# Chapter 2
# Product Design Knowledge Management Based on Design Structure Matrix

## 2.1 Introduction and Synopsis

Knowledge Management (KM) is currently receiving increasing attention in diverse areas. The concept of KM can be defined in the way to organize information or data. It is possible to think in the evolution from data to knowledge in the following way: data → information → knowledge [1]. It is a common sense that the past experience and information may play a crucial role in a new product development process. The main objective of KM could be understood to capture and store all the past experience and information, and make them usable in assisting the development of new products or in the enhancement of existing ones. On one hand, knowledge can be seen as a representation of the real world, on the other, it can be conceptualized as a product of the interaction between individual cognition and reality [2]. Ensuring efficient knowledge capture, sharing and reuse to support new product development, it is important to develop a structured way of processing the past experience and information behind the elements interactions.

As a structured modeling method, more recently the design structure matrix (DSM) model has been regarded as a good roadmap of system knowledge [3]. The process of making DSMs involves understanding and capturing the relation between system elements, which is a process of knowledge capture. DSM therefore is an excellent delivery medium of knowledge and as well as an arena for knowledge creation and management. Comprehensive research and development requirements along with a cogent framework, however, have not been proposed for integrating KM and DSM. This chapter tries to consolidate the KM method based on DSM and identify research and development issues in a unifying manner.

The remainder of the chapter is organized as follows. Section 2.2 reviews related research results. Section 2.3 proposes a comprehensive framework to illustrate KM lifecycle based on DSM. Section 2.4 focuses on how to create and capture both

© Springer Nature Singapore Pte Ltd. 2018

D. Tang et al., *Matrix-Based Product Design and Change Management*,
DOI 10.1007/978-981-10-5077-0_2

direct and indirect knowledge from different kinds of DSMs. Section 2.5 introduces how to organize and use the captured knowledge from DSMs. Section 2.6 concludes the chapter.

## 2.2 Related Research of Knowledge Management

It is widely observed that the society we live in has been gradually turning into a "knowledge society", and knowledge management has now been a critical success issue. It has been discovered that inventive problems in completely different areas have been solved with similar principles. This assists in stating why is so important to maintain a well-organized knowledge. In any development process, problems arisen, past experience and information may play a crucial role, to help find a solution in less time. In reality, the product related knowledge is distributed across the whole company, and this kind of know how represents an essential resource for successful competition in the market and should therefore be preserved and used as efficiently as possible. The pressure to shorten the time to market for new products and therefore to shorten development time requires an adequate management of knowledge possessed by a company. Normally, a knowledge management effort includes knowledge eliciting and recording. The resulting knowledge storage are intended to be live, up to data documents synthesizing the corporate expertise in the form of standards, rules, procedures, and best practices. In addition, knowledge management enables the recorded expertise becoming more easily reviewed, shared, and used [4–6].

Knowledge management can be conducted at different levels such as part or individual level, assembly level, and system level. Based on the interview at industries, Dong finds a trend that as design scale gets bigger, less knowledge is documented [3]. In other words, the knowledge capturing and recording is more poorly implemented at the system level than part or individual level, especially for the large-scale complex system. With part or individual knowledge, each person sees only part of the entire system, which could cause inconsistent understanding problems in project development. Although ideas are formed in the minds of individuals, interaction between individuals typically plays a critical role in developing these ideas. That is, communities of interaction would contribute to the amplification and development new knowledge.

System level knowledge can be defined as the knowledge concerning system interactions. Knowing system interactions in the product development process is critical for project management and decisions. System interactions mainly refer to the inter-relationships existing between element types and between element types during the product development cycle. Capturing inter-relationships between affiliated system elements is necessary if the product engineering is to be handled (i.e. analyzed, engineered, documented, developed, and managed) efficiently and effectively. These elements can be many things: physical components that make up the product, performance attributes expected by the consumer, engineering

requirements the product must meet, or tasks that create the product. Each of these different element types contributes to the development of a product. The idea of using DSM to represent the system interaction is not new. DSM method was proposed for system modeling and analysis more than 30 years ago. Steward firstly introduced DSM and developed some algorithms for manipulating the matrix as tools for systems design and analysis [7, 8]. However, it is not until recently that DSM methods started to attract attention for managing the complexity of large engineering systems and complex product development processes [9–13].

Most of the aforementioned DSM-based researches provide some insight into the interaction structure of a complex system or process (i.e. a dependency map); however, they fail to expose and explore the logic behind these dependencies. That is, if an element B is shown to interact with another element C, it is not clear why and how this relation exists and where it stems from [13]. Meanwhile, most of current DSM research is focused on the partitioning algorithms for system re-engineering, while the knowledge behind the DSM re-engineering is worthy of paying attention.

To conclude, system level knowledge is critical to the success of the design of large-scale systems, yet is often missing due to its empirical nature. The knowledge concerning system interactions requires the identification and understanding of the inter-relationships among the different elements. DSM, a structured method, has been proved to be most suitable for not only capturing system level interactions, but also providing analysis to the system interactions to aid the design and management of the system. However, how to deal with the system knowledge management based on DSM still deserves extensive research.

## 2.3 KM Lifecycle Framework Based on DSM

According to the KM lifecycle, a DSM-based knowledge management framework is proposed in Fig. 2.1. As an input of the knowledge management lifecycle, the system and its scale need to be defined at first. Following tasks are suggested to finish the KM lifecycle:

1. Knowledge acquisition

Identify and capture the knowledge used in the design process for solution, documentation and storing.

- *System interaction study*

List all system elements and study the information flow between them. The information flow can be got from the design document and interview with experienced designers.

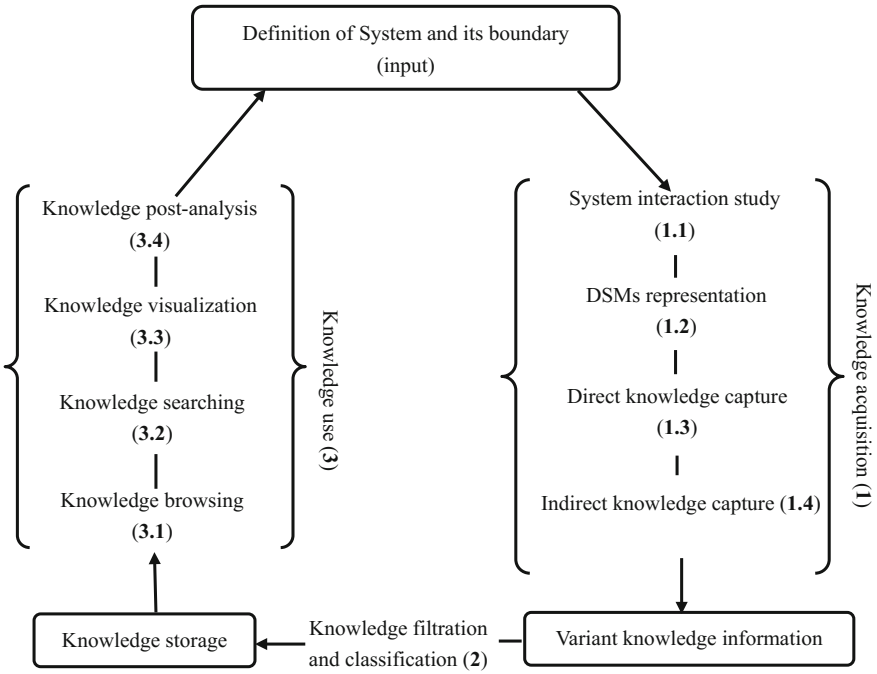



**Fig. 2.1** DSM based KM lifecycle framework

- *DSMs representation*

Build binary matrices to represent the information flow between system elements in product, process and organization domains.

- *Direct knowledge capture*

Focus on the direct interactions between system elements, extract the know-how behind them.

- *Indirect knowledge capture*

Focus on the indirect interactions among system elements, analyze the interaction source and trace the know-how behind them.

2. Knowledge organization

Filter the valuable knowledge and classify type of knowledge. This is done to support efficient storing and reuse of the formalized knowledge (algorithms, rules, relations and data).

3. Knowledge use

By retracing the design history knowledge, the designer is able to find the design decision, rational and related information that govern the existing product and from where new design could originate.

- *Knowledge browsing*

  Using DSM as a knowledge browser, the designer could locate his interested information conventionally.

- *Knowledge searching*

  Using some key words, the designer could obtain the related knowledge rapidly.

- *Knowledge visualization*

  Use graphic model to transparently represent the result of knowledge capturing.

- *Knowledge post-analysis*

Account the reuse frequency and clarify the quality of captured knowledge. Evolve and consummate the system knowledge based on the evaluation of designer.

## 2.4 Knowledge Acquisition

### *2.4.1 Direct Knowledge Capture from DSMs*

Development of complex products and large systems is a highly interactive social process involving hundreds of people designing thousands of interrelated components and making millions of coupled decisions. Generally, the complexity of a large system can be modeled through three domains of product development: a product domain, process domain, and an organization domain, which are explained as follows [14].

1. Product: A complex product or large system is decomposed into sub-systems, and these in turn may be further decomposed into sub-assemblies and/or components.
2. Process: A full development process is decomposed into phases or sub-processes, and these in turn may be further decomposed into tasks or activities.
3. Organization: A large development organization is decomposed into teams, and these in turn may be further decomposed into working groups and individual actors.

It is able to learn about the complex social phenomenon of product development by studying the patterns of interaction across the decomposed elements within each

domain [14]. DSM is always regarded as the suitable tool to describe the interactions and document the decomposition of each of the three domains.

To construct a DSM, one assigns the individual system elements of a domain to the rows and columns of a square matrix. Then, going down the list, if element *b* is an input to element *c*, one puts a mark in the column of *b* and the row of *c*. One continues until all (known) hierarchical and interdependent relationships have been accounted for. Marks below the diagonal represent forward information transfer which affects the later element. Marks above the diagonal represent information fed back to earlier element (see Fig. 2.2).

It is not enough to know only the form of DSM, it is also necessary to know the meaning of its marks and the knowledge inside. In a DSM, the off-diagonal marks are not simply placeholders. Corresponding to each is a specific real issue, involving at least two system elements. Thus the matrix understates the depth of the designers' knowledge for them; each of the marks stands for a specific problem or question.

Things are traceable, if they leave traces. The system level knowledge can be traced through the element interactions in DSM which can be regarded as a knowledge map. In other words, the system knowledge can be depicted by the marks in a DSM, and interpreted by the syntactic and semantic information of the element interactions.

A single-domain DSM can only reflect the system knowledge in an isolated area. For example, through a product DSM, we can directly know the physical and logical interdependencies among product elements (such as components). To capture the system level knowledge from a comprehensive view, we need to analyze the combined DSMs of three domains defined above. Generally, it can be assumed that product DSM, process DSM, and organization DSM are isomorphic to each other. That is, if component *j* either imposes geometry constraints or transfers forces, material, energy, and/or signals to component *i*, for component *i* to function

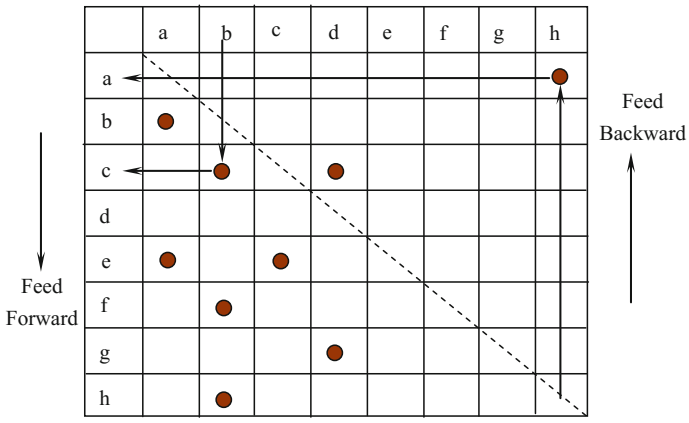



**Fig. 2.2** DSM with feed forward information and feedback information

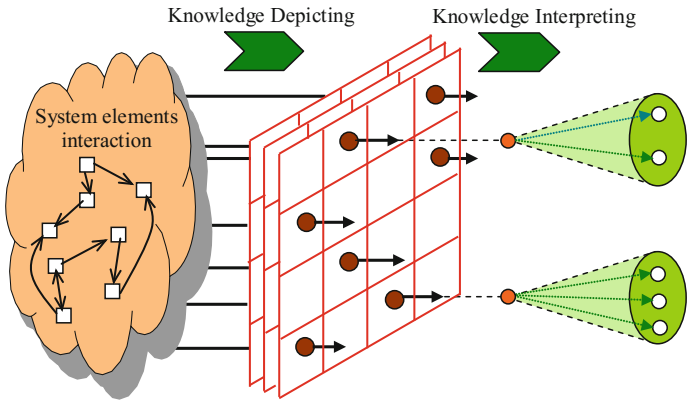



**Fig. 2.3** Direct knowledge capture from isomorphic three-domain DSMs

properly, technical information will flow from team *j* (activity *j*) to team *i* (activity *i*). Therefore, linking the same-position marks in the isomorphic DSMs (see Fig. 2.3), the comprehensive system knowledge can be depicted and interpreted by answering following questions:

- **What** are the product elements—design parameters, components, design representation objects, etc.—that are managed during product design and development, what are their characteristics and what are the links between them?
- **Who** are the actors or teams that play different roles in the creation, maintenance and use of those product elements and links?
- **Where** are the product elements represented: in physical media such as documents, drawings, spreadsheets or other files, or in less tangible things such as telephone calls, discussions, emails, meetings or people's brains?
- **How** can the product elements realized by different tasks or activities?
- **Why** are the product elements created, modified or evolved?
- **When** (in which lifecycle) are the product elements created, modified or evolved?

Figure 2.4 further illustrates the syntactic and semantic meaning of direct captured knowledge from three-domain DSMs which are isomorphic.

Under the isomorphic context, it is easy to understand that if two components have interactions, the teams (tasks) that design them also need to interact. However, in the development of highly complex products, it would be naive to expect a perfect mapping between three-domain DSMs. That is, due to product ambiguity and complexity, defined as the absence of knowledge about product variables and/or their interfaces, some product design interfaces or interactions are not foreseen at the outset of the project and are only discovered after design teams' work on the systems themselves [14, 15]. In other words, most of the product architectural knowledge is explicit and known by development organizations, while some interfaces or interactions between components are unspecified (or even

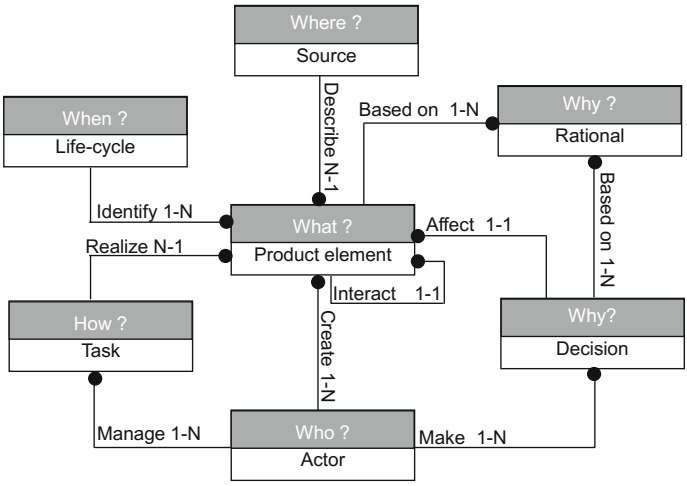



**Fig. 2.4** Syntactic and semantic meaning of direct captured knowledge from isomorphic three-domain DSMs

unknown) and only identified or documented during the design process itself. It then becomes important to determine where (in the product) those unidentified interfaces or interactions are likely to be, and how they can be uncovered. Therefore, this kind of absent knowledge could be captured through the misaligned mapping between three-domain DSMs (see Fig. 2.5).

Based on the captured knowledge above, we could discover the design history of every feature of a product, and make the past product development traceable. The

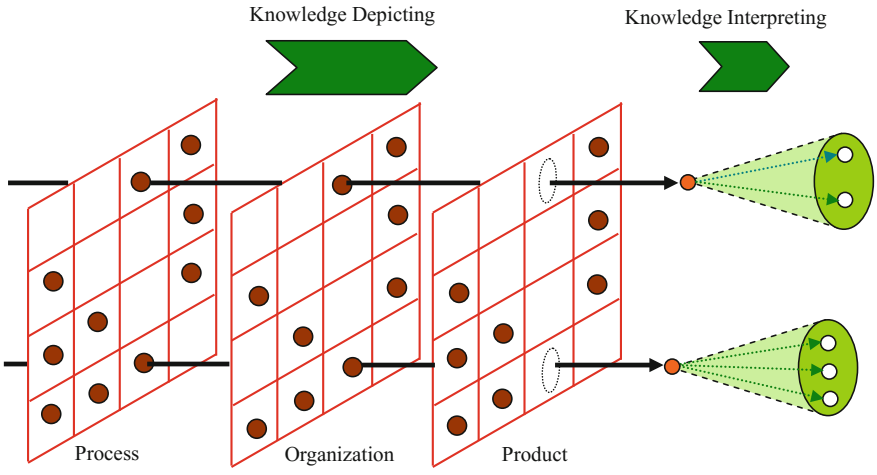



**Fig. 2.5** Knowledge capture through misalignment between three-domain DSMs

knowledge can be referred to the traceability defined in [16], and enable tracing the product development activities, design routes, designers' actions, decisions, reasoning, events, etc. The three-domain DSMs could be knowledge browsers to navigate the evolution of design items, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its deployment and realization, and through periods of on-going refinement and iteration in any of these phases. The DSM-based knowledge also can help predict the impacts of design changes through tracing designed items back to their sources. In addition, the captured knowledge can improve understanding of the design routes by linking designed items to rationales, decisions and assumptions behind them. Meanwhile, the knowledge assists on simulating design episode in another situation: for performing changes on existing solutions, reusing of the existing solutions in new projects, configuration of the new variant of the product, and educational process for inexperienced designers.

### 2.4.2 *Indirect Knowledge Capture*

In the above section, DSMs act as knowledge savers through analyzing direct interactions between system elements. However, just as shoals can exist hidden in the ocean, indirect interdependencies can exist [17]. As shown in a DSM in Fig. 2.6, there is a direct interaction from component $a$ to component $b$, while there are two indirect component interactions "$a \rightarrow d \rightarrow b$" and "$a \rightarrow f \rightarrow b$". Meanwhile, in some instances, we use the concept of indirect team interaction to hypothesize that team $a$, whose component has a design interface with component $b$, may not report direct interaction with team $b$ because it interacts with an intermediary team (team $k$, which also interacts with team $j$) which passes the information (to team $a$) that would otherwise have flowed directly from team $b$ to team $a$. Such kind of indirect interdependencies or interactions can be fatal or merely annoying, require extra work. Or they can be fortunate, revealing new knowledge of design, which can be called indirect knowledge. Indirect interdependencies or interactions are one of the primary sources for uncertainty in design processes.

In a DSM, each of the off-diagonal marks may require actions to deal with the related questions or problems. In many instances, there would be tradeoffs and conflicts, some of which would require a costly and time-consuming dispute

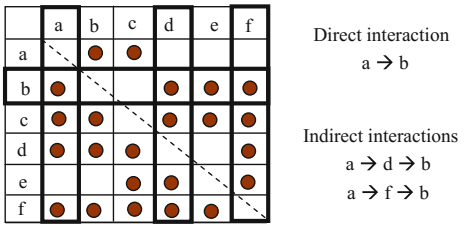



**Fig. 2.6** Direct and indirect interaction in DSM (modified based on [18])

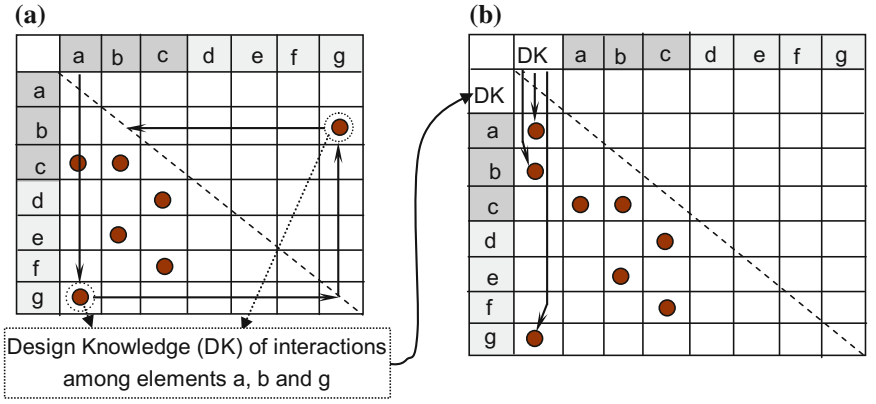



**Fig. 2.7** **a** As-is DSM, **b** rationalized DSM with design knowledge

resolution process [17]. Especially the marks above the diagonal could cause process iteration or natively affect the modular product architecture. Researchers have invented various partitioning algorithms to re-engineer the DSM and eliminate the marks above the diagonal. However, the algorithms can only provide context-free analysis of the DSM, what's the knowledge behind them is still unknown. The essential knowledge for re-engineering the DSM is to identify the iteration's cause and then eliminate it. To capture this kind of knowledge, it is required to study each iterative block separately which is caused by indirect interactions and identify reasons for iteration.

For example, a fragment of a cycle depicted in Fig. 2.7a shows the indirect interactions among elements *a*, *b*, and *g* which cause serious iteration problem. If they have experience with previous designs, the designers will know that the problem is certain to come up. They will also have knowledge about what was done in previous designs, and how the resulting artifacts performed. This knowledge gives the designers a set of preformed strategies to eliminate the iterations. In particular, many of the routes to failure will be known, and presumably do not need to be tried again. Therefore, a rationalized DSM can be set up shown in Fig. 2.7b with the design knowledge of interactions among elements *a*, *b* and *g*, which is useful for not only reducing unnecessary feedback or rework but also designing modular product architecture.

## 2.5 Knowledge Organization and Application

Through direct and indirect knowledge capture from DSMs, we can get various and unstructured knowledge such as personal experiences, situational decisions, learned facts by trial and error, etc. For efficient use and reuse of them, this knowledge needs to be organized by filtering, refining, classifying, and rearranging. In this

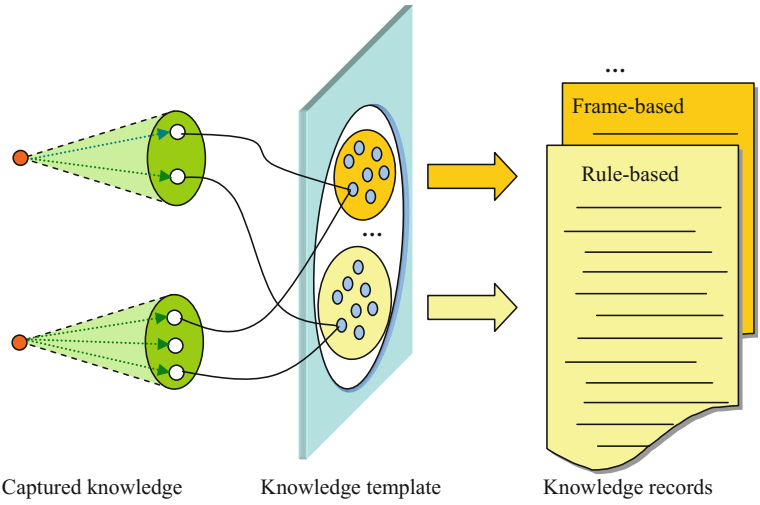



**Fig. 2.8** Knowledge organizations by knowledge template

research, the knowledge organization is conducted by a knowledge template. The knowledge template can provide the structure and meta information of knowledge type to enable designers filtering valuable knowledge and avoiding knowledge overload, and to ensure that all useful knowledge (innovative information) is saved. The knowledge organization process is shown in Fig. 2.8.

The knowledge template is defined according to the most popular ways of knowledge representation. Therefore one can differentiate the captured knowledge into typical representations, such as *rule*-based (knowledge in terms of facts and rules for manipulating facts), *frame*-based (a frame, concept, schema or unit as a record like structure, a form for encoding on a stereotyped situation; associated with a frame is a set of attributes, the descriptions or values of which are contained in slots) and *semantic net*-based (semantic nets are similar to frames; it is a network or graph of nodes linked together by arcs, arcs represent relations) representations. After filtration and classification, the different types of knowledge records are stored with sequence number for future reuse. Although each type of knowledge has a distinct purpose, together they can support knowledge-intensive activities throughout the process lifecycle.

It is possible to store knowledge in all kinds of ways, but this does not make it real for people who will use it. In this research, a prototype has been developed to enable the knowledge browsing, searching and visualization. Taking a hairdryer product as an example, Fig. 2.9 shows a screen of knowledge browsing. The user can navigate the DSM inner cells to check the characteristics of the product architecture. Figure 2.10 gives another screen about the interaction knowledge searching through defining the level of interaction (strong, medium and weak), context of interaction (physical connectivity, cost, time, etc.) and the milestone of interaction (conceptual design, embodiment design, detailed design, process

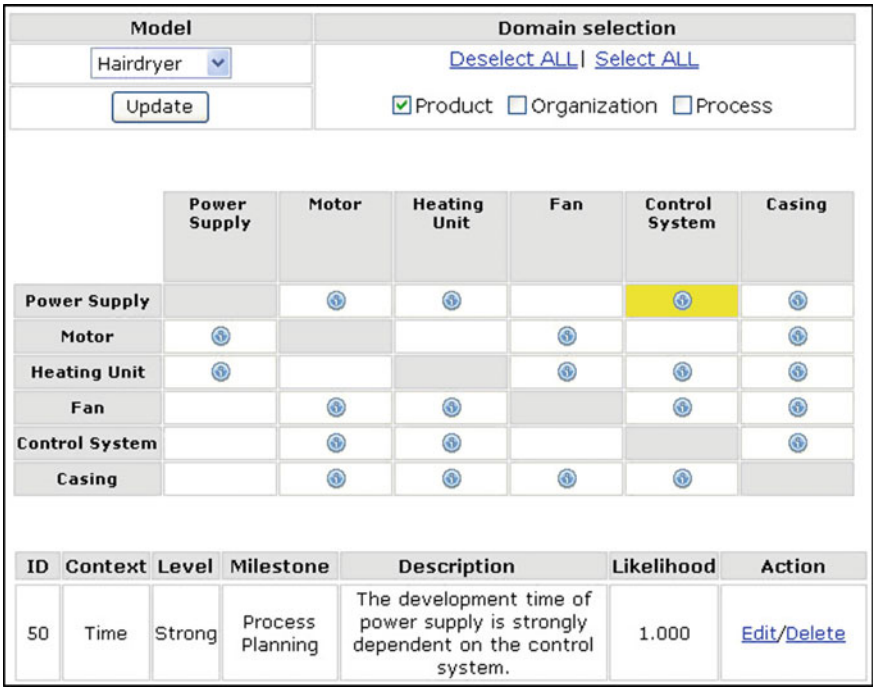



**Fig. 2.9** Knowledge browsing

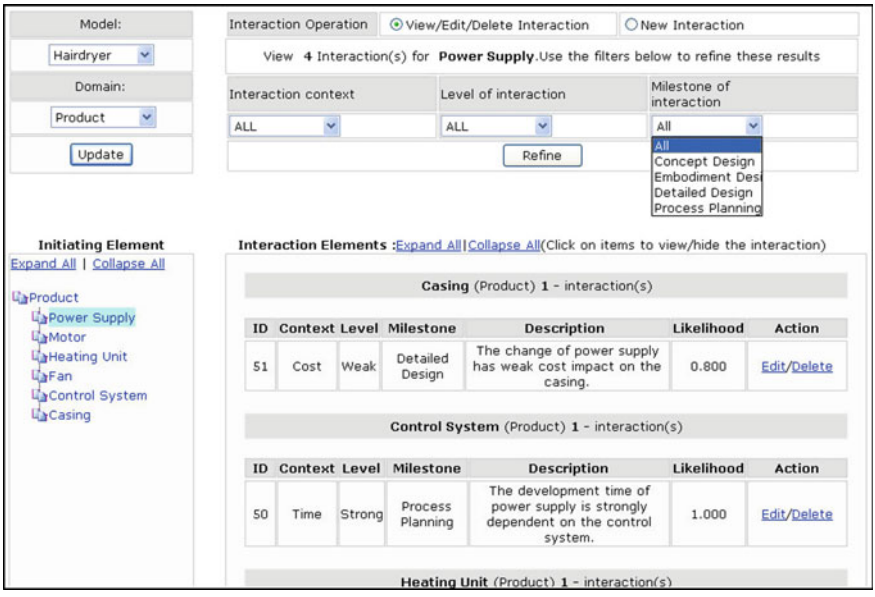



**Fig. 2.10** Knowledge searching

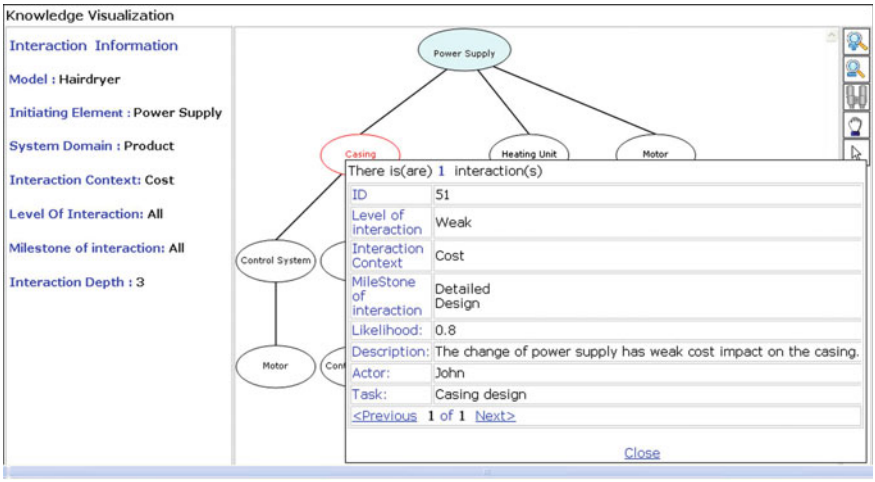



**Fig. 2.11** Knowledge visualization

planning, etc.). To be able to perform a better display of system interaction knowledge, a knowledge visualizing function has been realized in the prototype (see Fig. 2.11). Through the visualization result, the designer can easily grasp the overall situation of the system knowledge, and could conduct the change propagation analysis for better decision making.

## 2.6 Conclusion

The system level knowledge management concerning system interactions is often missing due to its empirical nature. In this chapter, DSM is proposed to be suitable means to define, capture and organize and distribute the system level knowledge. How to capture the direct and indirect knowledge through DSMs is presented in detail.

Clearly more work remains to be done. The knowledge discussed in this chapter mainly refers to the explicit knowledge. How to capture the implicit knowledge through DSMs is still an open question. With the knowledge captured, how designers simulate and predict the impact of a design change is another future work.

## References

1. Sorli M, Stokic D, Gorostiza A et al (2006) Managing product/process knowledge in the concurrent/simultaneous enterprise environment. Robot Comput Integr Manuf 22:399–408
2. Krogh GV (1998) Care in knowledge creation. Calif Manag Rev 40(3):133–153

3. Dong Q (2002) Predicting and managing system interactions at the early phase of product development process. PhD thesis, Massachusetts Institute of Technology
4. Mili F, Shen W, Martinez I et al (2001) Knowledge modeling for design decisions. Artif Intell Eng 15:153–164
5. Cheung WM, Bramall DG, Maropoulos PG et al (2006) Organizational knowledge encapsulation and re-use in collaborative product development. Int J Comput Integr Manuf 19(7):736–750
6. Jung J, Choi I, Song M (2007) An integration architecture for knowledge management systems and business process management systems. Comput Ind 58:21–34
7. Steward D (1962) On an approach to the analysis of the structure of large systems of equations. SIAM Rev 5:321–342
8. Steward D (1981) The design structure system: a method for modeling the design of complex systems. IEEE Trans Eng Manag 3:71–74
9. Eppinger SD (1991) Model-based approaches to managing concurrent engineering. J Eng Des 2:283–290
10. Eppinger SD, Whitney D, Smith R et al (1994) A model-based method for organizing tasks in product development. Res Eng Des 6:1–13
11. Yassine A, Falkenburg D (1999) Design process specifications management. J Eng Des 10:223–234
12. Tang D, Zheng L, Li Z et al (2000) Re-engineering the design process for concurrent engineering. Comput Ind Eng 38:479–491
13. Yassine A, Whitney D, Eppinger S et al (2003) Connectivity maps: modeling and analysing relationships in product development processes. J Eng Des 14(3):377–394
14. Eppinger SD, Salminen V (2001) Patterns of product development interactions. In: International conference on engineering design, ICED 01, Glasgow, August 21–23
15. Soe ME, Eppinger SD, Rowles CM (2004) The misalignment of product architecture and organizational structure in complex product development. Manag Sci 50(12):1674–1689
16. Štorga M (2004) Traceability in product development. In: International design conference—design 2004, Dubrovnik, May 18–21
17. Baldwin CY, Clark KB (2000) Design rules: the power of modularity, vol 1. The MIT Press, London
18. Clarkson PJ, Simons C, Eckert C (2004) Predicting change propagation in complex design. J Mech Des ASME Trans 126:788–797

http://www.springer.com/978-981-10-5076-3

Matrix-based Product Design and Change Management
Tang, D.; Yin, L.; Ullah, I.
2018, VIII, 208 p. 133 illus., 99 illus. in color., Hardcover
ISBN: 978-981-10-5076-3