Proceedings of DETC'04
ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference
Salt Lake City, Utah, USA, September 28 – October 2, 2004

DETC2004/CIE-57681

DESIGN RATIONALE ACQUISITION IN CONCEPTUAL DESIGN BY HIERARCHICAL INTEGRATION OF ACTION, MODEL AND ARGUMENTATION

Yutaka Nomaguchi
Department of Computer-Controlled Mechanical Systems
Graduate School of Engineering
Osaka University
Suita, Osaka 565-0871, JAPAN
Email: noma@mech.eng.osaka-u.ac.jp
Tel: +81-6-6879-7324, Fax: +81-6-6879-7325

Atsushi Ohnuma
Department of Computer-Controlled Mechanical Systems
Graduate School of Engineering
Osaka University
Suita, Osaka 565-0871, JAPAN

Kikuo Fujita
Department of Computer-Controlled Mechanical Systems
Graduate School of Engineering
Osaka University
Suita, Osaka 565-0871, JAPAN
Email: fujita@mech.eng.osaka-u.ac.jp
Tel: +81-6-6879-7323, Fax: +81-6-6879-7325

ABSTRACT

This paper proposes a framework for acquiring design rationale in the conceptual design. Knowledge management is getting much attention for supporting the early phases of design process. While the research outcomes on design rationale might be useful in the direction, their capabilities are still far from practice. Such outcomes are categorized into model-based, argumentation-based and action-based frameworks. These are complementary when knowledge acquisition facility and explanatory power is considered. This paper introduces the hierarchical model of design rationale for overcoming the shortcomings of individual approaches under three frameworks. It consists of argument level, model operation level and action level. Argument level represents a designer's decision making process. Model operation level represents a sequence of design operations. A design operation follows and is followed by a design stage, which records a design snapshot over conceptual product model. Action level represents design process as a sequence of operation primitives, which are elementary and atomic operations on the product model. These linkages of three levels enable to automatically acquire design rationale through operation primitives over a design support system.

Keywords: Design Rationale, Design Process, Argument Model, Conceptual Design

1 INTRODUCTION

As civilization becomes so matured and manufacturing becomes so global, the key for market success shifts, for instance, from production cost reduction to life-cycle cost reduction, from cost-based competition to value-based competition, and so forth. Such grand trend has formed the research streams of concurrent engineering, design-for-X methodologies, product family design, etc. Their underlying meaning in the scope of engineering design is that the concerns of design activity become more knowledge-intensive. For example, today's manufacturers are required to not only provide each product with high integrity of quality, cost and delivery but also provide a variety of products continuously over product families in order to accommodate them to variation of customers' requirements (e.g., Fujita and Yoshioka, 2003). To achieve rational and effective design of such a high-integrated product family, various kind of design knowledge, e.g., knowledge of technology, customers' requirements and cost estimation, should be managed for its intensive use. Further since decision-making shifts from operational ones to strategic ones, the planning phase and conceptual phase in the design process have become more essential than ever corresponding to the importance of knowledge. This circumstance must be linked with that knowledge management becomes one of crucial issues in manufacturing or business competition.

Behind the above trends in design and manufacturing practice, the stream from knowledge engineering to knowledge management highlights the role of knowledge acquisition (Dieng, 2000). In the field of design research, design rationale is thought to be crucial toward design knowledge management. Its acquisition has been getting much attention in design research. Design rationale includes purpose of design, a justification of logical reasoning in design, a notation of design, a method of design and all other background knowledge (Moran and Carroll, 1996). Among various research outcomes, an argument model, i.e., IBIS (Issue Based Information System) (Kunz and Rittel, 1970), is often used to explicitly represent design rationale. It facilitates to structuralize design process by the network model including *issue*, *position*, *argument* nodes. However, because it does not have any linkage to product models, its power is insufficient to share and reuse design knowledge in complicated design problems. Such limitation is common among many frameworks for design rationale, since they aimed to reveal the knowledge-oriented issues rather than to support practical design problems. When considering the aforementioned demands on knowledge management for supporting design activity, this gap between scientific interests and practical requirements should be overcome.

This paper proposes a new framework for design rationale acquisition in conceptual design with aiming the application to practical design process of complicated problems such as ones of product families, etc. In the framework, a hierarchical model of design rationale is introduced for integrating several approaches, which have been developed under respectively different viewpoints, design activity is formulated as a set of design operation primitives, and design rationale is automatically acquired through such operation primitives over a design support system. In the following, its concepts, implementation and brief demonstration are reported.

2 PROBLEM OF DESIGN RATIONALE ACQUISITION

2.1 Three Categories of Approaches

As mentioned above, design rationale has been a theme of design research. Under the past research progress, Garcia and Howard (1992) classifies various attempts into three categories; model-based rationale, argumentation-based rationale, and action-based rationale. ADCT (Domeshek and Holman, 2002) is a web-based cooperative design support environments on model-based design rationale representation. Systems based on this approach have a conceptual model of the product, which consists of various background information of design, i.e., customer's values, functions, structure, standards, regulations and physical first-principals. The model-based design rationale can be easily reused because the model has much explanatory power, while it is difficult to describe rationale because the model is far from designer's thinking process. PHIDIAS (Shipman and McCall, 1997) is a system based on the argument model PHI. It is easier to describe design rationale than the systems based on model-based representation, but it is not easy to acquire rationale because a designer has to designate nodes of statements as issue or position and their relations. Electronic-Notebook (Lakin *et al.*, 1989) is a system based on action-based representation. This system is connected to an integrated design tool environment, and automatically records logs of the design process performed on this environment. This approach easily acquires design rationale because rationale is byproduct of design. This must be a critical concept to acquire design rationale. However, it is impossible to reuse captured rationale because it is just a log of design.

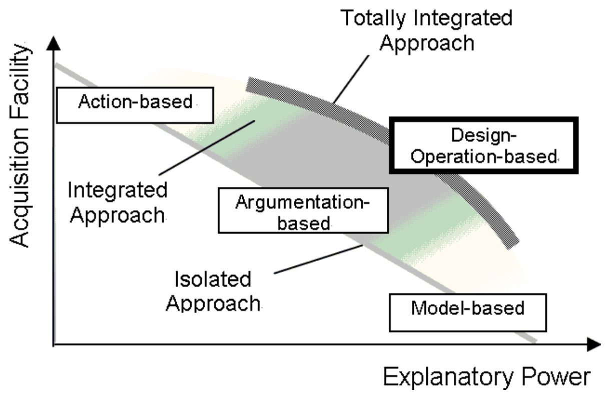


Figure 1. Approaches of design rationale acquisition

The facility to acquire design rationale and explanatory power of acquired design rationale is in tradeoff as shown in Fig. 1, when the above approaches are taken in isolated manner. Of course, the systems stated above usually integrate plural approaches. King and Banares-Alcantara (1997) proposed the IBIS-based argument model integrated with the conceptual product model. ADD (Garcia and Howard, 1992) and ADD+ (Garcia and de Souza, 1997) are the integrated implementation systems of argumentation-based and model-based design rationale, although its product model is specialized for HVAC systems. PHIDIAS (Shipman and McCall, 1997) integrates action-based and argumentation-based approaches by linking the log of the telecommunication tool and the argument model by keyword matching. These integrated approach can promote both acquisition facility and explanatory power. However, it still confronts the limitation unless the three approaches are totally integrated. Figure 1 indicates that any totally integrated approach should have explanatory power of argumentation-based and model-based approach, while it keeps knowledge acquisition facility of action-based approach.

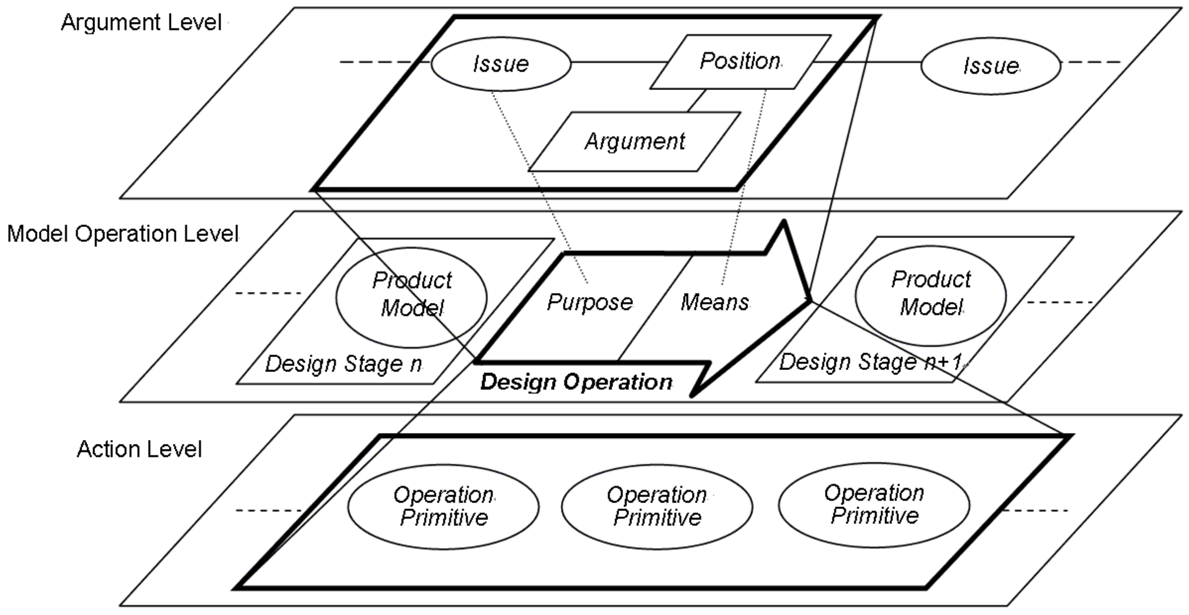


Figure 2. Hierarchical integration model

2.2 Design-Operation-based Approach

Although action-based approach has the best acquisition facility, the problem is that it is difficult to semantically link log of actions on design support tools with argumentation-based design rationale and model-based design rationale. The core idea proposed in this paper for integrating the tree approaches is *design operation*. The design operation is defined as the explanation of actions on the conceptual product model by explicitly defining the operation's purpose and means. The definition of the design operation includes the set of the *operation primitives* on the conceptual product model. Because of these features, the design operation takes part in respective design rationale approaches as follows. In the model-based viewpoint, the designer can carry out design on the conceptual product model by selecting the pre-defined patterns of design operation. In the argumentation-based viewpoint, the issue node, which corresponds to the purpose of the design operation, and the position node, which corresponds to the means, can be automatically generated by selecting a design operation. In the action-based viewpoint, argumentation-based and model-based design rationale can be acquired by recording the log of design operation primitives.

2.3 Hierarchical Integration of Design Rationale Approaches

Figure 2 illustrates the hierarchical model to integrate action-based, model-based and argumentation-based design rationale based on the design operation. This model consists of the three levels; argument level, model operation level and action level.

Argument level represents a designer's decision making process, which contains issue, position and argument nodes. Model operation level represents a sequence of design operations on the conceptual product model. A design operation follows and is followed by a design stage, which records a snapshot of a conceptual product model at a certain moment.

Action level represents design process by a sequence of operation primitives. An operation primitive is the most elementary and atomic operation on the product model.

Model operation level takes the core role in this hierarchical model. Whenever the designer carries out any design operation at model operation level, a sequence of operation primitives is activated at action level and the content of a product model is changed. This causes generation of issue and position nodes at the argument level.

2.4 Requirements for Implementation

Before discussing the details of design-operation-based integration, the requirements for implementing such an approach is discussed in the following.

2.4.1 Representation of design process The integrated approach should have a model of the design process to represent argumentation. Such a model should have the following features.

Representing decision making processes : The design process is organized based on designer's decision making. Takeda *et al.* (1992), for example, proposed cognitive design process model, which contains five steps; awareness of problem, suggestion, development, evaluation and decision. They used this model for protocol analysis of an actual design case. This kind of model can explicitly represent and structuralize design process.

Handling back-and-forth processes : The design process generally contains back-and-forth iterations. The model should handle such iterations in which plural alternatives for the same issue are concurrently evaluated, an issue is divided to sub-issues, and a designer goes back to a pended alternative.

Handling diverse types of information : The design process contains diverse types of information as text, drawings, figures, and CAD models. The design process model should be linked with them.

2.4.2 Conceptual model of product

To confront design rationale, a conceptual image of a product which is just like something in a designer's brain should be explicitly represented. Since such contents are so diverse, the terminology required for precisely distinguishing them become so huge. The introduction of abstracted schemes over direct representation can condense them into a relatively small number of patterns and relationships. By considering this viewpoint, the following features are important for acquiring design rationale.

Representing conceptual elements and their relations : The most common method for modeling a product is to divide it into a conceptual elements and relations among them.

In QFD (Quality Function Development), for example, a product is developed into a set of elements in different viewpoints such as functional, structural ones, and their mapping relationships are assessed for maximizing the product integrity. Kiriyama *et al.* (1992) proposed a metamodel, which is a symbolic network of concepts of a product. A metamodel represents a topological structure, attributes of a product, related physical phenomena and their underlying causality. Nomaguchi and Tomiyama (2002) used the metamodel as a basis of model-based design rationale.

This kind of conceptual product models by hierarchical system decomposition provides powerful representation of customer's requirements, functional requirements, physical first-principals, etc.

Comprehensive operation : Although a conceptual product model is powerful to represent design rationale, it is not comprehensive for the designer to operate the model directly. One of solutions for overcoming this shortage is to define meta-level operation, which is independent from a specific product. The meta-level operation is defined as an explanatory set of operation primitives. If the meta-level operation can be defined by relevant abstraction, it is more comprehensive for the designer to use meta-level operation than to operate the model directly because the designer can carry out design activity without attention to the operation primitives.

Integrating design tools and a product model : A designer is expected to carry out design on a conceptual model of a product as a workspace as much as possible. To enable it, various kinds of design support tools, i.e., QFD, value graph, cost evaluation, a solid modeler, etc., should be integrated with not only a conceptual product model but also all kinds of product models.

Regarding comprehensive operation, Yoshioka *et al.* (2001) proposed the meta-level operation mechanism of a metamodel-based design support environment. In this mechanism, every usual design operation is defined as a sequence of common primitives. The design support system implements specification set-up, solution synthesis, model operation, model analysis, etc. by various sets of operation primitives. This implementation framework makes the system flexible and robust against model diversity used in design activity. Such a feature is expected to be effective for acquiring design rationale as well.

2.4.3 Automatic acquisition of design rationale

The main feature of the design-operation-based approach is to acquire design rationale as a byproduct of design. The followings are required additionally to realize it, while the all requirements stated in the above are necessary as well.

Integrating product model and design process model : To acquire design rationale during design activity, a product model, which is integrated with various design support tools, should be integrated with a design process model so as that a designer implicitly describes his/her design process.

Automatic generation of design process model description : The integration of the product model and design process model reduces designer's workload to describe his/her design process. Beyond this, it is necessary that design process description is automatically generated from a log of operations on a product model.

3 HIERARCHICAL MODEL OF DESIGN RATIONALE

This section discusses the hierarchical model of design rationale that integrates action-based, model-based and argumentation-based design rationale. The hierarchical model is

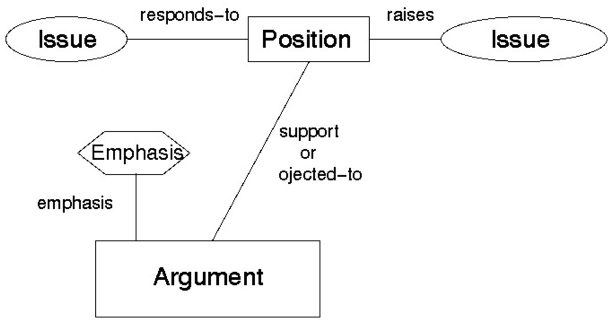


Figure 3. Argument model — Extension of IBIS —

the implementation methodology of the design-operation-based approach.

3.1 Argument Level

Argument level represents designer's decision making process. In this research, we developed the argument model by expanding IBIS (Kunz and Rittel, 1970). Figure 3 depicts our argument model. This argument model represents decision making process that includes issues and their positions by the network graph. The model consists of the following nodes and associated attributes and relations among them.

Issue is a node, which represents the issue discussed in the argument. An issue node has the following attributes;

- *Text description* : a description that explains the contents of the issue.
- *Focused information* : information in a product model that is argued in the issue.
- *Design stage* : a design stage in which the issue raised. (A design stage is explained later.)

Position is a node that represents an alternative solution to the issue. The position node has the relation *respond-to* to the issue, to which the position give a solution. The position node can be followed by an issue node, when a new issue is raised from the position. In this case, the relation *raise* is defined between the position node and the issue node. The position node has the following attributes;

- *Text description* : a description that explains the contents of the position.
- *Focused information* : information in a product model that is argued in the position.
- *Design stage* : a design stage in which the position suggested.

Argument is a node that represents an argument between one or plural positions. If the argument supports the position, the relation *support* is defined between them. If the argument objects to the position, the relation *objected-to* is defined. The argument node has the following attributes;

- *Text description* : a description that explains the contents of the argument.
- *Focused information* : information in a product model which is argued in the argument.
- *Design stage* : a design stage in which the argument raises.

Emphasis is a node which represents importance of the argument. The relation *emphasis* is defined between the argument and emphasis node. The emphasis node has the following attribute.

- *Importance* : a number that represents the importance of the argument.

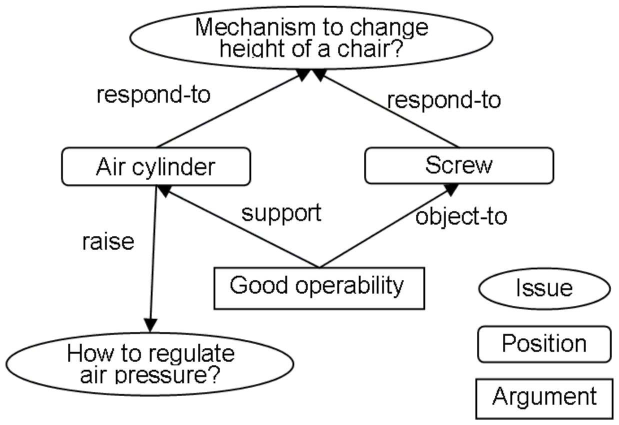


Figure 4. Example of argument representation

Text description of each node is written in HTML to handle various type of information by hyper link. The issue nodes, the position nodes the argument nodes, emphasis nodes and the relations between them represents designer's decision making process and back-and-forth process in design activity. Figure 4 illustrates an example of argument description in designing a chair.

3.2 Model Operation Level

Model operation level represents design operations performed on a conceptual product model.

3.2.1 Product architecture model A conceptual product model, which is developing by our research group for the research into product variety, is introduced. It is named 'product architecture model' for convenience in this paper. It is a network graph model to represent a product architecture, which is hierarchical and recursive relations between elements

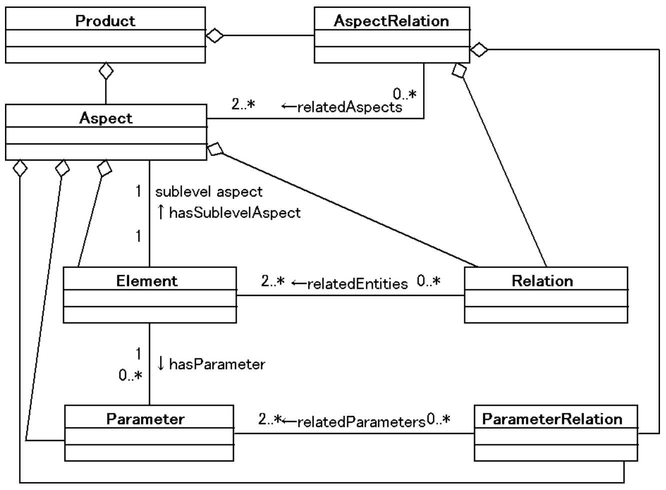


Figure 5. UML description of product architecture model

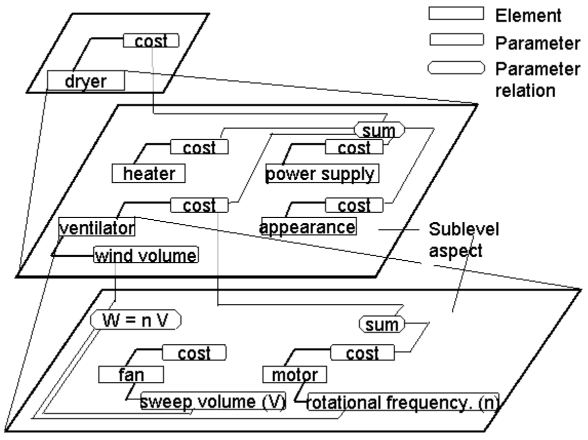


Figure 6. Hierarchical decomposition of aspects and elements in the example of hairdryer

of a product within a certain aspect view and between various aspect views. Because this model can integrate QFD, cost evaluation and other design support tools, it can be used as a workspace of a designer to acquire design rationale. Although the product architecture model can represent relations among products within/across product families, this research use it only to represent a single product. The product architecture model consists of the following nodes and relations. Figure 5 shows the UML (Unified Modeling Language) description of product architecture model.

Product is a node that gathers information about a product. A product node aggregates plural *aspects*.

Aspect is an aspect view, e.g., customer's requirement, function, and structure, and is associated with its contents to represent a product. That is, an aspect node represents information of a product from a certain aspect view. An aspect node aggregates *elements*, *parameters*, *relations* and *parameter relations*.

Element is a node that constructs a product. An element node has plural *parameters*. For example, elements of customer's requirement aspect are 'to dry hair,' 'highly safe,' etc., elements of function are 'generate air flow,' etc., and elements of structure are 'fan,' 'heater,' etc.

An element node has also one aspect, which is the linkage to its sub-elements. In the example of a hairdryer shown in Fig. 6, a structure element 'hairdryer' has an aspect view, which contains 'heater,' 'ventilator,' 'power supply' and 'appearance' elements in its sublevel abstraction. Furthermore, an element 'ventilator' has a sublevel aspect, which contains 'fan' and 'motor.'

Parameter is a node that represents attribute and character of an element. A parameter node has a unique value. For example, a customer's requirement element 'to dry hair' has a parameter 'importance,' and a structure element 'ventilator' has a parameter 'wind volume.'

Relation is a node that represents a relation between elements. For example, 'mechanical connection' is defined between 'fan' and 'motor,' and 'function-structure mapping' is defined between a function 'to ventilate wind' and a structure 'ventilator.'

Parameter relation is a node that represents a relation between parameters. A parameter relation has a qualitative or quantitative equation among the associated parameters. For example, among a parameter 'wind volume (W)' of 'ventilator,' 'rotational frequency (n)' of 'motor' and 'sweep volume (V)' of 'fan,' a parameter relation 'W = n * V' is defined.

Aspect relation is a node that represents a relation between aspects. A certain type of relations exists between elements and parameters in different aspects, i.e., 'function-structure mapping' exists between the function aspect and the structure aspect. An aspect relation node aggregates this type of relations. An aspect relation has plural aspects and aggregates relation nodes and parameter relation nodes.

3.2.2 Design operation The *design operation* is defined as a meta-level operation on the product architecture model. Figure 7 lists the design operations defined for the case study of designing a screwdriver stated in Section 4. Note that the lineup of design operations depends on the product model. This means that new design operations can be augmented by refining the product architecture model in the future works.

Figure 8 shows the definition of a design operation "Make function in detail." The format of the definition of each attributes is based on JAVA programming language. "Make function in

- Make customer’s requirement/function/entity in detail
- Add customer’s requirement/function/entity
- Set basic customer’s requirement/function/entity
- Set height
- Set width
- Set length
- Set color
- Set torque
- Set gear ratio
- Set revolution
- Set voltage
- Set constraint
- Set physical relational expression
- Edit QFD phase I
- Edit QFD phase II

Figure 7. List of design operations

Type: Make function in detail
Referred information: Function referredFunction;
Added information: Function addedFunctions[];
Purpose: To develop referredFunction to sub-functions.
Means: To add addedFunctions[] as sub-functions of referredFunction.
Operation primitives:

```
  developElement(referredFunction,
                     av = new AspectView());
                                  //operation primitive
  for (i = 0; i < n ; i++)
    addElement(av, addedFunctions[i]);
                                  //operation primitive
```

Figure 8. Definition of design operation “Make function in detail”

detail” is defined as a design operation to add plural functions to a certain function, which the designer wants to develop. The slot of *operation primitives* consists of one operation primitive *developElement* and n operation primitives of *addElement*. The underlined text represents the variable name, and bolded text represents the operation primitive.

The attributes of a design operation is explained with the example of Fig. 8.

Type : This indicates the type of design operations which are listed in Fig. 7, i.e., “Set height” and “Make function in detail”

Referred information type : This is the type of information of the product model, which is referred in execution of the design operation. As shown in Fig. 8, referred information type of “Make function in detail” is a single function. In the implemented design system, the content of this information is input by a designer.

Added information type : This is the type of information of the product model, which is added as a result of the design operation. As shown in Fig. 8, added information type of “Make function in detail” is the set of functions. The content of the information is inputted by a designer.

Purpose : This is text description that represents the purpose of the design operation. This is used when the generated text description of the issue node is defined. In case of “Make function in detail,” for example, the generated text of the issue node is “How to develop a function referredFunction to sub functions?” (see Fig. 11)

Means : This is text description which represents the means to complete the design operation. This is used when the generated text description of the position node is defined. In case of “Make function in detail,” for example, the generated text of the position node is “Function addedFunctions[] are generated as sub functions.” (see Fig. 11)

Operation primitives : This is a sequence of operation primitives automatically performed at action level when the design operation is carried out.

The assumption here is that design process on the product architecture model can be represented as a sequence of the design operations defined by this framework. Whenever a designer performs the design operation, the operation primitives defined in the design operation are activated to change the product architecture model. For each design operation, a script to generate description of the issue node and the position node in the argument model is defined (see Fig. 11). This script is activated when the design operation is performed, and then the description of the argument model which corresponds to the design operation is generated. The detail algorithm of this generation is stated in Subsection 3.4.

3.2.3 Design stage

A design stage is a node that records a snapshot of the product model at a certain moment. When a designer performs the design operation, the new design stage is automatically created and it records a new snapshot of the product model, which is changed as a result of the performed design operation. Because the preceding design stage is not vanished, the designer can turn back to a former design stage to which he/she wants. This can be done by selecting the node of the argument model, because it has link to the design stage (see Subsection 3.1).

3.3 Action level

Action level represents design process by operation primitives on the product model. We define an operation

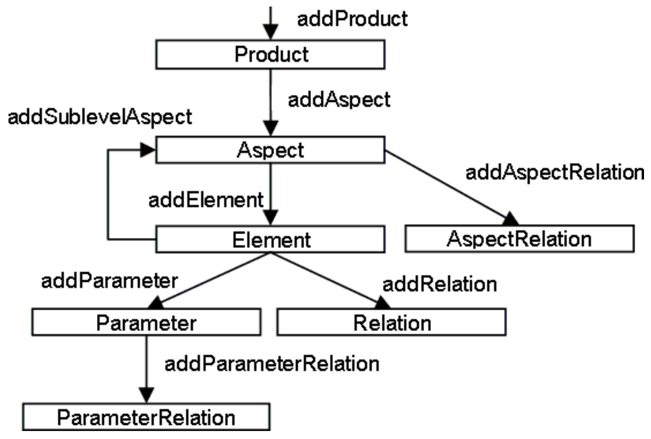


Figure 9. Operation primitives

Type: addElement
Referred information: AspectView referredAV;
Added information: Element addedElement;

Figure 10. Definition of operation primitive "addElement"

primitive, which is a fundamental augmentation of the product architecture model. We don't define an elimination operation, because our aim is to acquire all information of design process. When the designer wants to eliminate any information of the product model, he/she can turns back to the former design stage to perform a proper design operation again.

3.3.1 Operation primitive Because the product architecture model, which is used as the product model in this research, consists of seven classes as shown in Fig. 5, the eight types can be defined as a operation primitive as shown in Fig. 9. The eight operation primitives are required and enough to represent a fundamental augmentation of the product architecture model. An operation primitive has the following attributes.

Type : This describes a type of the operation primitive, i.e., *addElement*.

Referred information type : This is the type of information of the product model that is referred when the operation primitive is activated. For example, referred information of addElement is an aspect node to which a new element node is added.

Added information type : This is type of information of the product model that is added when the operation primitive is activated. For example, added information of addElement is a new added element node.

Figure 10 shows the definition of "add an element" operation.

```
ArgumentModelNode seletedNode = selectedNode();
if (seletedNode.isIssueNode) {
        // in case issue node is selected
   Issue issue = (Issue)seletedNode;
   } else {
   if (seletedNode.isPositionNode) {
        // in case position node is selected
   Issue issue = new Issue();
        // generate a new issue node.
   issue.setText("How to develop a function "
                     + referredFunction.getText
                     + "to sub functions?");
        // generate text description of an issue node
   issue.setFocus(referredFunction);
        // referredFunction is set as referred information
   issue.setDesignStage(designStageBeforDesignOperation);
        // link old design stage to issue node.
           }}
   Position position = new Position();
        // generate new position node
   for (i = 0; i < n; i++) {
        // generate text description of an position node
      position.addText("Function"
                           + addedFunctions[i].getText()
                           + ", ");
      }
      position.addText("are generated as sub functions ");
   for (i = 0; i < n; i++) position.addFocus(addedFunctions[i]);
        // added sub functions are set as added information
   position.setDesignStage(designStageAfterDesignOperation);
        // link new design stage to position node
   position.respondsTo(issue);
        // add respond-to relation between issue and
        // position node
```

Figure 11. Script to generate argument model for "Make function in detail"

3.4 Generation of Argument Representation

A feature of design-operation-based approach is to acquire design rationale during design activity on the conceptual product model. This subsection explains the algorithm to generate the argument representation as a result of performing design operations over the argument model. This algorithm can generate the issue and position nodes, while the designer manually describes the argument nodes.

The generation algorithm of the description of the argument model is as follows.

(i) The designer selects the design stage to perform the design operation by selecting the issue node or the position node of the argument model.
(ii) The designer selects the design operation, and inputs information required.

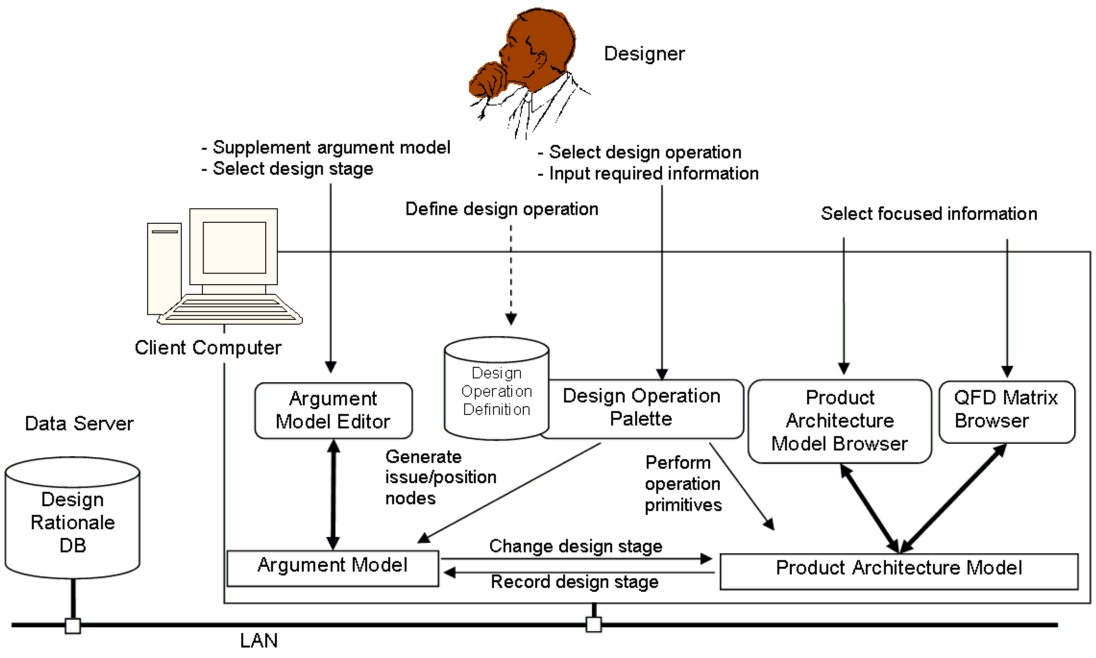

Designer
- Supplement argument model
- Select design stage
- Select design operation
- Input required information
Define design operation
Select focused information
Client Computer
Design Operation Definition
Argument Model Editor
Design Operation Palette
Product Architecture Model Browser
QFD Matrix Browser
Data Server
Design Rationale DB
Generate issue/position nodes
Perform operation primitives
Argument Model
Change design stage
Record design stage
Product Architecture Model
LAN

Figure 12. System architecture

(iii) The system executes operation primitives of the selected design operation.
(iv) The system creates the new design stage that records a snapshot of the product architecture model changed by the design operation.
(v) The system activates a generation script according to selected design operation. The script generates the issue node, the position node, text description of both nodes, *raise* and *respond-to* relation between them. In the case that the designer selected the issue node at the step (i), the system just generates a position node and makes respond-to relation between the selected issue node and generated position node instead of generating a new issue node. Figure 11 shows the script of the design operation "Make function in detail." The script is described based on the grammar of JAVA. The underlined text in this figure represents variables shared with operation primitives (see Fig. 8).
(vi) The designer describes contents of the argument node and the emphasis node.

4 IMPLEMENTATION

4.1 System Architecture

Based on the discussions above, an design system for conceptual design such as establishing product definition with the feature of design rational acquisition was developed in JAVA programming language (jdk 1.4.1) on Windows2000 and Solaris 8. Figure 12 shows the architecture of the system. The designer on the system can perform design by selecting design operation listed in the design operation palette. The system performs operation primitives to the product architecture model according to the selected design operation, and generates issue and position nodes in the argument model. The designer supplements the description of argumentation nodes. The system is integrated with Objectivity/DB, the object-oriented database. The database system is built on the distributed computer network so that designers on the network can share design rationale.

4.2 Functions

Because the system is integrated with the object-oriented database, the system has the function to store/restore design rationale. Besides, the system has the following functions, some of which are implemented for the case study of establishing product definition of a screwdriver.

Representing decision making processes by the argument model : The system can represent the designer's decision making process by the argument model, which consists of issue, position, argumentation, and emphasis node.

Managing design stages in back-and-forth processes : Each of the argument model nodes has the design stage, which records the snapshot of the product architecture model at a certain moment. This function facilitates the designer to turn back to previous design stage to redo design by selecting relevant the argument model node. Each design stage node has just references to objects of the product architecture model and activates the objects when the design stage is selected, this function does not waste so much memory resources of the computer.

Handling diverse type of information by HTML text : The argument model node has text description written in HTML text. This enables the designer record diverse types of information by hyperlinks.

Representing conceptual elements and relations : The system employs the product architecture model, which facilitates representation of customer's requirements, functional requirements for design, physical structure and their relations.

Comprehensive operation by design operation : We define design operations as meta-level operations on the product architecture model. The system shows the list of design operations and prompts the designer to select one. The design operation facilitates the designer to perform design by comprehensive operation on the product architecture model.

Integrating QFD into the product architecture model : The system integrates QFD matrix browser with the product architecture model. The designer can describe relative weight of customer's requirement, relative weight of relations between customer's requirement and function / function and structural component. The QFD facilitates the designer to evaluate the design alternatives from the view point of customer's requirement.

Integrating the product architecture model and the argument model : The designer can describe the argument model during design activity by using of the product architecture model.

Automatic generation of issue/position nodes in the argument model : The description of issue/position nodes in the argument model is automatically generated. This function reduces the designer's workload to describe the argument model.

4.3 Design Example

This subsection illustrates an application to designing a cordless screwdriver for demonstrating the capabilities of the implemented system. This design example is based on the structural analysis of the screwdriver, the whole shape and components of which are shown in Fig. 13. We analyzed customer's requirements, functions and QFD matrix of the screwdriver, and simulated its design process by using of the system.

Figure 14 shows the snapshot of the system while the designer carried out the design of a screwdriver. The design process on the system is carried out by selecting the design operation, which is listed in the design operation palette (Fig. 15). At first, the designer selects "Set basic Customer

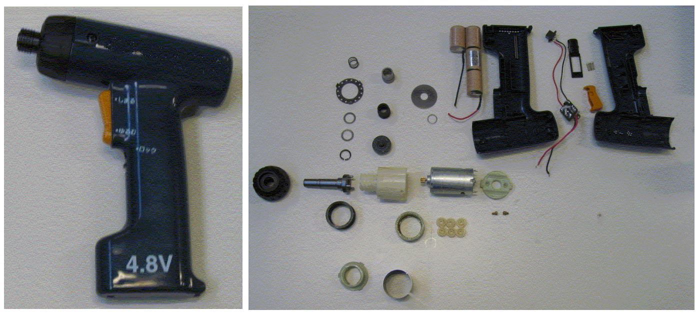
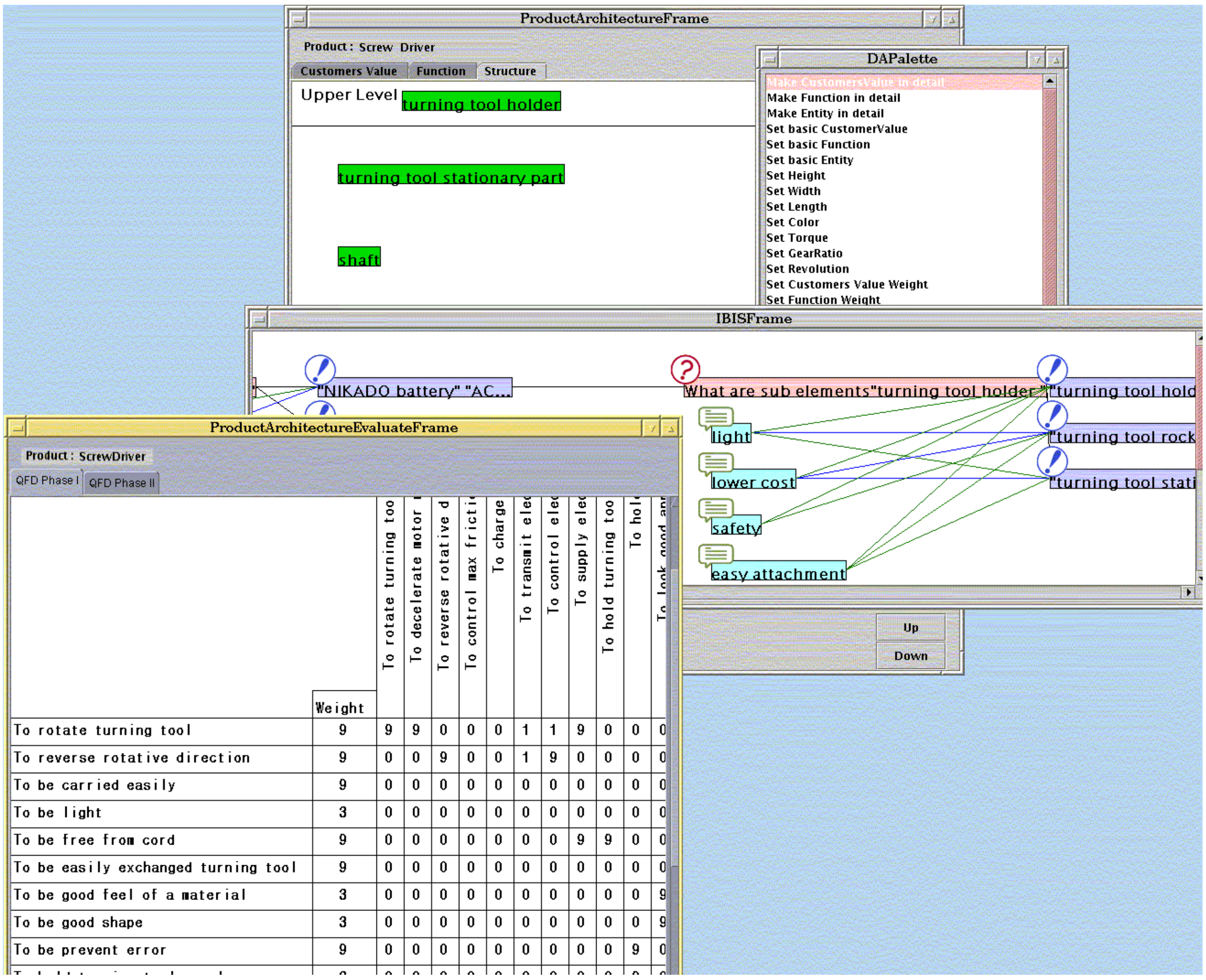


Figure 13. Screwdriver (left) and its components (right)

Value." The system prompts him/her to input the description of the basic customer value of the screwdriver. In this case, the designer described "Turn screw", and the basic customer value node is generated in the product architecture model.

After the basic elements in each aspect are represented, the designer recursively described the detail hierarchy model. Suppose the designer detail "Turing tool holder unit" in structure aspect. The designer selects the "Turing tool unit" node in the product architecture model browser, and selects "Make entity in detail" design operation. The system prompts to describe sub elements of the node. In this case, the designer suggested the mechanism, which holds the turning tool by the spring's elastic force, and described its consisting elements "Turning tool holder case," "Spring," "Ball bearing," "Shaft," and "Turning tool." By this design operation, the issue "What are sub elements of 'Turing tool unit'?" and its position "'Turning tool holder case,' 'Spring,' 'Ball bearing,' 'Shaft,' and 'Turning tool.' are added as sub elements." are automatically generated. Figure 16 shows an argument representation based on the argument model. The node marked by '? ' is the issue node, and the node marked by '! ' is a position node. In this case, the designer suggested other alternative solutions for "Turning tool holder." The designer selected the issue node of "What are sub elements of 'Turing tool unit'?" to turn back to the previous design stage. Then the designer selected the design operation "Make entity in detail" and described the alternatives. The second solution is the mechanism, which fixes the turning tool by the small screw bolt. The third is the mechanism which simply holds it by the elastic force of the rubber chuck. In Fig. 16, three alternative solutions are displayed in the argument model.

To evaluate there alternatives, the designer described the argument for them. In this case, such four argumentation nodes are created as "light," "low cost," "safety," and "easy attachment." Figure 17 shows the description of the argumentation node of "easy attachment." The designer described "the easier to attach to turning tool holder, the better" and put the explanatory figure by HTML text. In the argument model browser, the support link is represented in the green line,



Make CustomersValue in detail
Make Function in detail
Make Entity in detail
Set basic CustomerValue
Set basic Function
Set basic Entity
Set Height
Set Width
Set Length
Set Color
Set Torque
Set GearRatio
Set Revolution
Set Customers Value Weight
Set Function Weight

	Weight	To rotate turning too	To decelerate motor	To reverse rotative d	To control max fricti	To charge	To transmit ele	To control ele	To supply ele	To hold turning too	To hol	To look good an
To rotate turning tool	9	9	9	0	0	0	1	1	9	0	0	0
To reverse rotative direction	9	0	0	9	0	0	1	9	0	0	0	0
To be carried easily	9	0	0	0	0	0	0	0	0	0	0	0
To be light	3	0	0	0	0	0	0	0	0	0	0	0
To be free from cord	9	0	0	0	0	0	0	0	9	9	0	0
To be easily exchanged turning tool	9	0	0	0	0	0	0	0	0	0	0	0
To be good feel of a material	3	0	0	0	0	0	0	0	0	0	0	9
To be good shape	3	0	0	0	0	0	0	0	0	0	0	9
To be prevent error	9	0	0	0	0	0	0	0	0	0	9	0

Figure 14. System snapshot of designing screwdriver

and the opposing link in the red line. Because the second solution is supported by all of the four argumentation nodes, the designer chose it as the best. The designer also can use QFD matrix to verify the solution (see Fig. 14).

5 DISCUSSION

5.1 Related works

Design-operation-based approach can be evaluated from the three approaches, i.e., model-based, argumentation-based and action-based ones. Regarding model-based one, we use the product architecture model as a conceptual product model. Regarding argumentation-based one, we use the extended IBIS model as the argument model. And, regarding action-based one, we introduce design operation to automatically generate the description of the argument model.

The other aspect of design rationale acquisition is automatic versus user-intervention (Hu *et al.*, 2000). The automatic approach assumes that there is a method to capture the communication among the designers and design teams or between the designer and the design support system. The communication records can then be used to extract design rationale as they evolve during the design process. PHIDIAS (Shipman and McCall, 1997) is based on this approach. PHIDIAS first represents all of its knowledge, including semi-formal rationale as well as formal representation of physical objects, facts, and rules, in a common graph-based format. Then it generates hyper-links to interconnect items of knowledge by matching key words, regardless of their level of formality. This approach can reduce workload of the designer to input design rationale. However, it is difficult to acquire exactly design rationale but just linking among information. While the system

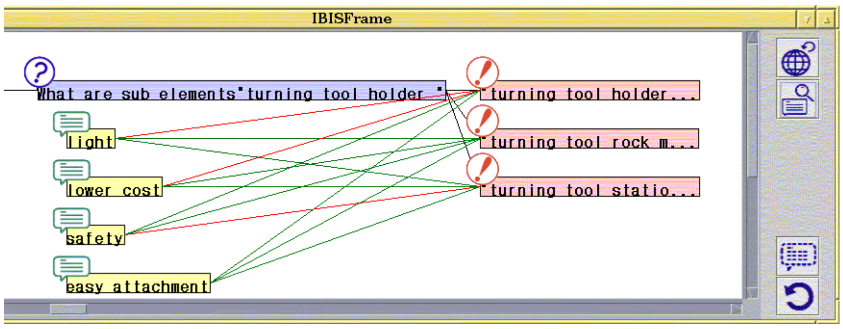
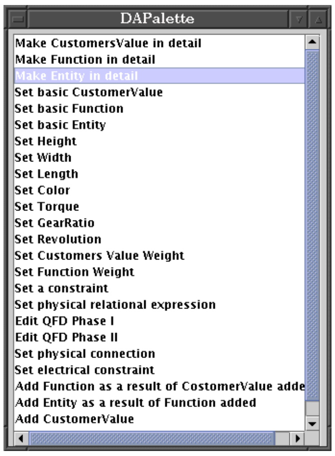
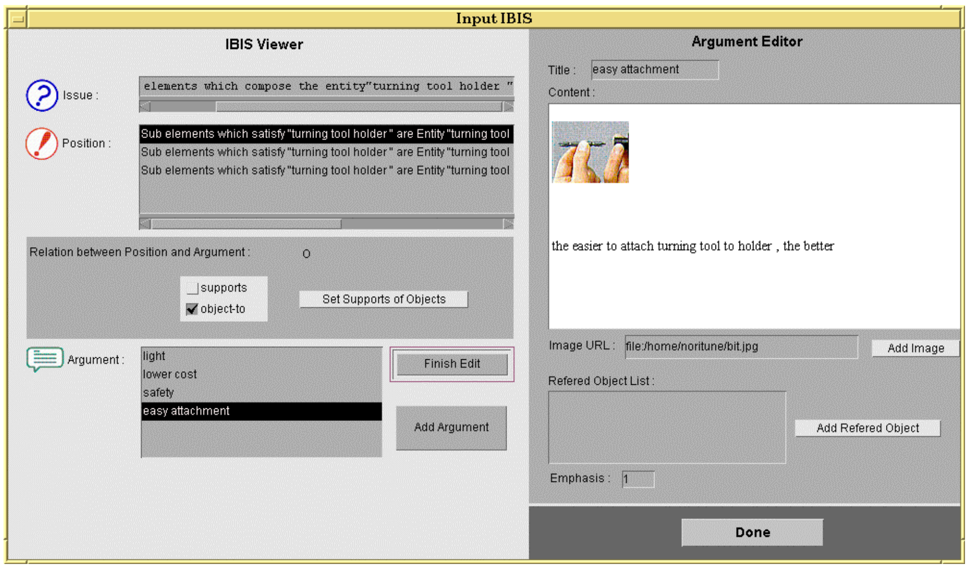


Figure 16. Alternative solutions described in argument model



Make CustomersValue in detail
Make Function in detail
Make Entity in detail
Set basic CustomerValue
Set basic Function
Set basic Entity
Set Height
Set Width
Set Length
Set Color
Set Torque
Set GearRatio
Set Revolution
Set Customers Value Weight
Set Function Weight
Set a constraint
Set physical relational expression
Edit QFD Phase I
Edit QFD Phase II
Set physical connection
Set electrical constraint
Add Function as a result of CostomerValue adde
Add Entity as a result of Function added
Add CustomerValue

Figure 15. Operation palette



Issue : elements which compose the entity"turning tool holder "

Position :
Sub elements which satisfy "turning tool holder" are Entity "turning tool
Sub elements which satisfy "turning tool holder" are Entity "turning tool
Sub elements which satisfy "turning tool holder" are Entity "turning tool

Relation between Position and Argument : 0
supports
[✓ object-to]

Argument :
light
lower cost
safety
easy attachment

Title : easy attachment
Content :
the easier to attach turning tool to holder , the better
Image URL : file:/home/noritune/bit.jpg
Refered Object List :
Emphasis : 1

Figure 17. Describing arguments

implemented is categorized into on automatic approaches, it can acquire not only links of information but also design rationale based on the argument model. The one of remaining issues of our system is that the designer has to input the description of the argument nodes. Garcia and Howard (1992)'s ADD is based on user-intervention approach. ADD implements a "designer's apprentice" mechanism. In ADD, a product model is represented as a set of parameters. If a designer proposes a value different from the value that the system expected, the system asks the designer for justification regarding these differences. The designer inputs the justification and the system records it. While the designer should manually input the justification, this approach has possibility to acquire designer's implicit knowledge when the system not officiously but relevantly asks questions to the designer. Our future works may include employing this approach to promote the designer to input the description of the argument node.

5.2 Corporate Application

To record design rationale, a manufacture generally attempts to document design, which includes not only the result of design but also background information of design (Suzuki *et al.*, 1996). Although documentation of design is not the regular work of a designer, it is reported that a designer spends 23% of his/her working time for documentation (Crabtree *et al.* , 1997). This is because documentation of design is very indispensable to record (or acquire) design rationale.

The problem is that a designer usually processes background information in implicit manner so that he/she merely remembers it after design is completed. Even if he/she records background information as a memo during design activity, which is not explicitly structured, it is difficult to systematically manage it once design activity is finished. This causes lack of background information in documenting design. In order to acquire background information, documentation of design should be carried out simultaneously with design activity. Although some groupware system, i.e., Lotus Notes, can partially support design documentation by integrating a CAD system and a word processor, they cannot support documentation itself. Documenting design during design activity is still heavy load for a designer.

The system implemented can be expanded to provide the design documentation environment, which facilitates describing design document structured by the argument model during design activity, when the product model is expanded enough.

6 CONCLUSIONS

This paper reported the research exploring a new framework for acquiring design rationale especially in the conceptual design stage of practical products. In the framework, the design process, argument, model operations are hierarchically integrated through design operation primitives. For ascertaining its fundamental capabilities, an experimental design system for conceptual design was implemented and applied to the design problem of a screwdriver. This application demonstrated that the proposed framework enables to automatically acquire design rationale simultaneously with progress of design activity, for a designer to track recorded rationale afterward, and so forth. Since the research and implementation has been just started, future works remain in various directions, such as of user-intervention support, retrieval and reusing of design rationale, enhancement of product model and associated design operations, toward practical usage for product design and development.

ACKNOWLEDGMENTS

The authors appreciate Satoshi Idaka, Yuji Suzuki and Masafumi Noritsune for their assistance in the design session and the system development. This research has been carried out partially at the Strategic Research Base, Handai (Osaka University) Frontier Research Center supported by the Japanese Government's Special Coordination Fund for Promoting Science and Technology.

REFERENCES

Crabtree, R. A., Fox, M.S., and Baid, N. K., 1997, "Case Studies of Coordination Activities and Problems in Collaborative Design," *Research in Engineering Design*, Vol. 9, No. 2, pp. 70-84.

Dieng, R., 2000, "Knowledge Management and the Internet," *IEEE Intelligent Systems*, Vol. 15, No. 3, pp. 14-17.

Domeshek, E. A. and Holman, E., 2002, "Web-based Design Coordination," *Proceedings of the 2002 ASME Design Engineering Technical Conferences*, Paper No. DETC2002/CIE-34404.

Fujita, K. and Yoshioka, S., 2003, "Optimal Design Methodology of Common Components for a Class of Products: Its Foundations and Promise," *Proceedings of the 2003 ASME Design Engineering Technical Conferences*, Paper No. DETC2003/DAC-48718.

Garcia, A. C. B. and Howard, H. C., 1992, "Acquiring Design Knowledge through Design Decision Justification," *AI EDAM*, Vol. 6, No. 1, pp. 59-71.

Garcia, A. C. B. and de Souza, C. S., 1997, "ADD+: Including rhetorical structures in active documents," *AI EDAM*, Vol. 11, No. 2, pp. 109-124.

Hu, X., Pang, J., Pang, Y., Atwood, M., Sun, W. and Regli, W.C., 2000, "A Survey on Design Rationale: Representation, Capture and Retrieval," *Proceedings of the 2000 ASME Design Engineering Technical Conferences*, Paper No. DETC2000/DFM-14008.

King, M. P. J. and Banares-Alcantara, R., 1997, "Extending the scope and use of design rationale records," *AI EDAM*, Vol. 11, No. 2, pp. 155-167.

Kiriyama, T., Tomiyama, T. and Yoshikawa, H., 1992, "Qualitative Reasoning in Conceptual Design with Physical Features," in Faltings, B. and Struss, P. (eds.): *Recent Advances in Qualitative Physics*, The MIT Press, Cambridge, MA, USA, pp. 375-386.

Kunz, W., and Rittel, H., 1970, "Issues as elements of information systems," *Working Paper* No. 131, Berkeley, University of California, Berkeley, Institute of Urban and Regional Development.

Lakin, F., Wambaugh, J., Leifer, L., Cannon, D., and Sivard, C., 1989, "The Electronic Design Notebook: Performing Medium and Processing Medium," *Visual Computer, International Journal of Computer Graphics*, Vol. 5, pp. 214-226.

Moran, T. P. and Carroll, J. M., 1996, *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates.

Nomaguchi, Y. and Tomiyama, T., 2002, "Design Knowledge Management based on the Model of Synthesis," Borg, J. C. and Farrugia, P. (eds.): *Proceedings of The Fifth IFIP WG5.2 Workshop on Knowledge Intensive CAD*, pp. 62-81.

Shipman, F.M. III, and McCall, R.J., 1997, "Integrating different perspectives on design rationale: Supporting the emergence of design rationale from design communication," *AI EDAM*, Vol. 11, No. 2. pp. 141-154.

Suzuki, H., Kimura, F., Moser, B. and Yamada, T., 1996, "Modeling information in design background for product development support," *Annals of the CIRP*, Vol. 45, No. 1, pp. 141-144.

Takeda, H., Tomiyama, T. and Yoshikawa, H., 1992, "A Logical and Computable Framework for Reasoning in

Design," *Proceedings of the 1992 ASME Design Theory and Methodology* (*DTM '92*), pp. 167-174.

Yoshioka, M., Nomaguchi, Y., and Tomiyama, T., 2001, "Proposal of an Integrated Design Support Environment Based on the Model of Synthesis." *Proceeding of the 2000 ASME Design Engineering Technical Conference and Computers and Information in Engineering Conference*, Paper No. DETC2001/DAC-21155.