DRIFT: A Framework for Ontology-based Design Support Systems

Yutaka Nomaguchi[1] and Kikuo Fujita[1]

Osaka University, 2-1 Yamadaoka, Suita, Osaka 565-0871, Japan

Abstract. This paper proposes a framework for ontology-based design support systems, called DRIFT (Design Rationale Integration Framework of Three layers), which records, structures and retrieves reflective operations and their relationships in design process. Although an ontology can provide concepts of static knowledge, such as knowledge about first principles of physical phenomena or prescription of successful design cases, a dynamic aspect of design process is usually out of ontology support. DRIFT automatically captures and manages the whole design argumentation through tracking design operations over an ontology-based design support system. This paper states our position and demonstrates a prototype system of DRIFT to show its performances and effectiveness.

1 Introduction

Engineering design is the process for generating a concept of useful artifact or product based on various kinds of scientific and engineering knowledge. According to Schön's assertion, it is insufficient only to apply the already-systematized knowledge to solve a complicated design problem, but it is important to dynamically, flexibly and adaptively acquire knowledge through reflection-in-action [1], which is trial-and-error design process that includes framing a problem, suggesting multiple alternatives, evaluating what-if and accepting or rejecting. We have been developing a framework for an advanced ontology-based design support system [2, 3], called DRIFT (Design Rationale Integration Framework of Three layers). A DRIFT system can capture reflective design process as a byproduct of inherent design operations that are defined over an ontology. This paper states our positions and briefly demonstrates a prototype system of DRIFT to show its performances and effectiveness.

2 Overview of DRIFT

2.1 Ontology and Reflection-in-action

An ontology gives a framework to manage various aspects of design knowledge. For instance, Kitamura has been developed a meta-data schema for systematically representing functionality of a product based on Semantic Web

technology for the management of the information content of engineering design documents [4]. Grosse's group has developed ontologies for knowledge-based systems to support modeling optimization and modeling physical phenomena [5, 6]. These ontologies provide concepts of static knowledge that is independent from a specific design case, such as knowledge about first principles of physical phenomena, typical patterns of modeling and optimization or prescription of function modeling.

Dynamic knowledge depending on a design case can be represented on the ontology, and can be reused and shared among different designers. On the other hand, due to the open-ended nature of design problems, a designer should create this dynamic knowledge through active design process. Schön stated that a key concept to do this is reflection-in-action [1]. Reflection-in-action is trial-and-error design process that includes framing a problem, suggesting multiple alternatives, evaluating what-if and accepting or rejecting. An advanced design support system should support this nature of design.

2.2 Architecture of DRIFT

DRIFT is a software module that is being developed to dynamically capture such reflective design process as a by-product of inherent design operations that are defined over an ontology. DRIFT facilitates a designer to compare multiple alternatives concurrently during design process, and to review associated design alternatives. Figure 1 shows the outline of the situation that a designer is involved under a DRIFT system. Foundation of DRIFT consists of three components; an interface for a designer to perform design operations, a simple truth maintenance system [9] to efficiently record a state transition of design, and an IBIS (Issue-based Information System) [10] based argumentation browser that shows accepted and rejected alternatives.

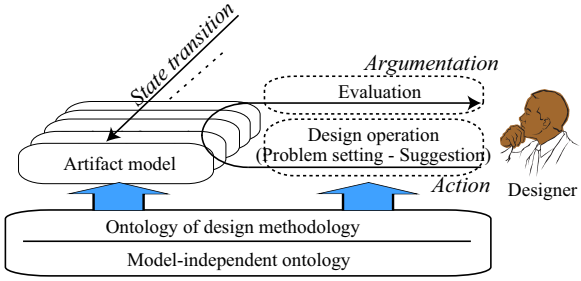

Fig. 1. Outline of reflective design process supported by DRIFT framework

Under the mechanism of DRIFT, each design operation is defined as a pattern of problem setting and alternative solution in order to capture both design state transition and an argumentation structure through an inherent design action. For example, a design operation that details a customer need of a product is defined as follows; a problem set by the operation is 'what are sublevel customer needs of it?' and its alternative solution is a set of its sublevel customer needs.

Since the system records all alternatives suggested as solutions of the problem, a designer can compare them and review one at any time.

An ontology is defined as a meta-data structure that enables capturing design state transition and argumentation structures almost automatically through designer's inherent design actions. While the system includes a set of fundamental ontologies for implementing basic functionality, another set of subsidiary ontologies for capturing and managing practical and complicated design operations must be configured for individual directions of design supports. In other words, it is necessary and essential that a set of specific ontologies must be formulated for supporting a particular type of design operations in order to apply a design methodology under the corresponding context.

3 DRIFT System for QFD-based Cost-worth Analysis

In order to demonstrate the validity and promise of DRIFT, a prototype design support system under a QFD-based cost-worth analysis method [11] was developed. This section briefly explains an ontology behind this system.

3.1 Ontology of DFX

The problem with defining an ontology for design support systems is how we decide a borderline that distinguishes static knowledge from dynamic knowledge. An effective and efficient guideline on this issue is that a design-for-X (DFX) methodology can provide this.

A purpose of a design methodology is to have prescriptions that advocate how design should be done in particular circumstances [7]. Various DFX methodologies have been revealed and proposed under the trends of concurrent engineering. Among various DFX methodologies, QFD (Quality Function Deployment) is a typical and comprehensive one [8]. It is effective for exploring and defining design requirements by a sequential procedure, for instance, across customer's requirements, functional realization, manufacturing modules, production process, etc. By means of its reflective refinement, a designer gets overall image of a product. While such an instruction is easy to understand for designers, it consists of some general and abstract concepts by excluding its case-dependent aspects. Therefore, it is suggested that QFD promotes more efficient and effective information sharing and discussion among designers in practical use. Although a DFX methodology is not a theory established by a scientific law such as physics, it is often used in design practices as a rational prescription of design contexts.

Because a purpose of a DFX methodology is to provide prescriptions that advocate how design should be done in particular circumstances, taxonomy of design and a pattern of design operations are tacitly included in a DFX methodology. Their meaning is what an ontology is.

3.2 Ontology Definition

Before implementation of a DRIFT-based system, an ontology corresponding to a set of DFX methodologies is configured as a base of a design support system. An ontology of a DRIFT system consists of taxonomy to describe an artifact and patterns of problem setting.

Taxonomy An ontology consists of two layers; model-independent layer, which consists of concepts independent from any specific design methodology, and model-dependent layer, which consists of concepts to represent specific prescriptive design methodologies. In a prototype system, 15 concepts are defined as shown in Figure 2. Element, hierarchy, relation, attribute and attribute value are model-independent concepts. A concept for representing specific methodology; value graph, function-structure mapping, QFD two-dimensional tables and cost-worth graph, is defined as a subclass of a model-independent concept. There is a possibility that other concepts are defined in addition to concepts explained in this subsection when another methodology is integrated.

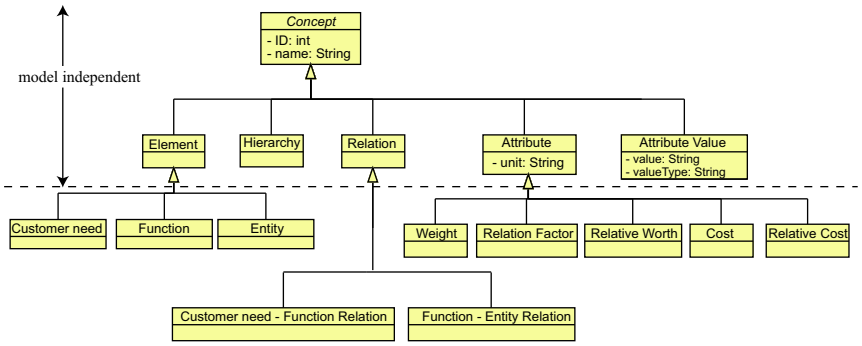

Fig. 2. Ontology for QFD-based Cost-worth Analysis

Patterns of problem setting A sequence of design operations under the implemented methodologies is, for example, executed in the following way; making QFD two-dimensional tables of product, estimating cost of entities, evaluating a balance of cost and worth, and go back to a former step if necessary. These steps are not proceeded straight forward but in iterative way. Therefore, a designer is required to reflectively consider elements, their relations, weights of customer needs, and cost of entities represented in two dimensional tables of QFD. The following ten patterns of problem setting are extracted for capturing a whole design process in this methodology; *what is a top customer need?* , *what is a top function?* , *what is a top entity?* , *what are sublevel customer needs?* , *what are sublevel functions?* , *what are sublevel entities?* , *how much is a customer need important?* , *which functions are related to a customer need?* , *which entities are related to a function?* and *how much is cost of entity?*

3.3 Design operation

A pattern of problem setting is used to reflectively operate a state of design and to make argumentation. This is implemented as a design operation performed over a DRIFT system. For the QFD-based cost-worth analysis method, this research defines ten design operations each of which corresponds to a pattern of problem setting. For example, a design operation for *'what are sublevel functions?'* is defined as follows.

Operation name: Function development
Refereed nodes: Function f
Added nodes: Function [] *subF*, Hierarchy h
Problem setting: What are sublevel functions of f ?
Solution: *subF*
Operation primitives: 1. adding *subF*, 2. adding h, 3. adding justification from f and *subF* to h

A design operation defines *operation primitives*, which are operations to TMS. When a design operation is performed, operation primitives of the list are performed. A design operation defines *referred nodes*, which are requisites of an operation, and *added nodes*, which are added as a result of an operation. In the definition of 'function development,' a function node is referred, and multiple function nodes and a hierarchy node are added as a result of this operation.

Under the above definition of design operations, the following procedure generates IBIS description after a design operation O_n is performed. The detail of this algorithm is explained in our articles [2, 3].

1. Creating a new issue node I_n, which has referred nodes of O_n in its focused node list, and has operation name of O_n in its operation type.
2. Searching an issue node I_s which is the same as I_n. If I_s is found, $I_n := I_s$.
3. Creating a new position node P_n, which has added nodes of O_n in its focused node list, and has operation name of O_n in its operation type.
4. Searching a position node P_s which is same as P_n. If P_s is found, $P_n := P_s$.
5. Searching a position node P_{n-1}, whose focused node list contains at least one of focused nodes of I_n.
6. Connecting *raised* relation from P_{n-1} to I_n.
7. Connecting *respond-to* relation from I_n to P_n.

Here, the *same* IBIS node is defined as a node which has the same operation type and the same focused node list.

4 Demonstration

4.1 Implementation

The DRIFT system for a QFD-based cost-worth analysis method was developed in Java programming language (jdk 1.4.1) on Windows XP. Figure 3 shows its architecture.

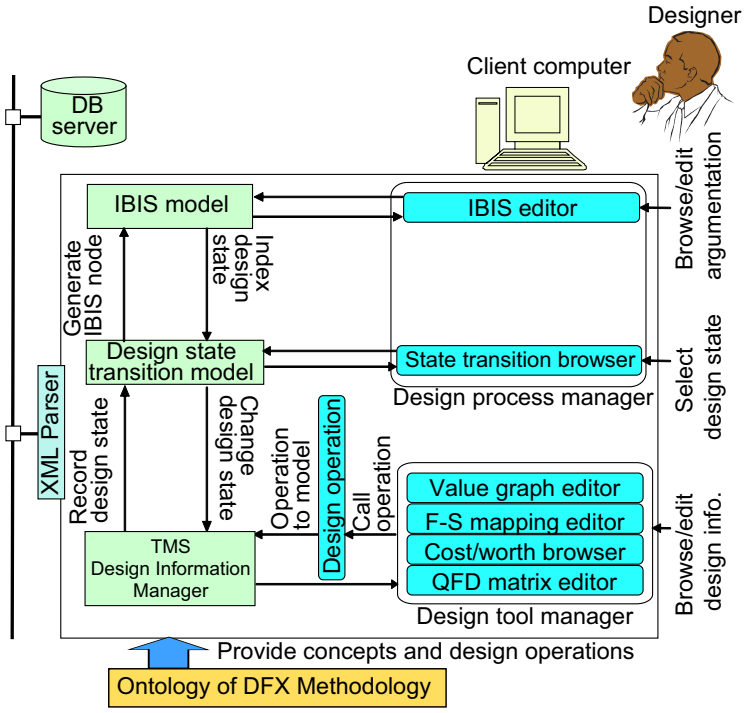

Fig. 3. System architecture

Under the implemented prototype system, a designer carries out identification of design target by tools of design methodologies; value graph, function-structure mapping, QFD two-dimensional tables and cost-worth graph. The system sends design operations to a truth maintenance system when a designer inputs design information on the tools. Design process is automatically recorded along designer's actions and operations over the embedded tools. A designer can edit description of an argumentation structure. A recorded design process is stored in database in XML format.

4.2 Design Example

This subsection illustrates an application to designing a cellular phone for demonstrating the capabilities of the implemented system. In this example, it is assumed that a product is aimed to Japanese market and that it is equipped with many value-adds such as camera, music player, electric money, bar-code scanner, etc.

A designer enumerates customer needs, functions and entities of a cellular phone by using value graph and function-structure mapping. Figure 4-① shows a hierarchical structure of customer needs on a value graph.

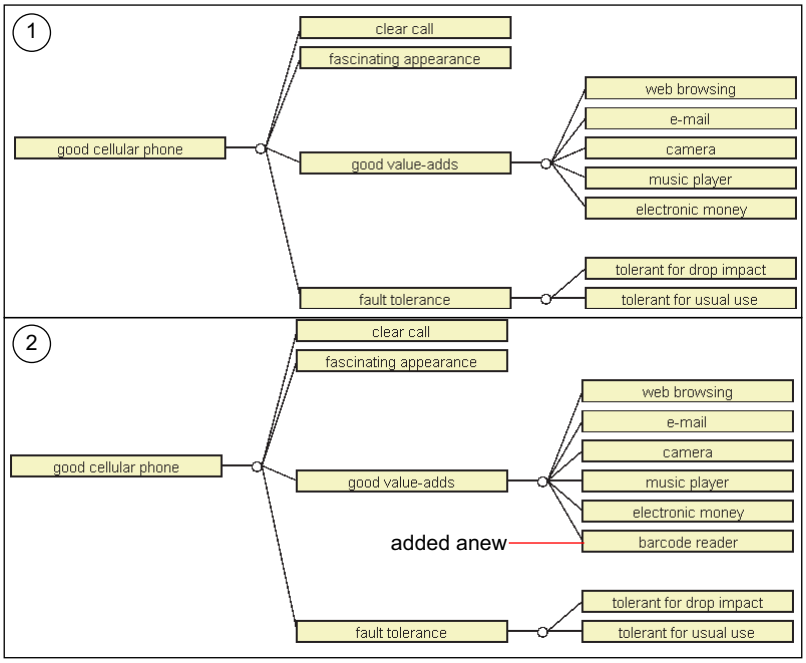

Fig. 4. Alternatives of hierarchical structure of customer needs

Then, a designer uses QFD table to set weights of customer needs (see Figure 5-①). The weights are deployed to weights of entities by automatic calculation of QFD two-dimensional tables. A designer also sets cost of each entity, and evaluates balance of relative cost and relative worth of each entities that cost-worth graph shows (see Figure 6-①). This graph reveals that cost of camera lens would be higher than its worth.

In order to dissolve this unbalance of cost and worth, a designer has to choose either from among two; (A) reducing relative cost or (B) increasing relative worth as shown in Figure 6. A possible solution can be enumerated according to the patterns of problem setting. According to problem setting 'how much is cost of entity?,' reducing cost of camera lens can be proposed as an alternative solution of (A). Otherwise, according to a problem setting 'how much is a customer need important?,' targeting customers who care good quality of camera can be proposed as an alternative solution of (B). In this case, a designer chose to increase relative worth of a camera lens. By a problem setting 'what are sublevel customer needs?,' a designer added 'barcode reader' as a sublevel customer need of 'good value-adds'(see Figure 4-②). Then, he/she revised customer needs weights by a problem setting 'how much is a customer need important?' as shown in Figure 5-②. A relative worth of a camera lens is increased by these operations (see Figure 6-②).

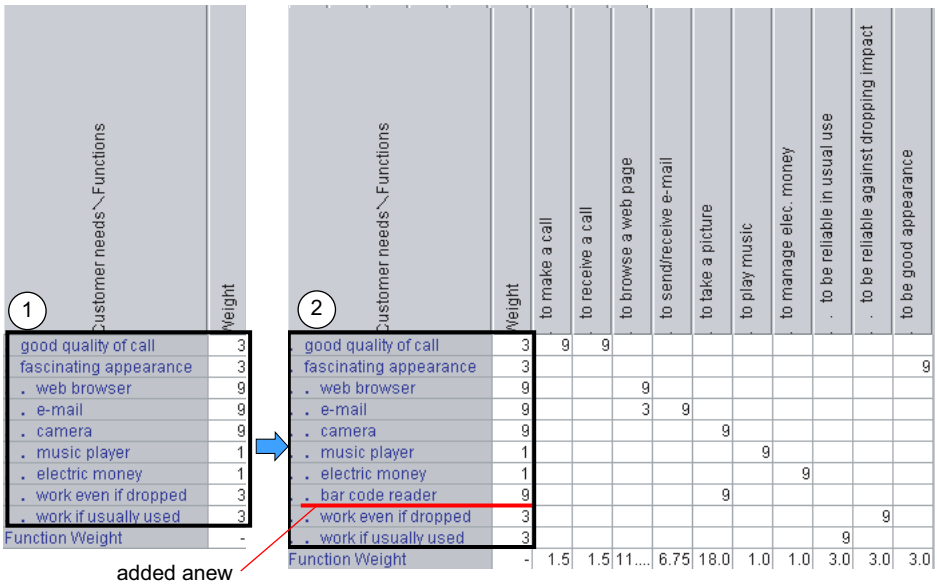

Fig. 5. Alternatives of weighting customer needs

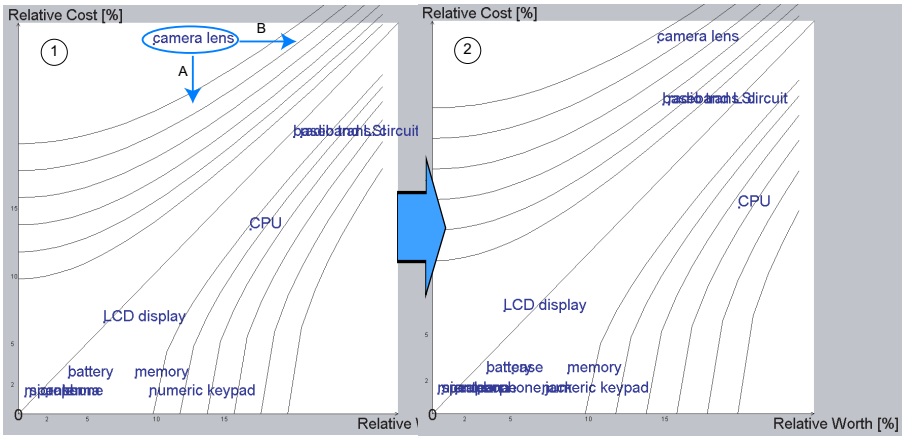

Fig. 6. Revision of cost-worth balance

The design process over the system is automatically captured as a byproduct of a sequence of design operations. Figure 7 shows a screen shot of a part of the captured process of the cellular phone design. A designer reviews and evaluates multiple alternatives of target customer needs by the QFD-based cost-worth analysis method.

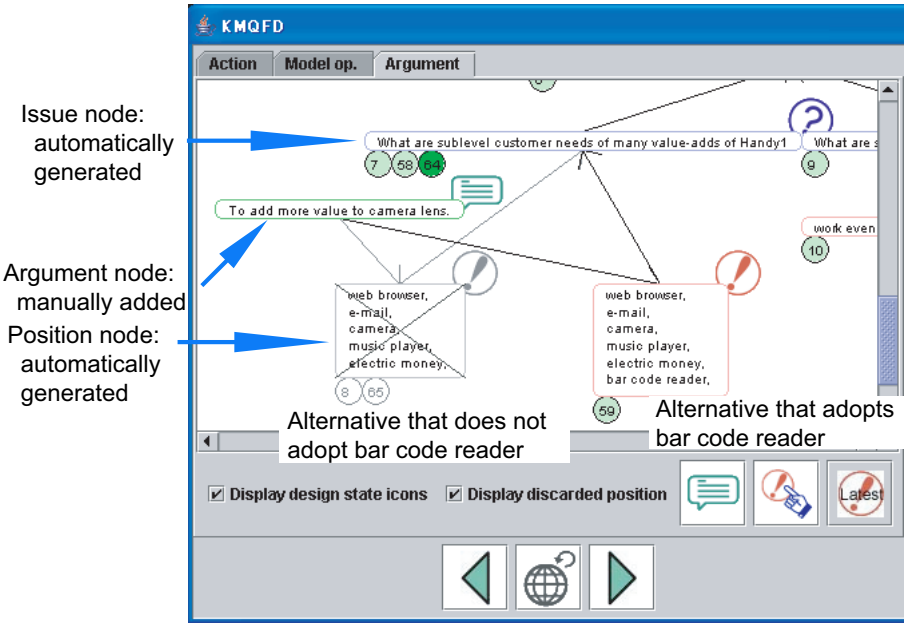

Fig. 7. A part of captured design process

An issue node (a node with a question mark) and a position node (a node with an exclamation mark) are automatically generated. A crossed node is a rejected position. An argument node (a node with a words balloon mark) is added manually by a designer to explain the branch of position nodes. In Figure 7, two positions are suggested for an issue of detailing a customer need for many value-adds, and a position that adopts bar-code reader is active now. Since all operations and all design states are recorded under a truth maintenance system, a designer can review discarded positions at any time for evaluating alternatives if he/she wants to review any of them again, and he/she can go back to any former design state. Such functionality facilitates designer's reflective refinement of alternatives over the prototype system through making full use of related methodologies.

5 Conclusion

This paper briefly introduces a DRIFT system, which is a new framework to capture reflective design process of practical products, and demonstrates an example of DRIFT-based implementation for QFD-based design operations. It helps a designer acquire dynamic knowledge through reflection-in-action over an ontology of DFX. Because any design stage contains reflective refinement of alternatives, the same expansion can be done for the other DFX methodologies and the other design methods when their ontologies and design operations are integrated to DRIFT. This expansion is included in our future works.

References

1. Schön, D. A.: The Reflective Practitioner - How Professionals Think in Action. (1982) Basic Books Inc.
2. Nomaguchi, Y., Ohnuma, A. and Fujita, K.: Design rationale acquisition in conceptual design by hierarchical integration of action, model and argumentation. Proc. DETC'04 ASME 2004 Design Engineering Technical Conf. and Computers and Information in Engineering Conf. (2004) DETC2004/CIE-57681
3. Nomaguchi, Y. and Fujita, K.: Ontology building for design knowledge management systems based on patterns embedded in design-for-X methodologies, Proc. of 16th International Conf. on Engineering Design (ICED '07). (2007) Paper No. 442
4. Kitamura, Y., Washio, N., Koji, Y., Sasajima, M., Takafuji, S. and Mizoguchi, R.: An ontology-based annotation framework for representing the functionality of engineering devices. Proc. DETC'06 ASME 2006 Design Engineering Technical Conf. and Computers and Information in Engineering Conf. (2006) DETC2006-99131
5. Grosse, I. R., Milton-Benoit, J. M. and Wileden, J. C.: Ontologies for supporting engineering analysis models. AI EDAM. (2005) **19**(1) 1–18
6. Witherell, P. Krishnamurty, S. and Grosse, I. R.: Ontologies for supporting engineering design optimization. Journal of Computing and Information Science in Engineering. (2007) **7**(2) 141–150
7. Dixon, J. R.: On research methodology towards a scientific theory of engineering design. AI EDAM. **1**(3), (1987) 145–157
8. Clausing, D.: Total Quality Development. A Step-By-Step Guide to World-Class Concurrent Engineering. (1994) ASME Press
9. Doyle, J.: A truth maintenance system. Artificial Intelligence. **12**(3) (1979) 231–272
10. Kunz, W. and Rittel, H.: Issues as elements of information systems. Working Paper No. 131, University of California, Berkeley, Institute of Urban and Regional Development. (1970)
11. Fujita, K. and Nishikawa, T.: Value-adds assessment method for product deployment across life stages through quality function deployment. Proc. 13th International Conf. on Engineering Design - ICED '01, Design Methods for Performance and Sustainability. (2001) 405–412