Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2002), *16*, 73–91. Printed in the USA.

DOI: 10.1017.S0890060402020073

Acquiring engineering knowledge from design processes

YOKO ISHINO AND YAN JIN

IMPACT Laboratory, Department of Aerospace and Mechanical Engineering, University of Southern California, Los Angeles, California 90089, USA

(RECEIVED January 30, 2001; ACCEPTED December 11, 2001)

Abstract

Knowledge management has recently become the focus of public attention in business and engineering. Because knowledge acquisition is situated in the upstream of knowledge management, capturing knowledge is an important step for enterprises to achieve successful knowledge management. We focus on *how engineers solve their design problems under given design contexts* and propose a novel model and methods to capture knowledge from engineering design processes. Our goal is to acquire *know-how* knowledge without disrupting the normal design process. A *three-layer design process model* is introduced to represent generic design processes, and a *grammar and extended dynamic programming* (GEDP) method is developed based on the process model. GEDP adopts the *grammar approach* and EDP to automatically identify meaningful clusters, called *operations*, from primitive design *events*. Our approach is evaluated through a case study of designing a double-reduction gear system.

Keywords: Design Process; *Know-How* Knowledge; Knowledge Acquisition; Strategic Knowledge; Three-Layer Design Process Model

1. INTRODUCTION

Engineering design is highly knowledge intensive in nature. In addition, in recent years, industrial design projects have grown larger in scale and more complex in content. In practice, a design task is usually divided into a number of highly coupled subtasks that require multiple designers to work together collaboratively. Some of the designers may be more skillful and have more experiences than the others. To maintain the quality of the overall design, it is desirable for designers to clarify and share their knowledge. Developing ways to capture engineering knowledge from experienced designers without disturbing their normal design process is a key to achieving successful knowledge sharing.

Although the value of capturing, managing, and utilizing design knowledge has long been recognized and much research has been carried out, capturing engineering knowledge without disturbing the designers' normal design process is still a challenge. Design processes are often ill structured and *ad hoc* and vary greatly, depending on the design contexts. Although operation data recorded by CAD systems may contain valuable information, it is difficult to generalize ways of obtaining useful design knowledge directly from the operation data.

On the other hand, asking designers to record and share their design knowledge may also be problematic. There are several reasons: (1) a substantial time commitment is required for a designer to record his or her design knowledge; (2) design knowledge is tacit and embedded in the design process in most cases, so it will be difficult for a designer to express it fully and accurately; and (3) forcing designers to record their knowledge would interrupt their natural thinking process and become an unacceptable burden for them.

Our goal is to automatically capture engineering knowledge that can facilitate the understanding of a designer's design intent and provide guidance for designers to explore alternative designs, without disrupting the normal design process. There are several research questions that must be addressed to achieve this goal. What is engineering knowledge? What kind of knowledge should be acquired and accumulated in a design process? These questions are the crux of a knowledge system in engineering design. Furthermore, we must also understand what a design process is. This is a critical question for extracting design knowledge from design processes. To address these questions it is necessary to

Reprint requests to: Dr. Yoko Ishino, 375 Central Avenue, #145, Riverside, CA 92507, USA. E-mail: okinaka@usc.edu

clarify knowledge structure in the engineering design field and develop a model of design processes.

This paper categorizes design knowledge and emphasizes the importance of *know-how* knowledge. A generic design process model called a *three-layer design process model* is proposed. Based on this model, a novel method called *grammar and extended dynamic programming* (GEDP) is introduced to acquire *know-how* knowledge. Generally, design knowledge and design processes may both vary, depending on the structure of the design organization, whether it is a team's collaborative design or a single designer's design, because coordination and negotiation are indispensable to collaborative design. For simplicity, this paper focuses on a single designer's case. However, the proposed model and method can be extended to team collaborative design situations. We successfully tested our knowledge acquisition framework in a case study of designing a double-reduction gear system.

This paper is organized as follows. Section 2 clarifies and categorizes engineering design knowledge and explains what knowledge on which we focus and why. Section 3 proposes a generic design process model called the three-layer design process model and describes the model in detail. In Section 4 we introduce the novel GEDP method for the acquisition of *know-how* knowledge and describe how it works. Section 5 presents a case study where the proposed model and methods are applied to a practical double-reduction gear system design problem. Sections 6 and 7 discuss the related work and makes concluding remarks, respectively.

2. ENGINEERING DESIGN KNOWLEDGE

2.1. Categories of engineering design knowledge

Many practical design problems have the following characteristics:

- One design task consists of multiple activities.
- These activities are dependent on one another.
- The product alternatives are evaluated based on multiple requirements.

To solve a complex design problem effectively and efficiently, different types of knowledge are utilized. There are many dimensions in which knowledge can be characterized. Examples of the dimensions include knowledge *representation* (e.g., symbolic knowledge vs. numerical knowledge), *availability* (e.g., documented knowledge vs. unwritten knowledge), *accessibility* (e.g., tacit knowledge vs. explicit knowledge), and *application function* (e.g., domain knowledge vs. strategic knowledge). Because we are interested in capturing engineering knowledge that can guide design activities in various design contexts, we categorize design knowledge based on application function, as shown in Figure 1.

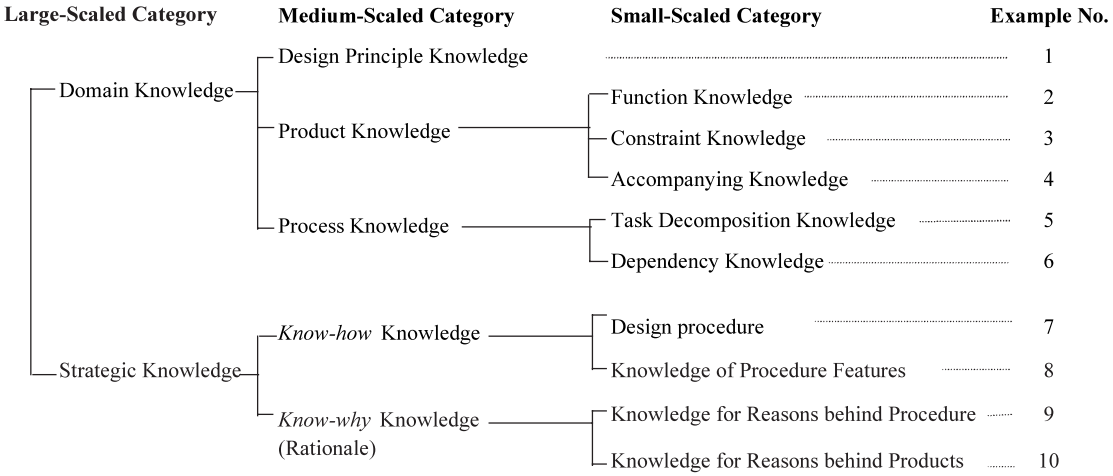
Domain knowledge is applied to characterize domain design problems. It is usually explicit and does not vary, depending on the designers. Domain knowledge can be categorized into three types. The first type of knowledge is called principle knowledge. It is knowledge about general principles of design. For example, principles of physics and principles found in systematic design (Pahl & Beitz, 1996) and axiomatic design (Suh, 1990) are principle knowledge. The second type of knowledge is product knowledge. It consists of function knowledge, constraint knowledge, and accompanying knowledge. Function knowledge is related to the physical or mechanical function and specification of products. Constraint knowledge is related to requirements that a product should meet. Accompanying knowledge is related to secondary needs of products, for example, customers' preferences, and so forth. Process knowledge is the third type of domain knowledge. It is about the design process, including information for task decomposition and the dependency information of the process. Task decomposition knowledge is about the manner in which design tasks are decomposed, and dependency knowledge describes relationships between subtasks. The content and magnitude of dependency between subtasks are affected by the task decomposition schemes.

On the other hand, strategic knowledge is about the way a designer proceeds with his or her design and what the designer's intent is that leads to the ways the designer performs his or her design. Strategic knowledge is usually implicit and designer specific. It can be categorized into two types of knowledge: *know-how* knowledge and *know-why* knowledge. *Know-how* knowledge is about the ways to identify design opportunities, define design directions, and manipulate design situations. In our research, we define design *know-how* knowledge as the *knowledge about design procedures*. A typical design procedure is a sequence of design operations. In many design situations, design procedures may exhibit certain patterns, for example, certain operations may appear in sequence, and certain operations always proceed, or succeed, in a specific design procedure. We call the patterns *procedure features*. *Know-how* knowledge, that is, design procedures and procedure features, reflects a designer's procedural intent or strategy. In practice, the design procedure and procedure features are often not well documented. Generally speaking, more experienced designers have more *know-how* knowledge and can select and apply it more efficiently.

Know-why knowledge, on the other hand, signifies why the object is designed the way it is. *Know-why* knowledge, which is also called design rationale, has two categories, knowledge for reasons behind procedure and knowledge for reasons behind products. The former means a conglomerate of detailed reasons why a designer did each operation. The latter indicates a designer's own decision mechanism for an assigned design mission. Because reasons behind design products can hardly be extracted from the products themselves and designers' evaluation criteria are often implicit, it is difficult to acquire design *know-why* knowledge.

Domain knowledge is usually explicit and describable before designing. Strategic knowledge, however, is often

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press



Example No.	Example of Knowledge
1	Layout the major functions first, and then get into auxiliary functions.
2	If volume, V, is fixed, pressure, P, is calculated by the equation, $P=nRT/V$.
3	The required reduction ratio should be accomplished strictly within +-4.0%.
4	Though the shape of a container doesn't affect the main function, a streamlined form seems preferable.
5	For decomposing gear design task, Plan P will cause less dependencies than Plan S does.
6	Activity A, deciding a gear material, and Activity B, determining a pitch of the gear, are mutually dependent.
7	Activity C, examining the effect of gear teeth, should be tried before Activity D, examining the effect of a gear material.
8	In order to reduce the cost, it is effective to change the gear material from No.1 to No.2.
9	Once Angle E had been designed as 60 degrees and then it was changed into 45 degrees. It was because at first the former condition seemed to provide comfortable space for users, and then the designer realized that 45 degrees was sufficient and it could work more compatibly with othre parts.
10	Although lighter, smaller and cheaper is better, the most important factor the cost.

Fig. 1. Knowledge categories and an example.

tacit and dependent on a designer's ability. From a knowledge management point of view, domain knowledge has been well collected and shared within organizations through development of product models (Bradshaw et al., 1997). Strategic knowledge, on the other hand, is often hard to share because of the difficulty of its acquisition. In this paper we focus on acquiring *know-how* knowledge as part of strategic knowledge. *Know-how* knowledge directly corresponds to useful design knacks and sharing it among designers can contribute to more effective collaboration. Moreover, it is still a challenge to capture *know-how* knowledge automatically. We argue that *know-how* knowledge is important engineering knowledge that should be shared among designers. In the following, we proposed a specific approach to acquire the *know-how* knowledge by analyzing design history.

2.2. *Know-how* knowledge acquisition

In order to not interfere with a designer's normal design process, we take an action-based knowledge capturing approach and focus on the data that can be obtained through observing design activities using a CAD system. To make it possible to collected needed data, the CAD system we use must contain domain knowledge and function as an interface for the designer to use other engineering tools. While a designer does his or her design through the CAD system, all actions he or she takes during the design process will be recorded. This approach alleviates the problem of interfering in the design process, but it creates a new problem in managing the large volume of information recorded. The model and methods to elicit meaningful chunks of knowledge from an enormous pool of data must be devised.

As depicted in Figure 2, our architecture to capture *know-how* knowledge contains two main modules: the monitoring module and the knowledge-capturing module. The former records designing events by monitoring a designer using the CAD tool. The latter module consists of two components: the data integration and the analysis. The data integration component translates design history from the design event log into a sequence of meaningful "*hows*." The analysis component identifies and explains procedure features in the sequence of *hows*. The mechanisms of the knowledge-capturing module are described in Section 4, and the model on which the mechanisms are based is discussed in Section 3.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

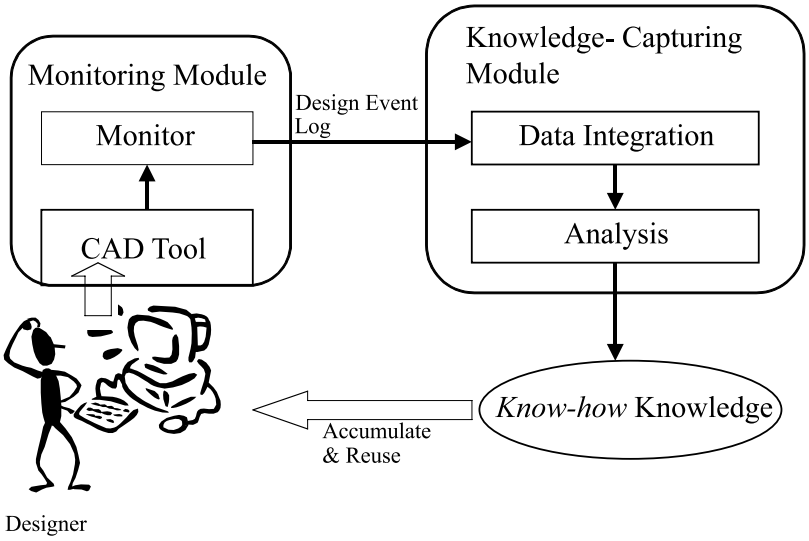


Fig. 2. The architecture of *know-how* knowledge acquisition; information flow.

3. THREE-LAYER DESIGN PROCESS MODEL

Designers usually decide on a specific design after many trial cycles. Based on our observation of designers' behaviors, most designers make tentative design decisions and create a prototype design at first. They then repeatedly adjust and refine certain design parameters to meet specific requirements. Finally they reach the final design after evaluating all alternatives that have been explored. This design process can be viewed as a trial and error process and designers seek the satisfactory solution according to their strategy. From this observation, we propose a three-layer design process model to capture general design processes. The three-layer design process model is schematically illustrated in Figure 3.

The three-layer design process model represents generic design processes based on three layers of information, namely, Event-Layer, Operation-Layer, and Product Model-Layer. Event-Layer captures primitive-level design events that are generated by designers through operating a CAD system. For example, "Change length A from 15 to 30" or "See document B," illustrated as "E" in Figure 3, is an event that occurred at Event-Layer. Operation-Layer represents higher level design operations that reflect meaningful design actions. Elements at Operation-Layer are design operations that can be generated by reasoning based on multiple design events found at Event-Layer. "Decrease the weight of the object" and "Increase the strength of the arm" illustrated as "Op" in Figure 3 are examples of design operations. The elements in Product Model-Layer are design alternatives, which are generated from multiple design operations, and are illustrated as "P" in Figure 3. An element in the Product Model-Layer is called a design alternative.

Based on our model of design processes, the goal of designers can be considered to be to create a final product model. To do so, designers intentionally plan and perform sequences of operations (Ops). Although the sequences of operations a designer performed cannot be observed directly, we can capture the events (Es) that were generated while the designer was performing the operations. In our research, we consider the sequences of operations as design *know-how* knowledge and capturing the operation se-

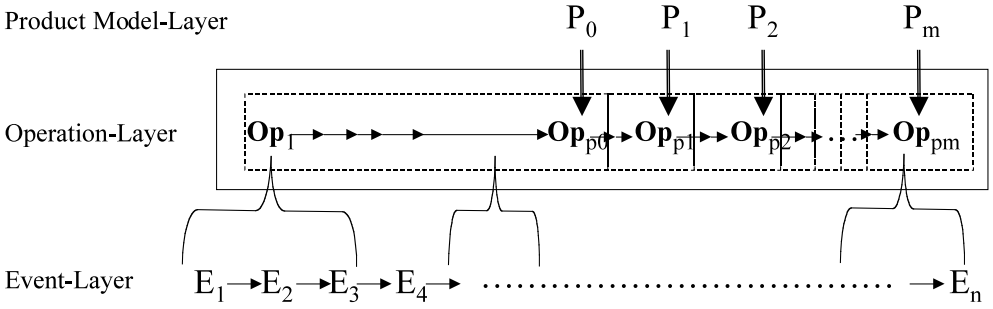


Fig. 3. A three-layer design process model; **E**, each event caused by a designer (e.g., "Change length A from 15 to 30," "See document B"); **Op**, meaningful action under a designer's certain intention, which is a cluster of plural events (e.g., "Decrease the weight of the object," "Increase the strength of the arm"); **P**, product model (design prototype) that represents design alternatives.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

quences of a designer will allow us to understand *how* the designer did his or her design.

Discovering *know-how* knowledge requires understanding of an executed design process in which many product models were created, explored, and discarded or adopted. Knowing the design operations performed to create each product model and the relationships between product models are necessary to put the design process into perspective. The goal is to find out what subsets of design operations are important features and where the features appear in the design process. Indeed, analyzing how product models relate to each other in their representative parameters provides us with an overall map of how product models were generated, explored, discarded, and finally adopted. Moreover, analyzing what sequences of operations were done to create a certain product model state may give us insights about why the designer did the operations at that point.

In addition to the three-layer model described above, we introduce a hierarchical structure of product models that represents the development process of design. The hierarchical structure is a time-series branching tree of product models in which a newly developed product model becomes a child of one of existing leaf product models. Basically, a child product model is derived from its parent product model with new feature updates or additions. During the design process, a child product model may be created and explored, and can be given up. In this case, backtracking takes place and the designer goes back to the parent product model to explore new opportunities.

The hierarchical tree structure of product models can be used to find procedure features. Procedure features can be found by checking the emergence of and distribution pattern of design operations in all design paths based on design principle knowledge. A further explanation about the identification of procedure features can be found in Section 4.4. An example of the hierarchical tree structure of the product model is shown in Figure 4. The structure also shows the design path that a designer went through during his design process. In Figure 4 we see that the product model P_4 is recognized as a child of P_2 (not P_3) by comparing their parametric features, although P_4 arises after P_3 chronologically. The designer might have given up on developing P_3 and gone back to modify P_2.

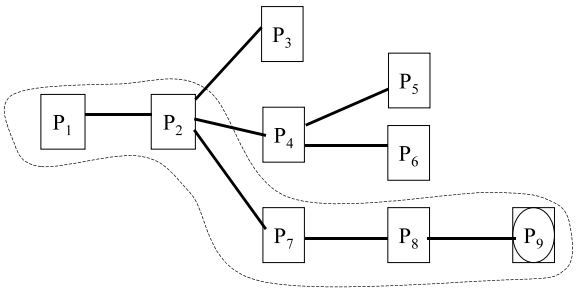


Fig. 4. The hierarchical tree structure of product models: $\mathbf{P_1}$, initial product model created by a designer; $\mathbf{P_9}$, final product model that meets the designer's requirements; $\mathbf{P_2}$–$\mathbf{P_8}$, derived product models generated through the design process; (—) parent and child relationship between product models; (• • •) the main path that leads from the origin to the finally selected product model.

4. GEDP APPROACH

4.1. GEDP for acquiring *know-how*

Capturing design operations and their relationships with product models from design events is a challenging task. The key issue is how to correctly cluster events into operations, because there are an enormous number of possible combinations of events.

From empirical studies we found that a design process is like a *story* that continues until the design goal (i.e., the final product model that satisfies all requirements) is achieved. Figuratively speaking, an *event* corresponds to a word, for example, a noun, a verb, an adjective, and so on. An *operation* corresponds to a sentence and a *product model* is equal to a paragraph. We define and apply rules of "how events emerge and make sense in an operation" in the same way that a natural language has its grammar. Our grammar works as ground rules for detecting sentences and pauses of paragraphs by searching for key words. However, it may not work properly when various punctuations and multiple interpretations are detected. In these cases, we need a new way to find the most reasonable punctuation and interpretation.

We developed the GEDP for achieving the most reasonable punctuation and interpretation. GEDP is a method that combines the *grammar approach* and EDP. First the grammar is defined and applied to the whole event log to extract clusters of events, and then EDP is applied to certain selected areas of the event log as a result of grammar approach to extract more detailed clusters if needed. The GEDP is schematically illustrated in Figure 5. The GEDP enables us to generate operations at Operation-Layer from the events captured at Event-Layer. It is the base of the data integration component in our knowledge-capturing module.

The GEDP approach has several advantages. First, it is a reasonable and understandable method for humans. Because this method adopts an analogy to a generic problem solving of human, for example, grammar rules and templates, it leads to understandable results. Second, this method has flexibility on template matching, because using EDP makes it possible to detect not only the same sequence as the template but also approximately similar ones. Third, this method is widely applicable. GEDP can be applied to general design problems, although the grammar has to be modified to fit the design context.

The algorithms of the grammar approach and EDP are described in the following two subsections.

4.2. Algorithms of grammar approach

Before starting the knowledge capturing, all events in the Event-Layer are enumerated and identified in the monitor-

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

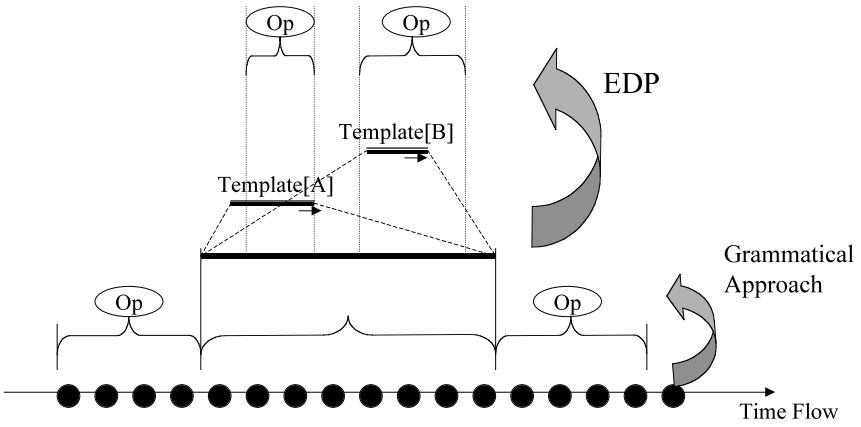


Fig. 5. The grammar and extended dynamic programming approach (GEDP): (●) a design event; Op, a design operation.

ing component. The contents of the events are dependent on the design context. The event-log data are stored based on the event ID number. Before applying the grammar approach, events are classified into several groups based on their contents. This classification is similar to the classification of words into verb, noun, adjectives, and so on in a natural language. The unique grammar rules and templates are assigned to each group. A rule template defines a one to one relationship between a typical event sequence and an operation. Examples of grammar rules and templates are shown in Tables 1 and 2, respectively.

The algorithm of the grammar approach is illustrated in Figure 6. As indicated in the figure, the main function of the grammar approach is CLUSTERING_BY_GRAMMAR, which receives the whole event sequence as its input and returns the whole operation sequence as its output. First of all, the whole event sequence is put in the variable *events*. The UPDATE-CURRENT function puts the head event of *events* into the variable *current*. The DISCRIMINATION-GROUP function investigates to which group *current* belongs and puts the value in S_G. The SEARCH-HARMONIOUS-EVENTS function searches *events* from the top sequentially to get the event sequence that goes harmoniously with *current*. The rules about the harmony are written as grammar rules and they depend on the group. This function also puts the resulting event sequence in *memory*. The CHOOSE-MANIPULATOR function selects *manipulator* according to S_G. The variable *manipulator* denotes how to make a cluster based on the kind of group. The GRAMMAR-MANIPULATION function applies *manipulator* to *memory* and obtains a corresponding operation ID in the variable *operation*. This *operation* is added to *operation-list*, and *events* are updated where UPDATE-EVENTS function omits *memory* from *events*. The actions from UPDATE-CURRENT to UPDATE-EVENTS are repeated until *events* is empty. Finally, a set of operations can be acquired.

Table 1. *Examples of grammar rules*

	Meaning of Group	Event ID	Grammar Rule
Group 1	Direct input of some parameters	21, 31, . . .	A member of Group 1 can build up an Operation on its own or one that is modified by members of Group 2. There is no limitation on the number of modifiers.
Group 2	Reference (to see a document)	101, 103, . . .	A member of Group 2 can modify another event that belongs to Groups 1, 3, 4, or 5. A member of Group 2 cannot form an Operation by itself.
Group 3	Increase of a parameter	66, 68, . . .	A member of Group 3 can build up an Operation on its own or by assembling two or more. Moreover, this can be modified by members of Group 2. There is no limitation on the number of modifiers.
Group 4	Decrease of a parameter	65, 67, . . .	A member of Group 4 can build up an Operation on its own or by assembling two or more. Moreover, this can be modified by members of Group 2. There is no limitation on the number of modifiers.
Group 5	Change of a location	91, 92, . . .	A member of Group 5 can build up an Operation on its own or by assembling two or more. Moreover, this can be modified by members of Group 2. There is no limitation on the number of modifiers.
Group 6	Cue to start something	1, 60, . . .	A member of Group 6 can build up an Operation individually. The element of this group is never modified.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

Table 2. *Examples of rule templates*

Template	Grammar Rules	Operation	Contents of Operation
T1	If it includes E21, then Op1	Op1	Determine the number of gear teeth.
T10	If it makes a combination out of {E66, E68, E70 ($n >= 1$)}, Then Op10	Op10	Prolong Y-length of shafts.
T15	If it makes a combination out of {E91, E92, E93, E94, E95, E96, E97, E98 ($n >= 1$)}, Then Op15	Op15	Apply EDP, because various operations can exist in Op15.

4.3. Algorithms of EDP

Originally, dynamic programming (DP) was an approach to solving sequential decision problems that was developed by Richard Bellman in 1957 (Bellman, 1957). The simplest DP context involves an *n*-step decision-making problem, where the states reached after *n* steps are considered terminal states and have known utilities. The main concept of DP is that choosing the best state (i.e., the state with the highest utility) in each step leads to the optimum final state. Recently, DP was applied to solving various pattern matching problems, for example, speech recognition (Sakoe & Chiba, 1990), image recognition (Chikada et al., 1999), and bioinformatics (Krogh et al., 1994). EDP is a method based on DP theory and modified to be suitable for this design problem.

When more than one prospective solution exists for a pattern matching, EDP is able to detect the most reasonable solution. Here, the most reasonable solution means the solution that has the highest value for a user based on a given objective criterion. Moreover, EDP enables us to detect not only the same sequence as the template but also approximately similar ones. It allows sequential errors to some extent, for example, deletion, insertion, or exchange of elements.

In EDP, templates, which are typical target fragments of elements and, in this case, sequences of design events, should be prepared before starting the analysis. Every template needs two elements. One is the definition of a one to one relationship between a typical event sequence to grasp and an operation ID, and the other is the value score of the templates. A user, such as a designer, can determine the value score based on his or her subjective evaluation of a design problem. A higher score means it is more important. Figure 7 describes the algorithm of EDP.

The algorithm shown in Figure 7 works as follows. The main function of EDP is CLUSTERING_BY_ EDP, which receives the delivered event sequence and the set of templates as its input and returns the corresponding operation sequence as its output. First of all, the template that has the highest *value score* of all templates is chosen and put into the variable *probe* by the UPDATE-HIGHEST-VALUE-

```
function CLUSTERING_BY_GRAMMAR(events)  return set of operations
    inputs: events ; sequence of events
    static: templates ; set of operation-templates, each template has event-operation rule
    variables: current ; a specific event
               S_G ; a group of the event
               memory ; a memory about the event sequence
               manipulator ; a grammar manipulator
               operation ; an operation risen from the event sequence
               operation-list ; a list of operation, initially null

    loop do until events is empty
        current <-- UPDATE-CURRENT(events)
        S_G <-- DISCRIMINATION-GROUP(current)
        memory <-- SEARCH-HARMONIOUS-EVENTS(S_G , events)
        manipulator <-- CHOOSE-MANIPULATOR(S_G)
        operation <-- GRAMMAR- MANIPULATION(manipulator, memory)
        operation-list <-- add operation
        events <-- UPDATE-EVENTS(events, memory)
    end
    return operation-list
```

Fig. 6. The algorithm of the grammar approach.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

```
function CLUSTERING_BY_ EDP(events, templates)   return set of operations
    inputs: events ; sequence of events
            templates ; set of operation-templates, each template has two elements,
                    event-operation rule and its value-score
    variables: probe ; a specific template
               extractant ; an extracted sequence of events
               operation ; an operation risen from the event sequence
               operation-list ; a list of operation

    loop do until templates or events is empty
        probe <-- UPDATE-HIGHEST-VALUE-TEMPLATE(templates)
        extractant <-- EDP-MANIPULATION(events, probe)
        templates <-- UPDATE-TEMPLATES(templates, probe)
        if extractant is not null then
                operation <-- TRANSLATE(extractant, probe)
                operation-list <-- add operation
                events <-- UPDATE-EVENTS(events, extractant)
    end
    return operation-list
```

Fig. 7. The algorithm of EDP.

TEMPLATE function. Then the EDP-MANIPULATION function, which is discussed below, extracts the event sequence whose mismatching score is less than the threshold by applying *probe* to *events* and puts the resulting event sequence into the variable *extractant*. The UPDATE-TEMPLATES function omits *probe* from *templates*. If *extractant* is not null, then the TRANSLATE function obtains a corresponding operation ID in *operation* by referring *probe* and *extractant*; then this *operation* is added to *operation-list*, and *events* are updated where the UPDATE-EVENTS function omits *extractant* from *events*. The above-mentioned actions from UPDATE-HIGHEST-VALUE-TEMPLATE to UPDATE-EVENTS are repeated until *events* or *templates* is empty. Finally, a set of operations can be acquired.

The algorithm of the EDP-MANIPULATION function is now described. The formula, $\boldsymbol{S}$ = EDP-MANIPULATION ($\boldsymbol{T}$, $\boldsymbol{P}$), indicates that the sequence $\boldsymbol{S}$, which is approximately similar to the sequence $\boldsymbol{P}$, is extracted out of the sequence $\boldsymbol{T}$. Because the elements (i.e., design events) in a sequence are arranged in a time-series order from left to right, their order is retained.

The definitions are as follows:

- $\boldsymbol{T} = [\mathrm{T}_0, \mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_i, \ldots, \mathrm{T}_I]$, $\boldsymbol{P} = [\mathrm{P}_0, \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_j, \ldots, \mathrm{P}_J]$. Here $\boldsymbol{T}$ and $\boldsymbol{P}$ symbolize sequential strings; T_i and P_j symbolize an element of each sequence; the bracket means elements in it are ranged in a time-series order from left to right; and I and J are the number of elements of $\boldsymbol{T}$ and $\boldsymbol{P}$, respectively.
- The concept of Levenshtein distance (Graham, 1994) is introduced, which is widely used in pattern matching problems. The Levenshtein distance shows the distance between one string (i.e., an array of characters) and another string and represents the minimum cost for one string changing into another one through insertion, deletion, or exchange.
- $d(i,j)$ represents the distance between T_i and P_j.
- $D(T,S)$ represents the distance between $\boldsymbol{T}$ and $\boldsymbol{P}$.
- $g(i,j)$ represents the distance between the string $[\mathrm{T}_0, \ldots, \mathrm{T}_i]$ and the string $[\mathrm{P}_0, \ldots, \mathrm{P}_j]$.
- $g(I,J) = D(T,S)$
- $B(i,j)$ represents the beginning point of the subset of $\boldsymbol{T}$ that is extracted from $\boldsymbol{T}$ to maximally coincide with $\boldsymbol{P}$. In other words, the string $[\mathrm{T}_{B(i,j)}, \ldots, \mathrm{T}_i]$ matches the string $[\mathrm{P}_0, \ldots, \mathrm{P}_j]$ the best.
- p is the cost of exchange between two events.
- q is the cost of deletion of an event.
- r is the cost of insertion of an event.

The manipulation proceeds according to the following:

1) Initialization
   $g(0,0) = 0$, $B(i,j) = 0$
   For $i = 1$ to $i = I$
      $g(i,0) = 0$, $B(i,0) = i$
   For $j = 1$ to $j = J$
      $g(0,j) = g(0,j-1) + r$, $B(0,j) = 0$
2) Iteration
   For $i = 1$ to $i = I$
     For $j = 1$ to $j = J$

$$g(i,j) = \min \begin{cases} g(i-1,j) + q & (1) \\ g(i-1,j-1) + d(i,j) & (2) \\ g(i,j-1) + r & (3) \end{cases}$$

where, If $\mathrm{T}_i = \mathrm{P}_j$, Then $d(i,j) = 0$
Else $d(i,j) = p$

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

$$\text{If } (1) = (2) = (3), \text{ Then } B(i,j) = \min \begin{cases} B(i-1,j) \\ B(i-1,j-1) \\ B(i,j-1) \end{cases}$$

$$\text{Else } (1) = (2) < (3), \text{ Then } B(i,j) = \min \begin{cases} B(i-1,j) \\ B(i-1,j-1) \end{cases}$$

$$\text{Else } (2) = (3) < (1), \text{ Then } B(i,j) = \min \begin{cases} B(i-1,j-1) \\ B(i,j-1) \end{cases}$$

$$\text{Else } (3) = (1) < (2), \text{ Then } B(i,j) = \min \begin{cases} B(i-1,j) \\ B(i,j-1) \end{cases}$$

$$\text{Else } (1) < (2) \text{ and } (1) < (3), \text{ Then } B(i,j) = B(i-1,j)$$

$$\text{Else } (2) < (1) \text{ and } (2) < (3), \text{ Then } B(i,j) = B(i-1,j-1)$$

$$\text{Else } (3) < (1) \text{ and } (3) < (2), \text{ Then } B(i,j) = B(i,j-1)$$

3) Finalization
   For $i = 1$ to $i = I$
   If $g(i,J) \leq$ *Threshold*, Then $\boldsymbol{S}_c = [\mathrm{T}_{B(i,j)}, \ldots, \mathrm{T}_i]$
   $\boldsymbol{S}_c$ represents a candidate of a target design event sequence. When more than one candidate $\boldsymbol{S}_c$ are found overlapping one another, the longest should be selected as the final solution sequence $\boldsymbol{S}$ out of them. We allow plural $\boldsymbol{S}$ to exist unless their scopes interfere with each other.

4.4. Acquiring *know-how* knowledge

In this research we also attempt to capture procedure features from the design procedure information obtained by applying GEDP. To achieve this goal, we combine the parametric design view with the GEDP and introduce several definitions.

First of all, we introduce the concept of *product model core* to represent the current status of a product model under development. A product model core is an essential part of its corresponding product model and is defined by a key subset of the parameters of the product model. Updating the parameter values of the product model core implies significant progress in product model development.

Next we require that the product models be created and developed through the following steps: (1) the *initial product model* is generated from an empty or incomplete product structure (i.e., a product model with parameters that have null values) by assigning nonnull values to the design parameters, and (2) a *derived product model* is created when a major design change (i.e., the design change that causes changes in the product model core) is made on an initial or derived product model. Subsequent design changes can be made to a newly derived product model to accommodate the major design change. Based on these definitions, one can determine when a product model is formed by monitoring the changes of the parameters of the product model core.

Besides being able to distinguish between different derived product models, we also need to know the *relationship* between the models, for example, how close or far away they are to each other. To evaluate the relationship between product models, we introduce a concept of *virtual distance* between product models. We define the distance between two product models as represented by the summation of a set of binary values resulting from the comparison of the parameters of the product model core. If the values of the same parameter of the two product models are different, then the binary value is 1, otherwise 0. When a new derived product model is formed, the distances between the product model and all other existing product models are calculated. After that the product model that has the closest distance is selected as the parent of the newly derived product model. In the product model tree structure, such as the one shown in Figure 8, a child product model is located and linked directly under a parent. If the closest distance is further than the predefined threshold, the product model is recognized to be independent of all other product models and have no parent. An independent derived product model with no parent is located as the root of a new tree, as shown in Figure 8.

Eventually, after performing GEDP and the analysis of the tree structure of product models, the following can be obtained: (1) relationships between product models as a time-series branching tree structure, (2) a list of the status of product models described by parameters of the corresponding product model cores, and (3) a list of design operations attached to individual product models in the occurrence order. These three items contains important and useful design procedure information.

In addition to the design procedure information, procedure features are also acquired through the following steps (see Fig. 9):

1. Find the main path that leads from the origin to the finally selected ideal product model. Treat other paths as branch paths.
2. Enumerate all the positions of design operations. Design operations on the main path are given numbers starting from 0 at the origin and increasing subsequently to the end of the path. Design operations on branch paths are assigned numbers starting with the number of the branching operation on the main path. This enumeration allows us to represent the depth of the tree structure and recognize the positions of occurrence of a specific type of operation in the design process. Figure 9 shows an example of the enumeration in which seven product models were generated and 22 design operations were performed. As shown in the figure, each design operation not only has its type identification (e.g., Op2 or Op12) but also its position numbers (e.g., 3 or 6) in the product model tree.
3. Find key indexes that represent emergence features of different types of operations in the design process. Instances of the key indexes include the *emergence*

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

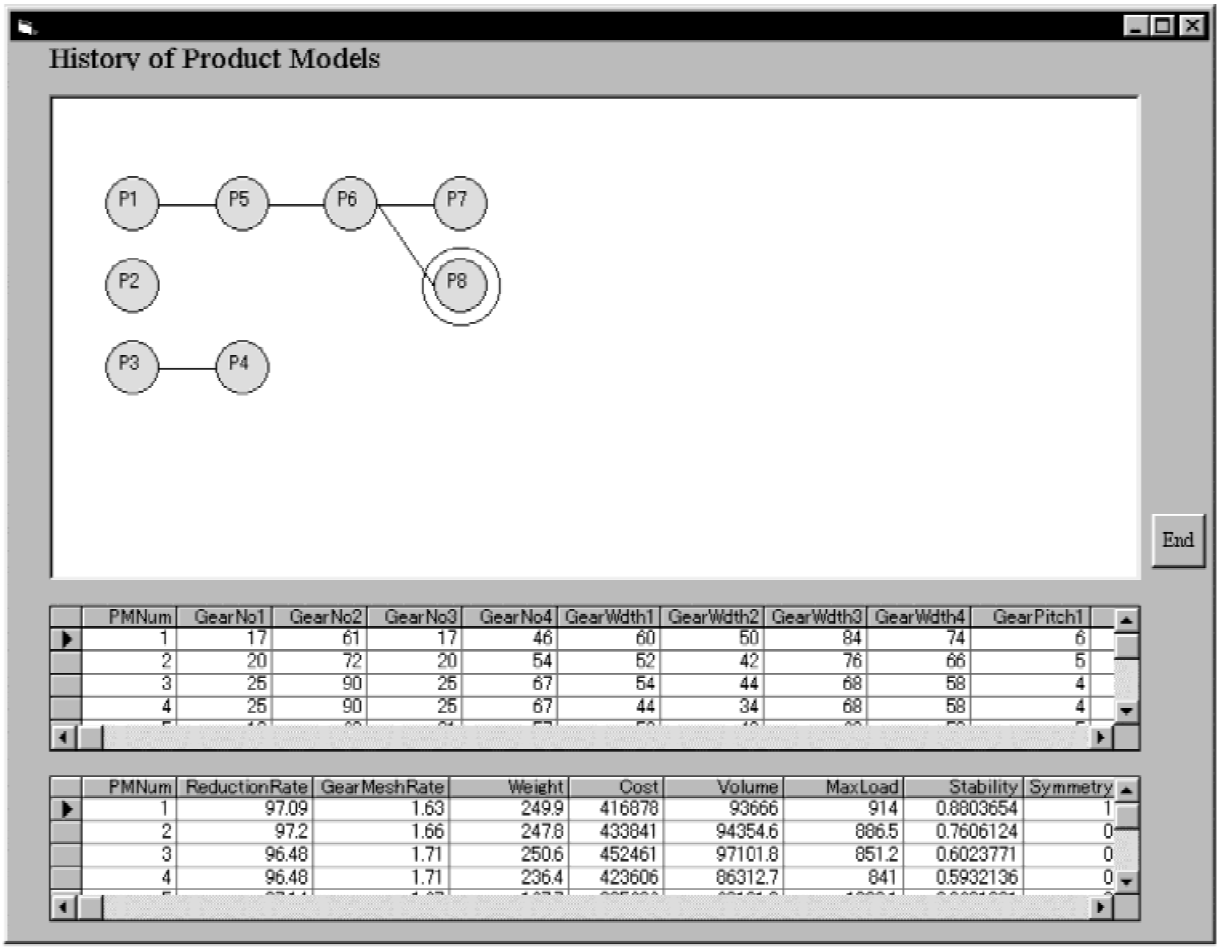


Fig. 8. An example of the hierarchical tree structure.

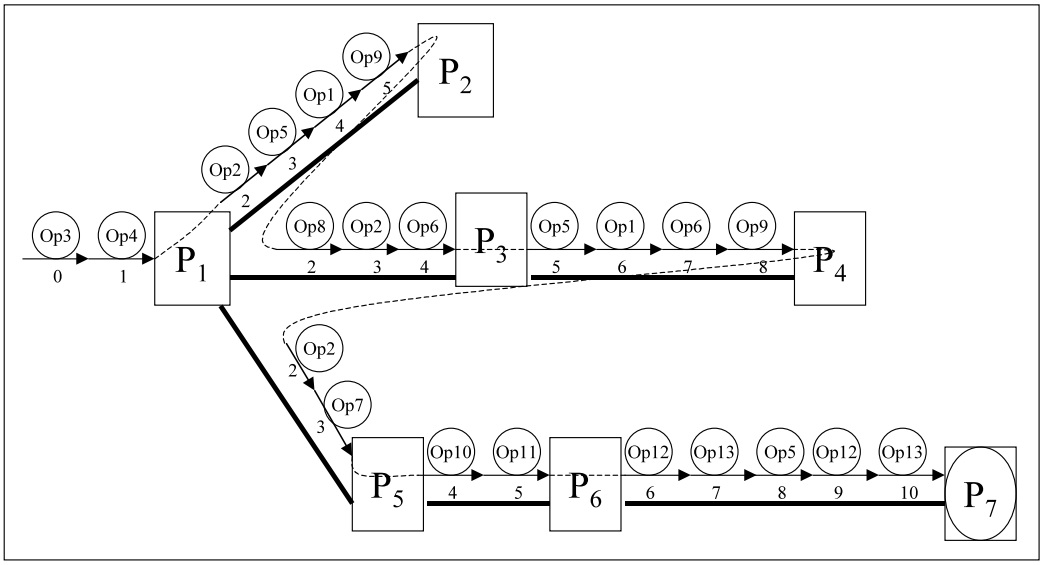


Fig. 9. The design procedure analysis for capturing procedure features: $\mathbf{P}_n$, product model, where P_1 is the initial product model and P_7 is the final product model; $\mathbf{P_1}$-$\mathbf{P_5}$-$\mathbf{P_6}$-$\mathbf{P_7}$, the main path that leads from the origin to the finally selected product model; (Opj) the design operation, where j indicates an identification of the contents of the design operation; ($\vec{i}$) the existence of a design operation, where i indicates the position number in the tree structure; (—) the parent and child relationship between product models.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

Table 3. *Example of key indexes of design operations*

Design operation ID	1	2	3	4	5	6	7	8	9	10	11	12	13
Emergence frequency	2	3	1	1	3	2	1	1	2	1	1	2	2
Average of position	5.0	2.3			5.3	5.5			6.5			7.5	8.5
Average of position in total process	0.45	0.21			0.48	0.50			0.59			0.68	0.77
Standard deviation of position	1.0	0.47			2.1	1.5			1.5			1.5	1.5

frequency, the *average of emergence positions*, and their *standard deviation*. The emergence features of the operations of the example Figure 9 are shown in Table 3. As shown in the table, Op2 occurred three times: between P_1 and P_2, P_1 and P_3, and P_1 and P_5. Its average emergence position is 2.3, and its standard deviation is 0.47.

4. Translate predefined design process meta knowledge into IF–THEN rules using the key indexes. Based on design principle knowledge, one can define a set of meta-level principles about design processes. For example, "Design operations which frequently appear in the early stage of the design process are related to a crucial change of product specifications" is a useful meta-level knowledge about the design process. To use this type of knowledge effectively, we encoded it using the key indexes described above. An example of such encoded rules is "IF an identified design operation has its emergence frequency $>= 3$, its average position < 0.5, and its standard deviation of the position < 1.0; THEN the emergence pattern of the design operation is a procedure feature of *early examination*."
5. Search for procedure features based on the key indexes based IF–THEN rules. Based on the IF–THEN rules described above, procedure features of the design process can be captured through a production system reasoning mechanism. For example, in Figure 9 and Table 3, only Op2 matches the above-mentioned IF–THEN rule. Therefore, design Op2 represents an *early examination* procedure feature of the design process.

5. CASE STUDY

5.1. Double-reduction gear system

Our proposed methods were evaluated in a case study on the design of a double-reduction gear system. The double-reduction gear system is composed of four gears, three shafts, bearings, and a case. Basically, the number of teeth in the gears determines the speed reduction rate. Because the power of the revolution makes the torque and the bending moment, the gears and shafts are designed to stand up to the force. We developed the CAD system called "Gear-CAD." Gear-CAD is an integrated design environment that allows designers to access all the information they need and to use the tools they need. Gear-CAD has all the domain knowledge about this double-reduction gear system, supports the design, and simultaneously records the entire designer's log, which contains all events he or she generated during the design process. Figure 10 shows an example of the Gear-CAD screens, and Figure 11 shows the structure of the gear design problem.

The requirements and conditions for the gear design problem follow.

Requirements:

1. All design components are determined in detail, that is, the size and position.
2. The required reduction ratio is 10:1.
3. Lighter, smaller, and cheaper is better on the assumption of using the equipment in outer space.

Conditions:

1. Spur gears that have teeth with a 20° pressure angle are utilized in this system.
2. The input power and speed of rotation are 10.0 kW and 500 rpm, respectively.

Our monitoring system including Gear-CAD and knowledge capturing system were developed in Windows 98 OS. The demo system was written in Visual Basic 6.0.

5.2. *Know-how* knowledge

A user who has enough knowledge on this problem designed the double-reduction gear system using Gear-CAD. From the design log, *know-how* knowledge was captured as follows.

5.2.1. Event

Gear-CAD stores a list of events captured during the design process. The list consists of the event ID and the associated action, for example, "Event-ID 5; See document No.2" and "Event-ID 150; Input the number of gear teeth of pinions." Events are recorded by the event ID and supplementary comment if needed. Table 4 shows an event log with a total of 472 events.

5.2.2. Operation

By using GEDP, 150 operations were captured from 472 events, as shown in Table 5. For example, the event se-

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

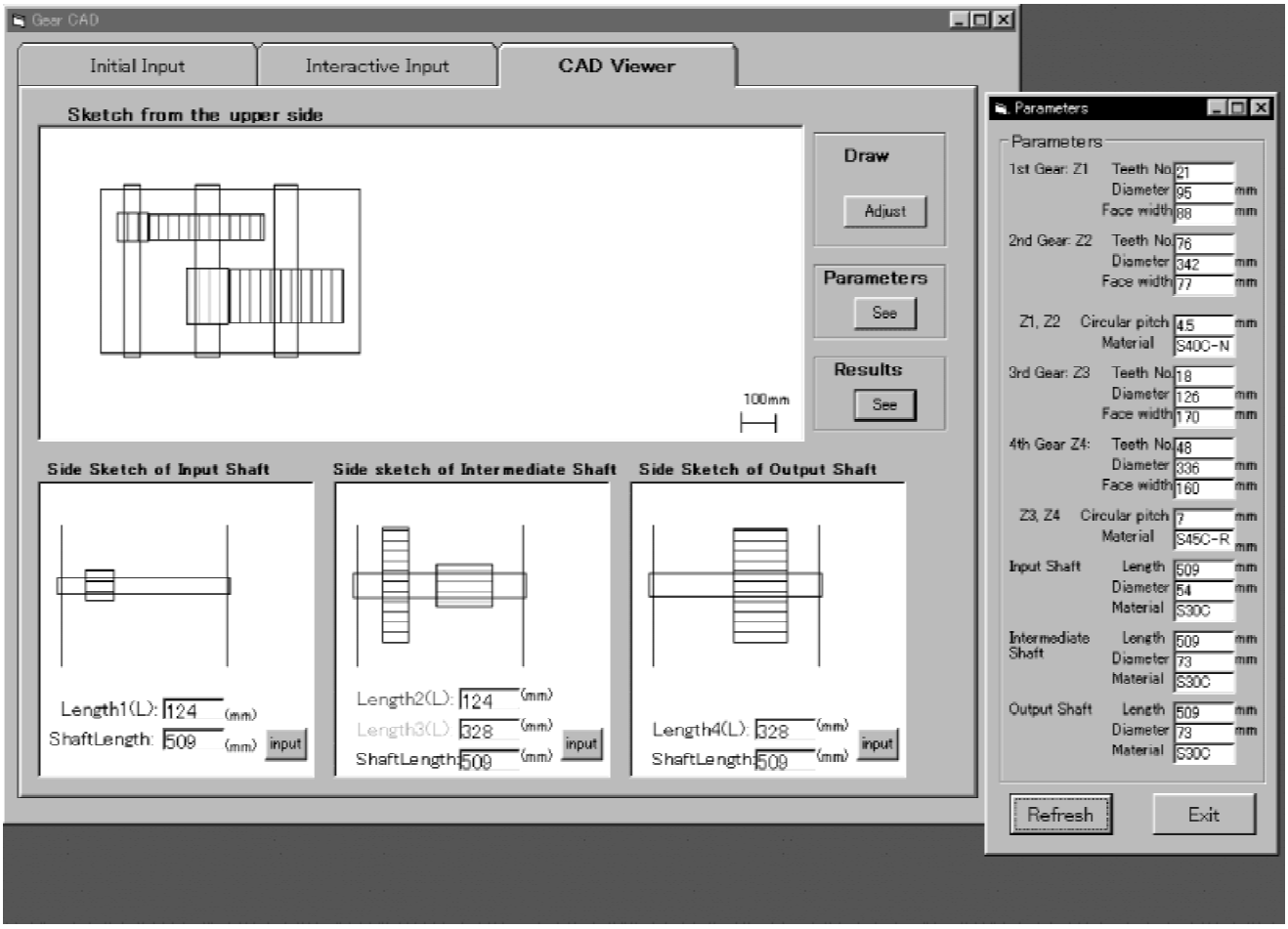


Fig. 10. An example of a gear-CAD screen.

quence from 35 to 43, $\boldsymbol{S}_0$ = [E93, E94, E96, E91, E93, E94, E96, E95, E97], was obtained as a meaningful sequence by the grammar approach; then $\boldsymbol{S}_1$ = [E93], $\boldsymbol{S}_2$ = [E94, E96], and $\boldsymbol{S}_3$ = [E91, E93, E94, E96, E95, E97] were captured by EDP as [Op155: Adjust Y-Positions of Z1 and Z2 Gears], [Op156: Adjust X-Positions of Z2 and Z3 Gears], and [Op155: Adjust Y-Positions of All Gears], respectively. In this case, the template $\boldsymbol{P}_1$ = [E91, E93] for $\boldsymbol{S}_1$, $\boldsymbol{P}_2$ = [E94,

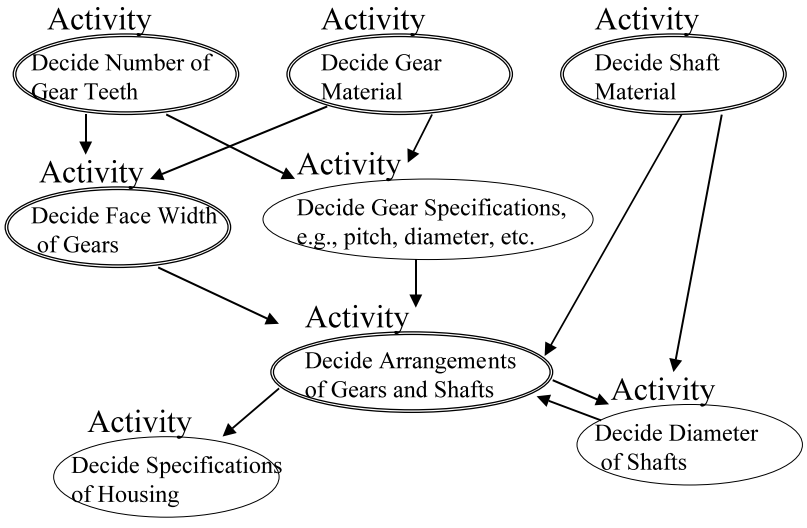


Fig. 11. The structure of the gear design problem. The input by humans is contained in the dark-edged circles and that calculate automatically is contained in the light-edged circles.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

Table 4. *Example of event log data*

No.	Event ID	Arguments	Time
1	1	#	2.1
2	103	#	2.1
3	104	#	3.6
4	123	#	3.6
5	124	#	5.2
6	5	#	5.7
7	107	#	5.9
8	108	#	11.4
9	21	Z(1): 17, Z(3): 17	11.9
10	30	#	12.7
11	129	#	12.9
12	31	GearMaterial(1):1, GearMaterial(2):1	17.3
13	130	#	17.3
14	129	#	17.7
15	125	#	17.7
16	126	#	21.7
17	12	#	37.8
18	41	Z1w:60, Z2w:50, Z3w:84, Z4w:74	40.2
19	50	#	42.1
20	131	#	42.1
21	51	ShaftMat(1):1, ShaftMat(2):1, ShaftMat(3):1	46.1
22	132	#	46.1
23	131	#	46.4
24	127	#	46.4
25	128	#	47.5
26	60	#	47.5
27	91	Move Z1y:−22.5	50.6
28	92	F = 1, Move Z1x:465	50.6
29	65	Shorten S1:−274	50.6
30	93	Move Z2y:−22.5	51.8
31	94	F = 1, Move Z2x:−3	51.8
32	96	F = 1, Move Z3x:−3	51.8
33	98	F = 1, Move Z4x:−3	51.8
		⋮	
468	200	#	907.0
469	87	#	907.1
470	138	#	907.1
471	137	#	907.2
472	201	FinalDgn: 8	913.0

E96] for $\boldsymbol{S}_2$, and $\boldsymbol{P}_3$ = [E91, E93, E95, E97] for $\boldsymbol{S}_3$ were applied. Because the longer template had the higher value score, $\boldsymbol{S}_3$ was found at first, and then $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ were found from the rest. It turned out that the GEDP was flexible against a deletion, an insertion, and an exchange of elements by comparing templates and the obtained sequences.

5.2.3. Product model

From the viewpoint of parametric design, eight product models were obtained. The parent–child relationships among product models are shown in Figure 8, where the parameters for measuring the relationship were the number of gear teeth, the gear materials, the face width of the gears, and the shaft materials. Finally, product model 8 was chosen as the best.

5.2.4. Knowledge

According to the relationships among product models, the path PM1–PM5–PM6–PM8 led to the ideal solution as a main path. By comparing the operations captured by GEDP and the history of the product model, the *know-how* (i.e., procedure information) in Table 6 was acquired.

The product models that were obtained and the main parameters are shown in Table 7.

The procedure features were found by a search using the design process meta-knowledge rules described in Section 4.4. Examples of actually obtained procedure features are presented.

F1. The determination on the gear properties might have priority, because the gear teeth numbers, gear materials, and gear positions were determined in the early stage of design by the time PM5 was formed.

F2. The determination on shaft materials might be a lightweighted activity, because it still changed frequently in the latter stage (from PM5 to PM8).

F3. The Y-Position of the gears might be important to finalize the design, because “Adjust Y-Positions of Z3, Z4 Gears” was repeated at three times in procedures for PM8.

5.3. Discussion

Our GEDP method based on the three-layer design process model was developed to extract *know-how* knowledge without disrupting a normal design process. Although the experiment discussed was only an example to evaluate this method, the results from the application of the GEDP method to the double-reduction gear system demonstrated the effectiveness of the method.

Initially, design procedures were acquired by this bottom-up method. GEDP had several advantages. First, the captured operations shown in Table 6 were easy to understand for a user and the stream of operations represented sufficient content of a design context. Because 150 operations were captured from 472 events, the average abridging rate is 3.1. In addition to the trial mentioned above, several other experiments were carried out and the rate was similar: 43 operations from 139 events (abridging rate of 3.0) and 92 operations from 305 events (abridging rate of 3.3). Although the rate seems slightly small, it depends on the CAD system. Because Gear-CAD was developed to specialize in the double-reduction gear system problem, an event itself tended to represent specialized contents, for example, “Input the face width of gear.” If this method is applied to a general CAD system, events represent more general actions and the abridging rate must be increased. Second, GEDP showed flexibility in pattern recognition. As mentioned in Section 5.2.2, GEDP allowed the deletion, insertion, or exchange of elements in a target sequence to some extent. Third, this method had the potential of wide application. In both the grammar ap-

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

Table 5. *Example of acquisition of operations from event log*

Event No.	Event ID	Ope No	Ope-ID	Arguments	PM_N
1	1 →	1	16	Start CAD	1
2	103				
3	104				
4	123				
5	124	2	1	M(103), M(107), Z(1):17, Z(3):17	1
6	5				
7	107				
8	108				
9	21				
10	30				
11	129	3	2	GearMaterial(1):1, GearMaterial(2):1	1
12	31				
13	130				
14	129				
15	125	4	3	Z1w:60, Z2w:50, Z3w:84, Z4w:74	1
16	126				
17	12				
18	41				
19	50				
20	131	5	4	ShaftMat(1):1, ShaftMat(2):1, ShaftMat(3):1	1
21	51				
22	132				
23	131				
24	127	6	17	Start Drawing	1
25	128				
26	60				
27	91	7	151	Move Z1 Formation = 1	1
28	92				
29	65 →	8	11	Shorten_S1:274	1
30	93				
31	94	9	156	Adjust X-Positions of All Gears	1
32	96				
33	98				
34	69 →	10	13	Shorten_S3:182.5	1
35	93 →	11	155	Adjust Y-Positions of Z1,Z2 Gears	1
36	94	12	156	Adjust X-Positions of Z2,Z3 Gears	1
37	96				
38	91				
39	93				
40	94	13	155	Adjust Y-Positions of All Gears	1
41	96				
42	95				
43	97				
			⋮		
472	201 →	150	6	FinalDgn: 8	8

proach and EDP, renewal of each rule makes it applicable to generic design problems. It is anticipated that in the general case the grammar component becomes simple and the EDP component is extended with many templates.

Further, procedure features were obtained. Three features identified as F1–F3 could never be acquired by means of a mediocre parameter analysis about product models, because they require the analysis of relationships between the tree structure and design operations generated by applying the GEDP. The procedure knowledge such as the examples in Table 6 contributed largely toward achieving the procedure features.

Know-how knowledge can be used in practical design. In practice, it is common for a group of designers to iteratively design different versions of the same artifacts such as cars and electrical appliances. Under these conditions, understanding other designers' *know-how* knowledge is important for managing the effectiveness and efficiency of collaboration. Furthermore, knowing expert designers' methods of design can help knowledge transfer. For instance, knowing the *know-how* of an expert or veteran designer may provide other designers with insights about designing and improve their design process. This knowledge transfer is important, especially when the new design-

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

Table 6. *Acquired know-how* knowledge

Operations for PM 1	Operations for PM 5	Operations for PM 6	Operations for PM 8
Start CAD	Z(1):19, Z(3):21	ShaftMat(1:3, ShaftMat(2:3, ShaftMat(3):3	Z(1):19, Z(3):21
M(103), M(107), Z(1):17, Z(3):17	GearMaterial(1):3, GearMaterial(2):6	Z1w:52, Z2w:42, Z3w:60, Z4w:50	GearMaterial(1):3, GearMaterial(2):6
GearMaterial(1):1, GearMaterial(2):1	Z1w:52, Z2w:42, Z3w:60, Z4w:50	Start Drawing	Z1w:52, Z2w:42, Z3w:60, Z4w:50
Z1w:60, Z2w:50, Z3w:84, Z4w:74	ShaftMat(1):1, ShaftMat(2):1, ShaftMat(3):1	Adjust Y-Positions of Z1,Z2 Gears	ShaftMat(1):1, ShaftMat(2):1, ShaftMat(3):1
ShaftMat(1):1, ShaftMat(2):1, ShaftMat(3):1	Start Drawing	Adjust X-Positions of Z2,Z3 Gears	Start Drawing
Start Drawing	Move Z1 Formation = 1	Adjust Y-Positions of Z3,Z4 Gears	Adjust X-Positions of All Gears
Move Z1 Formation = 1	Shorten_S1:250	Adjust X-Positions of All Gears	Adjust Y-Positions of Z3,Z4 Gears
Shorten_S1:274	Move Z1 Formation = 0	Move Z4 Formation = 0	Move Z4 Formation = 0
Adjust X-Positions of All Gears	Prolong_S1:250	Shorten_S1:296,25, Shorten_S2:292.5	Shorten_S1:333.75, Shorten_S2:322.5
Shorten_S3:182.5	Adjust X-Positions of All Gears	Prolong_S1:3.75	Prolong_S1:11.25
Adjust Y-Positions of Z1,Z2 Gears	Move Z1 Formation = 0	Shorten_S1:11.25, Shorten_S3:292.5	Shorten_S3:322.5
Adjust X-Positions of Z2,Z3 Gears	Adjust Y-Positions of All Gears	Prolong_S1:11.25	See the Parameters
Adjust Y-Positions of All Gears	Adjust Y-Positions of Z1,Z2 Gears	Prolong_S1:18.75	See the PMs
Prolong_S3:78.75	Adjust X-Positions of All Gears	Adjust Y-Positions of Z3,Z4 Gears	See the previous PMs
Shorten_S1:157.5	Shorten_S1:247.5	Shorten_S2:18.75	Adjust Y-Positions of Z3,Z4 Gears
Adjust X-Positions of Z2,Z3 Gears	Prolong_S1:247.5	Shorten_S3:18.75	Adjust Y-Positions of Z3,Z4 Gears
Adjust Y-Positions of Z3,Z4 Gears	Shorten_S1:292.5, Shorten_S2:281.25, Shorten_S3:285	See the Parameters	Adjust Y-Positions of Z3,Z4 Gears
Adjust X-Positions of All Gears	Prolong_S1:11.25, Prolong_S3:3.75	See the PMs	Shorten_S1:22.5, Shorten_S2:15
Move Z4 Formulation = 1	See the Parameters	See the previous PMs	Prolong_S1:7.5
Move Z1 Formation = 1	See the Parameters		Shorten_S3:15
Prolong_S1:26.25			See the PMs
Adjust X-Positions of All Gears			See the previous PMs
Shorten_S1:8.75, Shorten_S2:281.25, Shorten_S3:281.25			See the previous PMs
See the Parameters			
See the PMs			

Table 7. *Obtained product models*

	Value of parameters														
Product Model No.	Teeth No.	Width (cm)	Pitch (cm)	Material	Length A (cm)	Length B (cm)	Position		Length (cm)	Diameter (cm)	Material		Length X (cm)	Length Y (cm)	Length Z (cm)
1															
Gear No. 1	17	60.0	6.0	1	40.0	45.0	1	Shaft No. 1	105.0	4.6	1	Case	744.0	218.0	541.0
Gear No. 2	61	50.0	6.0	1	40.0	177.8	—	Shaft No. 2	238.0	6.2	1				
Gear No. 3	17	84.0	9.0	1	160.8	57.0	—	Shaft No. 3	134.0	7.3	1				
Gear No. 4	46	74.0	9.0		57.0	57.0	1								
2															
Gear No. 1	20	52.0	5.0	2	54.8	128.3	0	Shaft No. 1	203.0	4.5	2	Case	892.0	183.0	532.0
Gear No. 2	72	42.0	5.0	2	54.8	128.3	—	Shaft No. 2	203.0	5.9	2				
Gear No. 3	20	76.0	8.0	2	126.3	56.8	—	Shaft No. 3	203.0	6.9	2				
Gear No. 4	54	66.0	8.0	2	126.3	56.8	0								
3															
Gear No. 1	25	54.0	4.0	3	40.8	43.3	1	Shaft No. 1	104.0	4.2	3	Case	837.0	188.0	569.0
Gear No. 2	90	44.0	4.0	3	40.8	147.5	—	Shaft No. 2	208.0	5.6	3				
Gear No. 3	25	68.0	7.0	3	143.0	45.3	—	Shaft No. 3	208.0	6.7	3				
Gear No. 4	67	58.0	7.0	3	143.0	45.3	1								
4															
Gear No. 1	25	44.0	4.0	3	35.8	29.5	1	Shaft No. 1	85.0	4.2	3	Case	837.0	166.0	569.0
Gear No. 2	90	34.0	4.0	3	35.8	130.0	—	Shaft No. 2	186.0	5.6	3				
Gear No. 3	25	68.0	7.0	3	124.3	41.5	—	Shaft No. 3	186.0	6.7	3				
Gear No. 4	67	58.0	7.0	3	124.3	41.5	1								
5															
Gear No. 1	19	52.0	5.0	3	39.8	146.0	0	Shaft No. 1	206.0	4.7	1	Case	770.0	186.0	442.0
Gear No. 2	68	42.0	5.0	3	39.8	146.0	—	Shaft No. 2	206.0	6.4	1				
Gear No. 3	21	60.0	6.0	6	122.0	63.8	—	Shaft No. 3	206.0	7.4	1				
Gear No. 4	57	50.0	6.0	6	122.0	63.8	0								
6															
Gear No. 1	19	52.0	5.0	3	39.8	116.0	0	Shaft No. 1	176.0	4.3	3	Case	770.0	156.0	442.0
Gear No. 2	68	42.0	5.0	3	39.8	116.0	—	Shaft No. 2	176.0	5.7	3				
Gear No. 3	21	60.0	6.0	6	114.5	41.3	—	Shaft No. 3	176.0	6.7	3				
Gear No. 4	57	50.0	6.0	6	114.5	41.3	0								
7															
Gear No. 1	19	52.0	5.0	3	32.3	108.3	0	Shaft No. 1	160.0	4.4	2	Case	769.0	141.0	441.0
Gear No. 2	68	42.0	5.0	3	32.3	108.3	—	Shaft No. 2	160.0	5.8	2				
Gear No. 3	23	56.0	5.5	6	97.5	43.0	—	Shaft No. 3	160.0	6.9	2				
Gear No. 4	62	46.0	5.5	6	97.5	43.0	0								
8															
Gear No. 1	19	52.0	5.0	3	32.3	97.3	0	Shaft No. 1	150.0	4.4	2	Case	770.0	130.0	442.0
Gear No. 2	68	42.0	5.0	3	32.3	97.3	—	Shaft No. 2	150.0	5.8	2				
Gear No. 3	21	60.0	6.0	6	88.3	41.3	—	Shaft No. 3	150.0	6.9	2				
Gear No. 4	57	50.0	6.0	6	88.3	41.3	0								

ers deal with the same or similar design tasks as the expert designer.

On the other hand, there are some limitations of our method. The first one relates to the types of design problems to which our method can be applied and the requirements that need to be imposed on the CAD systems. Generally, we need to develop specific rule sets used in GEDP for specific types of design problems. More rules will be needed to increase the applicability of GEDP. At the same time, the number of rules that need to be developed and how easily they can be developed depend on what events can be captured by the CAD system being used. Our approach requires that individual components and their parameters be recognizable and monitorable by the CAD system. Although the domain-specific rules would be sufficient for our method to operate, design task specific rules will be needed to capture more useful knowledge. In general, parametric design and routine design are most suitable for our method.

The second limitation is that the obtained *know-how* knowledge is usually domain specific and context specific. For example, we cannot apply *know-how* for a gear system as it is to a problem of designing a robot arm. To further generalize the *know-how* knowledge, we plan to deal with the problem of capturing *know-why* knowledge.

6. RELATED WORK

Recently knowledge management are the key words in enterprises and business organizations to create value from an organization's intangible assets (Davenport, 1998; Liebowitz, 1999). Generally speaking, knowledge management consists of knowledge capturing, securing, retrieving, and distributing steps, and a great deal of research on each step has been executed. We focus on the engineering and technology aspects of knowledge management and especially address the issue of knowledge capturing.

While many researchers in engineering and artificial intelligence focused their knowledge capturing research on documented domain knowledge (Bradshaw et al., 1997), research on capturing *know-how* knowledge received little attention. That is because *know-how* knowledge seems domain specific and there is much difficulty in formalizing how to capture generic *know-how* knowledge. Despite the difficulty, some researchers point its importance out and struggle to capture and manage it; for example, Knowledge Infrastructure for Collaborative and Agent-Based Design was proposed for the agent system to use procedure knowledge and process knowledge in collaborative design (Jin & Zhou, 1999; Jin et al., 1999). However, the general way to capture *know-how* knowledge remains a research topic.

On the other hand, *design rationale* itself was studied from different point of views (Moran & Carroll, 1996). There are three major models: argumentation-based design rationale, action-based design rationale, and model-based design rationale. In the first approach the rationale is represented as a set of arguments (pros and cons) attached to issues, and the issues are interconnected. The Issue-Based Information System (IBIS), developed by Rittel (Kunz & Rittel, 1970), is an example of such methodologies. Various implementations of the IBIS concept were developed, for example, gIBIS (Conklin & Begeman, 1988), PHI (McCall, 1991), and DRL (Lee, 1990). However, the argumentation-based approach focused only on the design rationale; *know-how* knowledge is not considered.

Next the action-based rationale was developed. The claims are that actions can be explained by themselves (Lakin et al., 1989). Although this approach includes *know-how* knowledge in its target, it creates a new problem in managing the large volume of information recorded.

Model-based design rationale was the last to be proposed. The Active Design Document system (Garcia & Howard, 1992) is based on a certain computational model of design rationale, which is developed for parametric design tasks. Although this system works effectively, its model and method are limited to a certain subject. In contrast, we propose a much more general model to apply to broad design problems. The Design History Tool (Chen et al., 1991) is proposed, which has both action-based and model-based features. The system stores structured and hierarchical representations of design based on what designers said and performed. Although their system can provide more information about design rationale, it interrupts the designers' design process and involves a huge analysis cost for each design case.

Recently several studies about action- and model-based design rationale were tackled with *know-how* and *know-why* knowledge capturing. Ganeshan et al. (1994) proposed the framework to capture *how* and *why*, in which the core idea is to model design as a selection from predefined transformation rules. When a rule is selected, the choice is recorded along with the rationale associated with that rule. In their approach, the designers' activities are constrained and they are translated into the predefined rules beforehand. They did not address the issues of capturing *know-how* knowledge from the bottom-level information such as events in CAD systems.

Myers et al. (1999) proposed the framework to capture design rationales from general CAD data. They developed an experimental system, the Rationale Construction Framework (RCF), which automatically acquires rationale information for the detailed design process. It is valuable that they aim to develop a framework to apply to a general design problem. Although their research purpose is very similar to ours, there are some differences between us in both conceptualization and approach. In RCF they regarded design history as a conglomerate of detailed design rationales and focused on capturing many partly isolated design rationales in detail. Their focus is different from finding useful *know-how*, especially procedure features. On the other hand, our goal is to capture *know-how* knowledge, including procedure information and procedure features as mentioned above. Therefore, the approaches taken in Myers et al.'s research were different from ours. In their research,

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press

simple pattern matching is executed to detect design procedures using general predefined rules called design metaphors and qualitative reasoning is used to capture design rationale. In our research, complex pattern recognition (GEDP) is conducted to detect design procedures and a search based on rules derived from design principle knowledge is used to acquire procedure features.

In addition, some of the industrial Case-Based Reasoning (CBR) systems can be considered as capturing design process knowledge in terms of ways to fix problems. Several recent publications addressed specific issues of applying CBR to design (Maher & Pu, 1997). The application of CBR to design domains that involve mechanical and other physical devices has extended the notion of design cases to include a representation of general knowledge in the form of causal or heuristic knowledge. Goel et al. (1997) directly addressed the representation of causal behavior through the use of *structure–behavior–function* (SBF) models. SBF models are useful to diagnose products. However, they focus on states of product models rather than design history.

7. SUMMARY AND FUTURE WORK

Because engineering design is highly knowledge intensive, capturing, managing, and utilizing knowledge can yield significant benefits for both designers and enterprises. In this paper we categorized design knowledge, focused on *know-how* knowledge capturing, and proposed a novel model and methods. The three-layer design process model represents a generic design process by differentiating process information at different levels of detail. The GEDP method developed based on the process model acquires design procedures in a bottom-up way by extracting knowledge layer by layer through grammar and DP based analysis. Moreover, analyzing the relationships between the hierarchical tree structure of product models and design operations generated from the application of GEDP enables us to acquire the procedure features. The effectiveness of the proposed approach was demonstrated by a gear design prototype system Gear-CAD.

Because the aim of our approach is to acquire *know-how* knowledge without disrupting a normal design process, we adopted a bottom-up approach. In general, a bottom-up approach involves the difficulty that design events can be too minute and complicated to manage unitarily. To conquer this difficulty, we devised an approach in which two methods of pattern recognition are combined. We showed that our approach is effective in dealing with the low-level information management problem. In general, parametric design and routine design are most suitable for our method.

A limitation of our approach is that, although we can capture what a designer does as *know-how* knowledge, the reason *why* the designer did his or her design this way is still weak. *Know-why* knowledge is needed to profoundly understand *know-how* knowledge. To deal with the limitation, we plan to focus on *know-why* knowledge capturing as the next step of our research. We also plan to integrate the current system in an environment of networked design support agents to further support collaborative design through effective knowledge capturing and management.

ACKNOWLEDGMENTS

This research was supported in part by a NSF CAREER Award under Grant DMI-9734006. The authors are grateful to the NSF for its support.

REFERENCES

Bellman, R. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bradshaw, J.M., Carpenter, R., Cranfill, R., Jeffers, R., Poblete, L., Robinson, T., Sun, A., Gawdiak, Y., Bichindaritz, I., & Sullivan, K. (1997). Roles for agent technology in knowledge management: Examples from applications in aerospace and medicine. *Proc. AAAI Spring Symposium on Artificial Intelligence in Knowledge Management (AIKM'97)*, pp. 9–16.

Chen, A., Dietterich, T.G., & Ullman, D.G. (1991). A computer-based design history tool. *NSF Design and Manufacturing Conference*, pp. 985–994.

Chikada, T., Yoshimura, M., et al. (1999). An off-line signature verification method based on a hidden Markov model using column images as features. *Proc. 9th Biennial Conference of the International Graphonomics Society (IGS'99)*, pp. 79–82.

Conklin, E.J., & Begeman, M. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *Proc. Conf. Computer-Supported Cooperative Work (CSCW'88)*, pp. 140–152.

Davenport, T.H., & Prusak, L. (1998). *Working Knowledge: How Organizations Manage What They Know*. Boston: Harvard Business School Press.

Ganeshan, R., Garrett, J., & Finger, S. (1994). A framework for representing design intent. *Design Studies 15(1)*, 59–84.

Garcia, A.C.B., & Howard, H.C. (1992). Acquiring design knowledge through design decision justification. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 6(1)*, 59–71.

Goel, A., Bhatta, S., & Stroulia, E. (1997). KRITIK: An early case-based design system. In *Issues and Applications of Case-Based Reasoning in Design* (Maher, M., & Pu, P., Eds.), pp. 87–132. Mahwah, NJ: Erlbaum.

Graham, A.S. (1994). *String Searching Algorithms*. River Edge, NJ: World Scientific.

Jin, Y., & Zhou, W. (1999). Agent-based knowledge management for collaborative engineering. *Proc. Design Engineering Technical Conferences (DETC'99) in ASME*.

Jin, Y., Zhao, L., & Raghunath, A. (1999). ActivePROCESS: A process-driven and agent-based approach to collaborative engineering. *Proc. Design Engineering Technical Conferences (DETC'99) in ASME*.

Krogh, A., Brown, M., Mian, I.S., Sjolander, K., & Haussler, D. (1994). Hidden Markov models in computational biology: Application to protein modeling. *Journal of Molecular Biology 235*, 1501–1531.

Kunz, W., & Rittel, W. (1970). Issues as elements of information systems. Working paper 131. Center for Planning and Development Research, University of California, Berkeley.

Lakin, F., Wambaugh, J., Leifer, L., et al. (1989). The electronic design notebook: Performing medium and processing medium. *Visual Computer: International Journal of Computer Graphics 5*, 214–226.

Lee, J. (1990). SIBYL: A qualitative decision management system. In *Artificial Intelligence at MIT: Expanding Frontiers* (Winston, P., & Shellard, S., Eds.), Vol. 1, pp. 104–133. Cambridge, MA: MIT Press.

Liebowitz, J. (Ed.). (1999). *Knowledge Management Handbook*. Boca Raton, FL: CRC Press.

Maher, M.L., & Pu, P. (Eds.). (1997). *Issues and Applications of Case-Based Reasoning in Design*. Mahwah, NJ: Erlbaum.

McCall, R. (1991). PHI: A conceptual foundation for design hypermedia. *Design Studies 12(1)*, pp. 30–41.

Moran, T.P., & Carroll, J.M. (1996). *Design Rationale: Concepts, Techniques, and Use*. Mahwah, NJ: Erlbaum.

Myers, K.L., Zumel, N.B., & Garcia, P. (1999). Automated capture of rationale for the detailed design process. *Proc. 11th Conf. Innovative Applications of Artificial Intelligence (IAAI'99)*.
Pahl, G., & Beitz, W. (1996). *Engineering Design: A Systematic Approach*. New York: Springer–Verlag.
Sakoe, H., & Chiba, S. (1990). Dynamic programming algorithm optimization for spoken word recognition. In *Readings in Speech Recognition* (Waibel, A., & Lee, K., Eds.), pp. 159–165. San Mateo, CA: Morgan Kaufmann.
Suh, N.P. (1990). *The Principles of Design. Oxford Series on Advanced Manufacturing, Vol. 6*. New York: Oxford University Press.

Dr. Yoko Ishino is a Research Associate in the IMPACT Laboratory at the University of Southern California. She received her PhD degree in Information Engineering from the University of Tokyo in 1999. In her doctoral dissertation she investigated computer-aided strategic concept formation based on knowledge acquisition from questionnaire data. Her research interests are primarily in computer–human interactions that facilitate human intellectual activities, for example, engineering design. Since she joined the IMPACT Laboratory in the fall of 1999, she has studied knowledge acquisition in mechanical engineering and agent-based collaborative engineering support.

Dr. Yan Jin is Associate Professor of Mechanical Engineering at the University of Southern California and the Director of the USC IMPACT Laboratory. He received his PhD degree in Naval Engineering from the University of Tokyo in 1988. Since then Dr. Jin has done research on knowledge-based systems, distributed problem solving, and organization modeling, along with their applications to computer-integrated manufacturing, collaborative engineering, and project management. His current research interests include design process modeling, agent-based collaborative engineering support, and computational organization modeling. He is a recipient of a 1998 National Science Foundation CAREER Award.

https://doi.org/10.1017/S0890060402020073 Published online by Cambridge University Press