MODIFICATION
        changeable(Arg2)
```

The point is that the new predicate *changeable* is introduced in the new section named modification. The predicate *changeable(a)* specifies that the argument *a* can be changed in the modification process.

It is easy to imagine that the history manager might fail to match the new entities with those of the original operations owing to the history changes. For instance, remember the operation *'intersect_curves'* between a line and a circle. A user might change the circle radius so small that they never intersect each other. In this case, the new operation outputs no points and the history manager fails to match the results for the history as scenario. When the history manager finds this situation, it warns the user of the mismatching. The history manager ensures topological equivalence of the models by checking number and type of the result entities for each operation. The history manager can investigate the causes of the mismatching by upward tracing which will be mentioned in the next section.

3.2. Upward Propagation of History

The history mechanism performs model modification by means of modifying the history as shown before. A user must select the relevant operations to modify the result model. However the relationship between the model and the executed operations is not evident to him even if he has built the model. In order to support the exploratory modeling, the capability of pointing out the causes from the history is also necessary. These causes are simply the model derivation we presented as the goal of our study in the previous section. The history mechanism provides the model derivation in terms of the mapping from the result model to the causing operation. This mapping can be obtained by tracing the history in an upward direction. The predicate *changeable* also takes part in this investigation process, because a user is specially interested in those points of the history where he has several alternatives. The predicate specifies all of those points. The history manager traces back the history, distinguishes the operations that have some changeable arguments, and enumerates these operations to the user.

The history can be trimmed by upward propagation. The modeling process involves experimentation and backtracking. This means that the history should include many operations which have had no effect on the final result model. These operations are unnecessary to keep in the history. By tracing the history from the result model in an upward direction, the history manager can select all operations affecting the result. The trimmed history is enough to reconstruct the result. Because of the capability of history modification, this trimmed history is equivalent to the variational representation of the result model.

3.3. Characteristics of the History Mechanism

Past research on the data model has tended to focus on methods for reasoning to maintain the interpretations of individual models. To obtain the reasoning mechanism, it forces modeling systems to introduce the logic base data representation that makes the systems less efficient and forces the system designer to construct model derivation mechanisms by himself in the modeling system. The logic base data representation is also disadvantageous for the applications that use the result model.

On the contrary, the history mechanism tends to separate the model derivation from the internal model representation. The history representation is completely independent from the internal model representation. The only thing that a system designer has to do is to specify the characteristics of operations, as we have shown in previous sections. The designer need not be concerned with the internal data representation or the processing mechanism. The history mechanism maintains model derivation and supports exploratory modeling by itself. The operation specifications expressed in predicates are based on the syntax of the procedural operation. Therefore, it is easy to apply the history mechanism to a modeling system using established techniques.

The representation of the operation's specifications in predicates is useful in introducing the axiomatic method of formal semantics. The history mechanism makes it possible to represent the semantics of each operation. By means of additional semantic rules, it becomes possible to handle complex relations which are caused by plural operations. For instance, it can deduce the relation *concentric(c1, c2)* from the fact *center_of(p, c)* which is asserted by the operation *'Make_circle'*. The inference rule should be:

```
concentric(C1, C2) ←
      center_of(P, C1)
  & center_of(P, C2)
  & not(equal(C1, C2))
```

The semantics can be used in the modification process. Suppose there are two concentric circles. Since those circles stand for a boss, a bearing, etc. in a mechanical drawing, the size difference between them is elementary. The larger circle should be modified such that the resulting circle is larger than the other modified circle, in spite of the modification. The relation concentric is useful in this context.

4. EXAMPLES

We have implemented a prototype of the history manager using Lisp (Kyoto Common Lisp)[10, 11] on a VAX/UNIX† system. History items and rules are represented in symbolic expressions. The prototype has been applied to a simple drafting system which is written in the C program language. This drafting system itself provides only simple drawing utilities such as *pick_object_with_mouse*, *draw_parallel_line*, *draw_tangent_line*, *draw_cotangent_line*, *intersect_curves*, *make_circle*, etc. The internal data representations of the drafting system are so simple that each entity contains only numerical values. For instance, a *circle* entity has only three floating point numbers specifying the coordinates of its center and its radius.

In this drafting system, every drawing process begins with a origin and two axes (X-axis and Y-axis), which are used as reference geometry. Figure 5 shows an example of the early stage of a drafting process just after the first five operations:

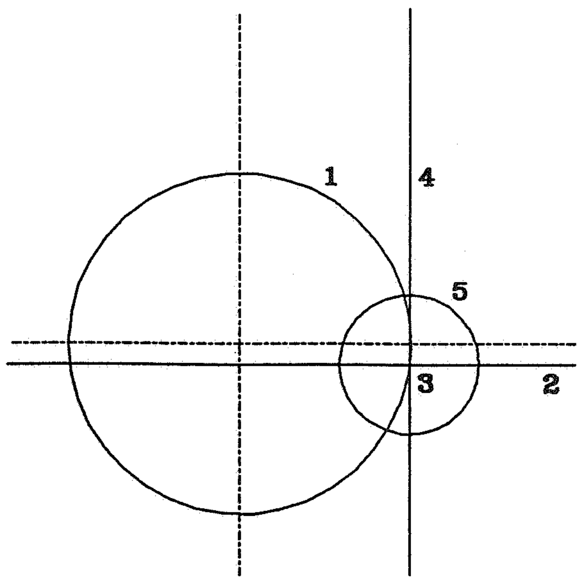

Figure 5. An Example of Early Stage of Drafting Process

† UNIX is a trademark of Bell Laboratories.

1. The user has made a circle whose center was the origin,
2. drawn a line which is parallel to the X-axis,
3. made a intersect point between the circle and the line,
4. drawn a line which was passed through the intersect point and is parallel to the Y-axis,
5. made a circle whose center is at the intersect point.

The result shape was generated by 122 operations which had actually affected the result (Figure 6). By using the inference rule mentioned in the last section, the system can define concentric circles in the drawing. Figure 7 displays all concentric circles defined by this mechanism. This relation can be used in the modification process so that a circle is automatically modified according to the changing of the concentric circle.

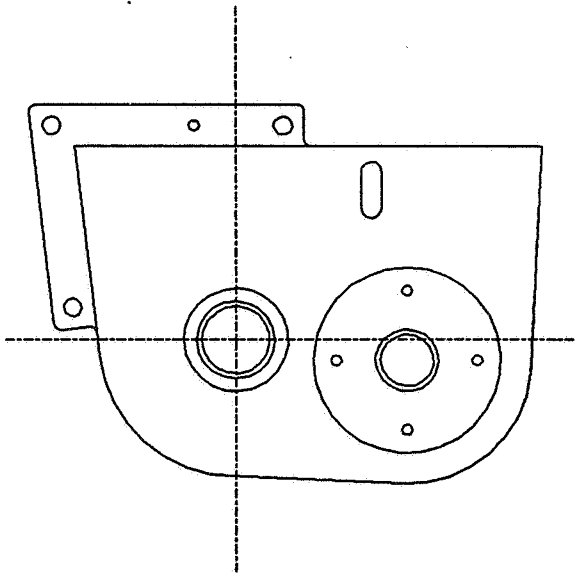
Figure 6. The Result Figure

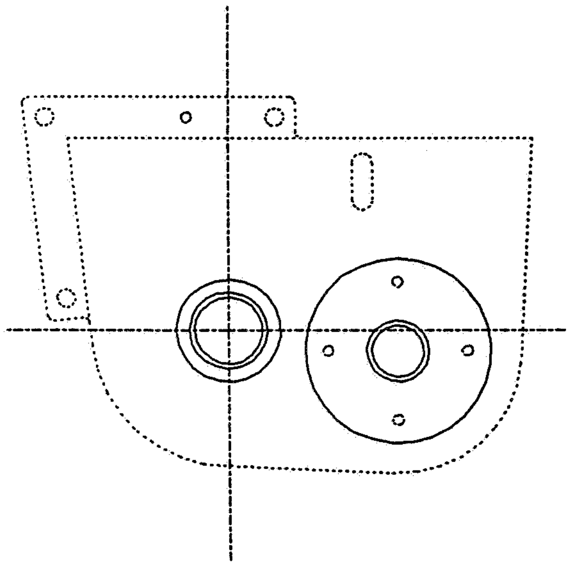
Figure 7. The Display of Concentric Circles

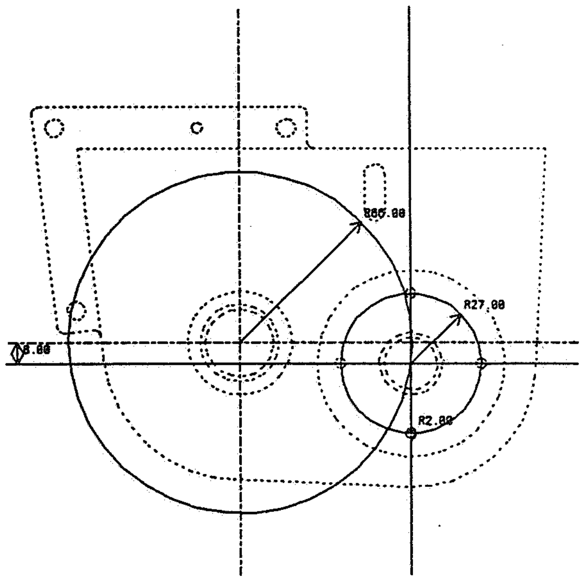

Figure 8. The Display of Causing Operations

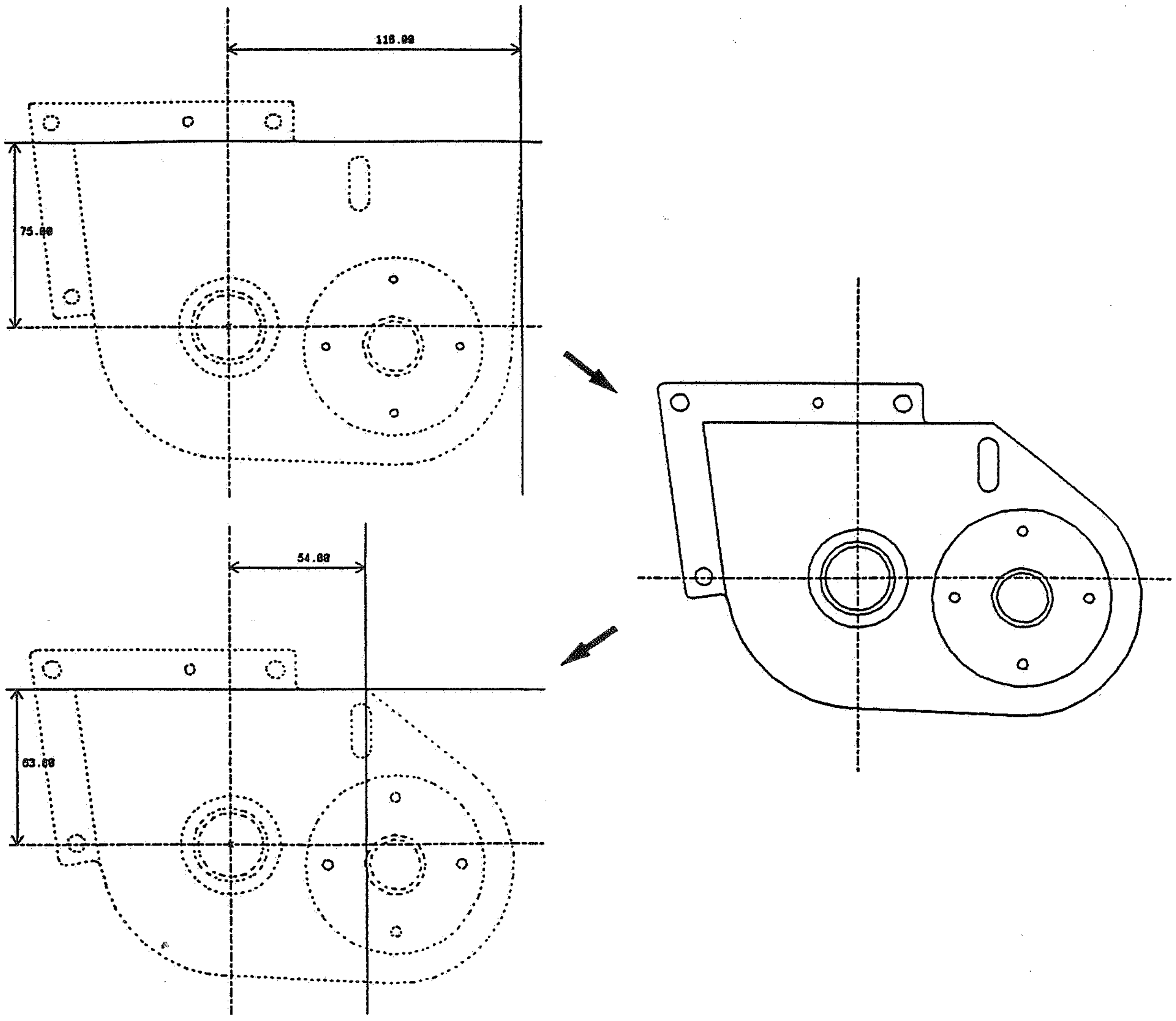

Figure 9. Typical Cycle of the Model Modification

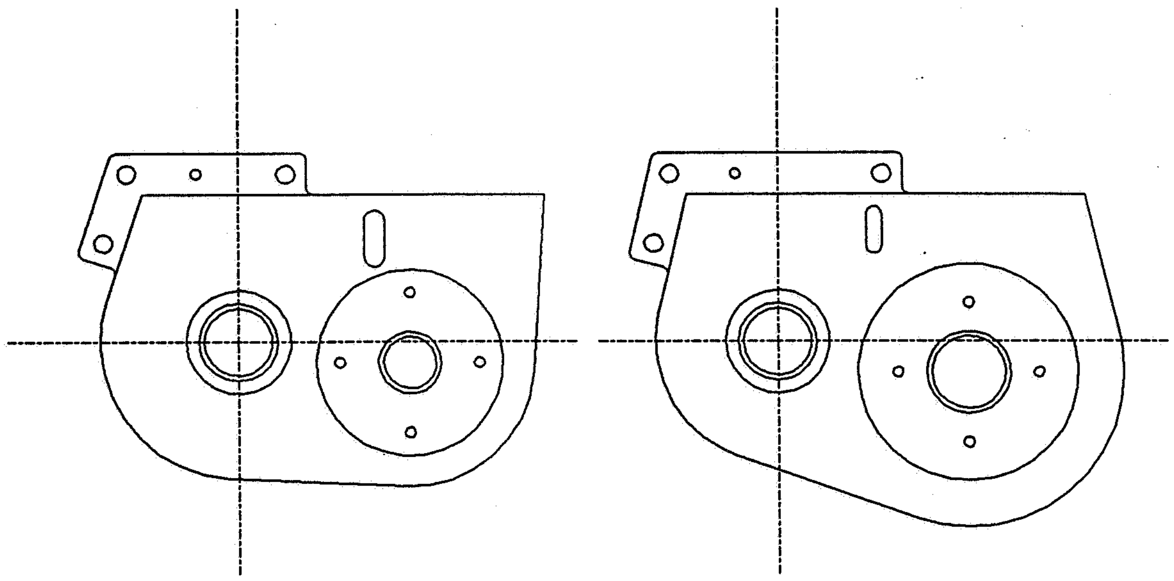

Figure 10. The Modified Figures

The system can show the causing operations and relating entities by means of the upward propagation of the history. In the case of figure 8, the system traced the history from the little circle which was picked by the user with a mouse. According to the predicate *changeable*, the system pointed out the geometric values that the user can change in the modification process. Figure 9 shows the typical cycle of the model modification with the history mechanism. The user requested the causing operations for the location of the point at the upper right corner. The system replied that the point was defined by intersecting lines which are parallel to X-axis and Y-axis respectively. As the user modified the distances between the line and the axis, the system modified the figure only in those parts which have been changed by the downward propagation of the history. Again the user requested the causing operations. By the repetition of the modification cycles, the user easily changed the shape (Figure 10). The modification of a figure is so easy done that the drafting system looks as if it supports variational figures by itself.

5. CONCLUSIONS

In this paper we proposed a history mechanism to improve the user interaction of CAD systems. This was achieved by keeping a history of the modeling process composed of the user's interaction in order to maintain the model derivation. The history mechanism has the following characteristics:

1. The history representation of the history mechanism is completely independent from the internal data representation of the modeling system.
2. The history representation is based on the syntax of procedural operations.
3. Users still have creative freedom while constructing models with the history mechanism.
4. Based on the symbolic manipulation capability, models can be modified by creating new derivations by analogy with original ones.
5. By introducing the axiomatic method of formal semantics, the history mechanism can also handle semantics of operations.

By means of these characteristics, the history mechanism obtains the following advantages:

1. The history mechanism can be easily applied to the established techniques for CAD systems. The system designers of modeling systems need not worry about simultaneously managing high level interactions and ensuring an efficient implementation.

2. The history mechanism provides sufficient utilities to support a trial and error method of modeling, which is the upward and downward propagation of history.
3. The history mechanism will clarify the user's intention by applying a deductive inference on the model derivation.

Much study remains to be done concerning the history mechanism. One primary subject is history modification. At the current stage of our research, the history can be modified only with the special arguments of operations. A history editor that will allow a user to modify the history much more flexibly shall be introduced to the history mechanism. The environments of the history mechanism should be extended. One aspect of the environment is a sophisticated user interface. Since our prototype system is just a trial history mechanism, the user interface is still poor. A user interface that processes a powerful graphics capability is desired. The interaction technique library for general purpose can be supplied by carefully analyzing the connection between the history manager and the user interface. The other aspect of the environment is a modeling system. Since our final goal is the realization of high level interaction management for a product modeling system, we would like to apply a history mechanism to a currently existing prototype product modeling system.[2, 12]

ACKNOWLEDGEMENTS

This work has been partly supported by the Product Modeling System's Development Project of Japan Society of Precision Engineering and by RICOH Co.,Ltd.

REFERENCES

[1] Fumihiko Kimura, Shinji Kawabe, and Toshio Sata, "A Study on Product Modelling for Integration of CAD/CAM," *Proceedings of the IFIP WG.5.2/WG.5.3 Working Conference on Integration of CAD/CAM*, pp. 227-252, North-Holland, 1984.
[2] T. Sata, F. Kimura, H. Suzuki, and T. Fujita, "Designing Machine Assembly Structure Using Geometric Constraints in Product Modelling," *Annals of the CIRP*, vol. 34, pp. 169-172, 1985.
[3] F. -L. Krause, P. Armbrust, and M. Bienert, "Methods Banks and Product Models as Basis for Integrated Design and Manufacturing," *Proceedings of the 2nd International Conference on the Manufacturing Science and Technologie of the Future*, Slovenian Academy of Science and Art Ljubljana, Sept. 1985.
[4] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.
[5] T. M. V. Janssen, *Foundations and applications of Montague grammar Part 1: Philosophy, framework, computer science*, Centre for Mathematics and Computer Science, Amsterdam, 1986. (CWI Tract 19)
[6] William Joy, "An introduction to the C shell," in *UNIX Programmer's Manual*, Berkeley, November 1980.
[7] Tapio Takala, "User Interface Management System with Geometric Modeling Capability: A CAD System's Framework," *IEEE Computer Graphics and Applications*, vol. 5, no. 4, pp. 42-50, IEEE Computer Society, April 1985.
[8] Farhad Arbab and Jeannette M. Wing, "Geometric Reasoning: A New Paradigm for Processing Geometric Information," *Proceedings of the IFIP WG.5.2 Working Conference on Design Theory for CAD*, pp. 145-165, North Holland, 1987.
[9] Robert Light and David Gossard, "Modification of geometric models through variational geometry," *CAD*, vol. 14, no. 4, pp. 209-214, Butterworth & Co.,(Publishers) Ltd., July 1982.
[10] Guy L. Steele, Jr., *Common LISP: The Language*, Digital Press, Burlington, 1984.
[11] Taiichi Yuasa and Masami Hagiya, *Kyoto Common Lisp Reference Manual for VAX11 Unix 4.2bsd*, Research Institute for Mathematical Science, 1984.
[12] Fumihiko Kimura, Hiromasa Suzuki, and Lars Wingard, "A Uniform Approach to Dimensioning and Tolerancing in Product Modelling," *Preprints of CAPE'86*, pp. 165-178, 1986.