ELSEVIER

Knowledge-Based Systems 11 (1998) 449–456

Knowledge-Based SYSTEMS

Supporting the acquisition and modeling of requirements in software design

Yasuyuki Sumi[a,*], Koichi Hori[b], Setsuo Ohsuga[c]

[a]*ATR Media Integration and Communications Research Laboratories, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*
[b]*Research Center for Advanced Science and Technology, The University of Tokyo, Komaba, Meguro-ku, Tokyo 153-8904, Japan*
[c]*School of Science and Engineering, Waseda University, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan*

Received 28 July 1998; accepted 17 August 1998

Abstract

This paper describes a system for supporting the construction of software *requirement models*, which are initial computable models representing users' requirements in software design. The system principally consists of two components, a system for aiding the formation of requirement concepts by visualizing a user's thought space, and a knowledge-based system which automatically assembles the ascertained requirement concepts into a requirement model. The system extracts reusable components of a requirement model, corresponding to the users' abstract requirement concept, from a store of similar past cases. The components are then automatically arranged using heuristic reasoning. By using the system, users can make their requirement concepts more mature, and simultaneously get computable requirement models as by-products. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Modeling; Software requirements acquisition; Supporting idea formation

1. Introduction

In this paper, we propose a system for aiding the construction of *requirement models* in software design.

Numerous studies on intelligent support for software development, such as automated programming systems [1, 2], have been conducted. This has caused the main focus of software development activities performed by humans to shift to the description of *what* computers are required to solve as opposed to *how* computers are to solve a particular problem [3]. At the same time, because of the need to apply computer systems to a wide variety of domains, it has become essential to integrate target domain experts, i.e. the systems' users, along with computer experts, into the process of determining specification of requirements. However, it is very difficult to ascertain a system's complete requirements from users who are not experts in software development, let along transform these into formal specifications in the software design. Therefore, an intelligent supporting environment for establishing the specifications of requirements is currently being actively sought [4].

Hori and Ohsuga [5] highlighted the importance of aiding the articulation of requirements in the intelligent support for software development. Consequently, they proposed that software development should be considered a mediation of two self-organizing worlds, that is, the metal world, where the user formulates his/her ideas, and the computational world, where he/she formalizes the ideas. The organization of both worlds should proceed concurrently. Consequently, an environment that aids both the articulation of its users' requirements and the structuring of those requirement concepts into a computable model concurrently would be indispensable.

In this paper, we propose a system for aiding the construction of a software requirement model through the integration of two techniques, that is, a method for the support of idea formation and knowledge-based automated modeling. The former aids users in articulating their requirements, while the latter automatically structures these requirement concepts into a tentative requirement model to be interpreted later by a computer. The system extracts reusable components of a requirement mode, corresponding to the users' abstract requirement concept, from a store of similar past cases. The components are then automatically arranged using heuristic reasoning.

* Corresponding author. Tel.: + 81-774-95-1401; Fax: + 81-774-95-1408.

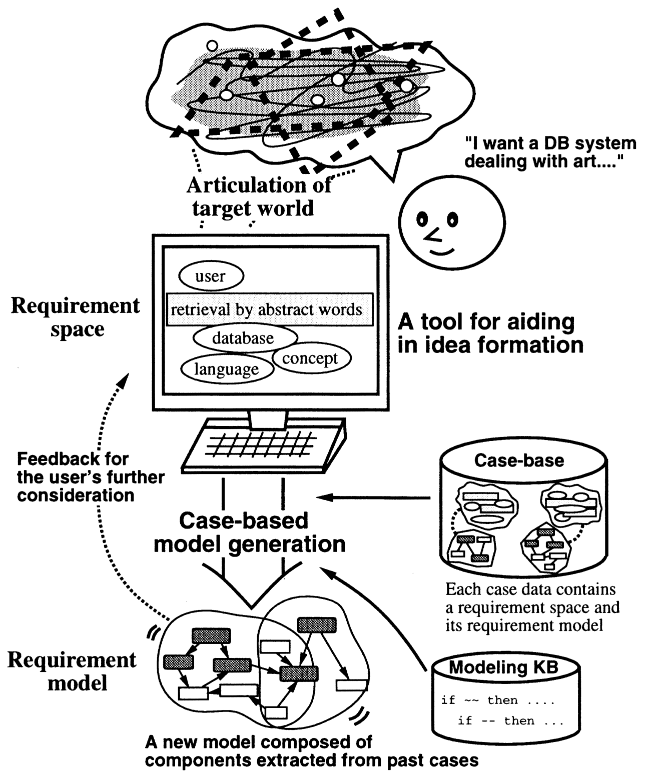

"I want a DB system dealing with art...."
Articulation of target world
Requirement space
user
retrieval by abstract words
database
concept
language
A tool for aiding in idea formation
Feedback for the user's further consideration
Case-base
Case-based model generation
Each case data contains a requirement space and its requirement model
Requirement model
Modeling KB
if ~~ then ....
if -- then ...
A new model composed of components extracted from past cases

Fig. 1. The architecture of a system for aiding in the construction of a requirement model in software design.

2. System overview

Fig. 1 illustrates the architecture of the system proposed in this paper. The intended user for the system is someone researching advanced information engineering, and who wants to build a software prototype related to his/her emerging ideas. As such, the intended user need not have established concrete requirements for the software, and the target domain need not be well-structured. The user can write down fragments of his/her software requirements on memos and build his/her requirement space using an idea formation support tool. This tool should be able to visualize the requirement space, to aid in the analysis and acquisition of appropriate requirements. Consequently, the knowledge-based system generates a tentative requirement model according to the current requirement space by exploiting such knowledge as past software designs. This enables the user to recognize and analyze his/her own requirement concepts as a structured model. The user is also able to modify the requirement space using the tool, and observe all changes in the requirement model caused by the modification. Repeating this process makes the requirement concepts mature, and simultaneously leads to a by-product, that is, a computable requirment model.

The resulting pairs of the requirement space and its model are added to the case-base for later use. Namely, the case-base in the proposed system matures according to its users and fields. To increase the reusability of products, in not only the lower stream of software development, but also in the upper one, claims have been made stating that domain-dependent analysis and modeling are necessary [6]. The method proposed in this paper can serve as an answer to these claims.

A brief explanation follows for the *requirement space* and *requirement model* as proposed for this system.

- A requirement space is a metric space which enables a user to visualize requirement concepts and their relationship to the user's origanal ideas, mainly consisting of abstract and domain-dependent expressions. This space also plays an important role in mediating between the well-structured world of computer languages and the abstract world of concepts.
- A requirement model is an initial software model, which aids in the formulation of a user's requirements to succeeding phases such as details specification and implementation. This model is expressed as a network of *entities*, and *relationships* among the entities; each relationship object has a frame structure with several slots and fillers, i.e. entities. Such a requirement model represents a target world wherein the user's problems and requirements lie, in addition to containing functions or data structures which are expected to be implemented.

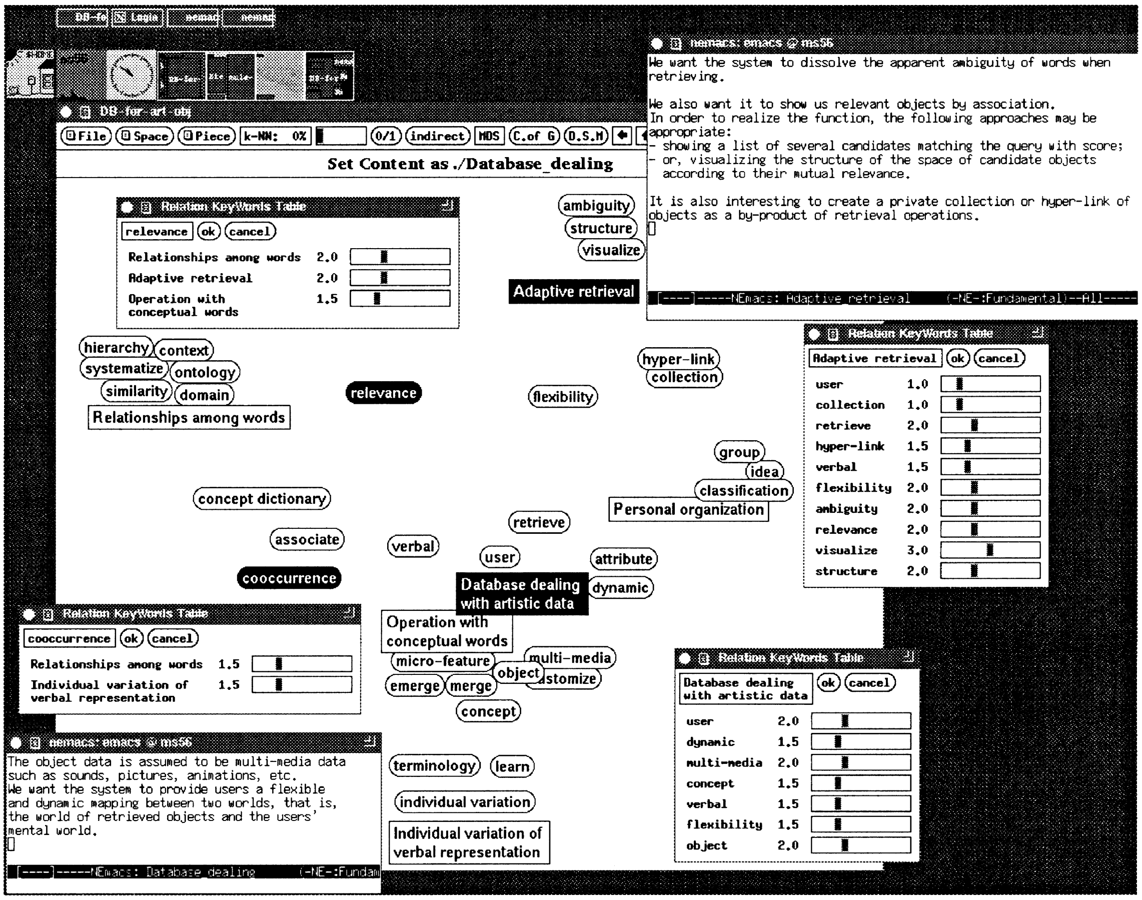
3. Forming ideas as a requirement space

The authors' research group has proposed several computer tools for aiding idea formation by visualizing snapshots of the topological structure of a user's thought space[1] using statistical methods [7–10]. These tools have been successfully applied to personal idea formation, knowledge acquisition, human communications support, information retrieval, and so on.

The system proposed in this paper employs one of these, namely CSS (communication support system) [10]. CSS visualizes the structure of a user's thought space by automatically mapping electronic memos, called *text-objects*, into a two-dimensional metric space according to the number of common *keywords* declared in the text-objects. The structure of the space is statistically determined by a set of text-objects with weighted keywords as multivariate data: two axes of the space correspond to two principal eigenvectors of the data. Intuitively, in the space, a pair of text-objects with more common keywords is located closer together and these keywords are mapped around the pair.

Fig. 2 shows an example of CSS being used in forming ideas for the development of new software. The main window in Fig. 2 depicts the metric space, which we call the requirement space in this paper. The rectangular icons in the space indicate text-objects containing requirements

[1] Here, we define a *thought space* as a mental space consisting of fragments of ideas or knowledge and the relationships among them in a thought process.



Relation KeyWords Table

relevance (ok) (cancel)

Relationships among words	2.0
Adaptive retrieval	2.0
Operation with conceptual words	1.5

nemacs: emacs @ ms56

```
We want the system to dissolve the apparent ambiguity of words when
retrieving.

We also want it to show us relevant objects by association.
In order to realize the function, the following approaches may be
appropriate:
- showing a list of several candidates matching the query with score;
- or, visualizing the structure of the space of candidate objects
  according to their mutual relevance.

It is also interesting to create a private collection or hyper-link of
objects as a by-product of retrieval operations.

[----]-----NEmacs: Adaptive retrieval     (-NE-:Fundamental)--All-----
```

Relation KeyWords Table

Adaptive retrieval (ok) (cancel)

user	1.0
collection	1.0
retrieve	2.0
hyper-link	1.5
verbal	1.5
flexibility	2.0
ambiguity	2.0
relevance	2.0
visualize	3.0
structure	2.0

Relation KeyWords Table

cooccurrence (ok) (cancel)

Relationships among words	1.5
Individual variation of verbal representation	1.5

Relation KeyWords Table

Database dealing with artistic data (ok) (cancel)

user	2.0
dynamic	1.5
multi-media	2.0
concept	1.5
verbal	1.5
flexibility	1.5
object	2.0

nemacs: emacs @ ms56

```
The object data is assumed to be multi-media data
such as sounds, pictures, animations, etc.
We want the system to provide users a flexible
and dynamic mapping between two worlds, that is,
the world of retrieved objects and the users'
mental world.

[----]-----NEmacs: Database dealing      (-NE-:Fundam
```

Fig. 2. Example of using CSS to form a requirement space.

concepts, and the oval ones, their keywords. CSS does not restrict the content of texts into text-objects; hence, users can exploit informal documents written during routine research as well as the upper stream of software development, namely, research memos and notes pertaining to abstract ideas.

CSS is implemented with the X Window System on a UNIX workstation, and offers a graphical user interface with multiple windows. CSS manages a data table, which contains text-objects, keywords, and weight values given to pairs of text-objects and keywords. The designating of keywords and their weights is determined by the users. The users can change the data set according to their own thoughts. Whenever a user instructs CSS to reconfigure the space, it calculates and shows the new configuration according to the current data.

CSS enables users to reconsider their own ideas and requirements from a global perspective. Users can recognize several modules of their requirements by viewing clusters emerging in the space reconfigured by CSS, and grasp the topological relationships between these modules.

Since CSS can store an increasing number of results in its database, users can share and reuse them in the development of new ideas. Consequently, they can use CSS to visualize the conceptual structure of existing software. These 'past requirement spaces' can be used as tentative spaces at the beginning of developing new ideas by providing better organized spaces containing both concrete ideas as well as abstract ones. Moreover, such a CSS database can be a domain-dependent repository of collaborative work by groups of people.

4. Structuring requirement concepts into a computable model

4.1. Knowledge-based modeling approach

In the previous section, we briefly explained a method to ascertain requirements with CSS. In order to exploit and improve the requirements in the lower stream of software development, we need to transform them into a requirement model described in a computational manner.

Structuring requirement concepts into a requirement model is a repetitive process of model transformation which consists of *initial model building*, *model evaluation* and *model modification*, i.e. a typical process of non-deterministic problem solving [11]. It is rare for the ascertained requirements to match an earlier requirement model; hence, there is no alternative but to gradually improve a model to

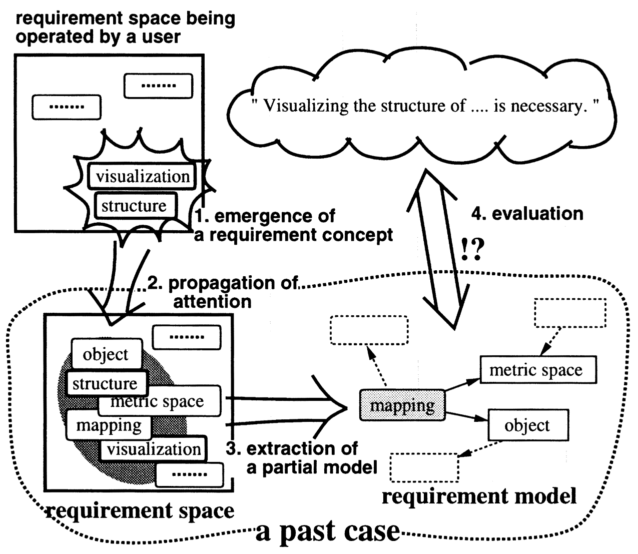

requirement space being operated by a user
" Visualizing the structure of .... is necessary. "
visualization
structure
1. emergence of a requirement concept
4. evaluation
!?
2. propagation of attention
object
structure
metric space
mapping
visualization
3. extraction of a partial model
metric space
mapping
object
requirement model
requirement space
a past case

Fig. 3. Extraction of a partial model matching a user's requirement concept from past cases.

meet the requirements, using a variety of knowledge, by trial and error.

In this paper, we mainly focus on the automation of generating the initial model prior to the repetition of model evaluation and modification. Namely, the goal is to automate the construction of a prototype as a tentative model for analysis, evaluation and modification, in order to arrive at a requirement model meeting the requirement concepts externalized in the requirement space of CSS.

We adopt a knowledge-based approach exploiting the information from previous cases, namely, pairs of a requirement space and its requirement model produced in previous cases of software development. The knowledge-base stores various kinds of modeling knowledge which are utilized for the prototyping and assessment of requirement models to meet the users' requirement spaces. We perform implementation to describe the task knowledge for prototyping the models as well as the operational knowledge for evaluating them such as: model generation, the detection of unnoticed relationships between entities in models belonging to different cases, the extraction of parts from a model based on certain viewpoints, and the verification of semantic constraint satisfaction of models. Among them, we especially describe the task knowledge for model generation later.

To describe and utilize the knowledge, we employ a knowledge-processing system called KAUS (knowledge acquisition and utilization system) [12] which was developed at the University of Tokyo, Japan. KAUS is a declarative language for knowledge representation and reasoning, which is suitable for describing and utilizing structured information. In this study, we use KAUS for: (1) the description, management and operation of a tentative model inferred and previous models in the case-base; (2) the description and utilization of modeling knowledge; and (3) an inference engine for modeling.

To provide users with tentative models that correspond to their requirement spaces, we implement the following automated tasks for model generation:

- Extracting, from past requirement models, components of a requirement model that may meet requirement concepts as they emerge in a user's requirement space.
- Composing a tentative requirement model, by selection and modification, that corresponds to the entire requirement space from those parts.

The following sections describe these tasks.

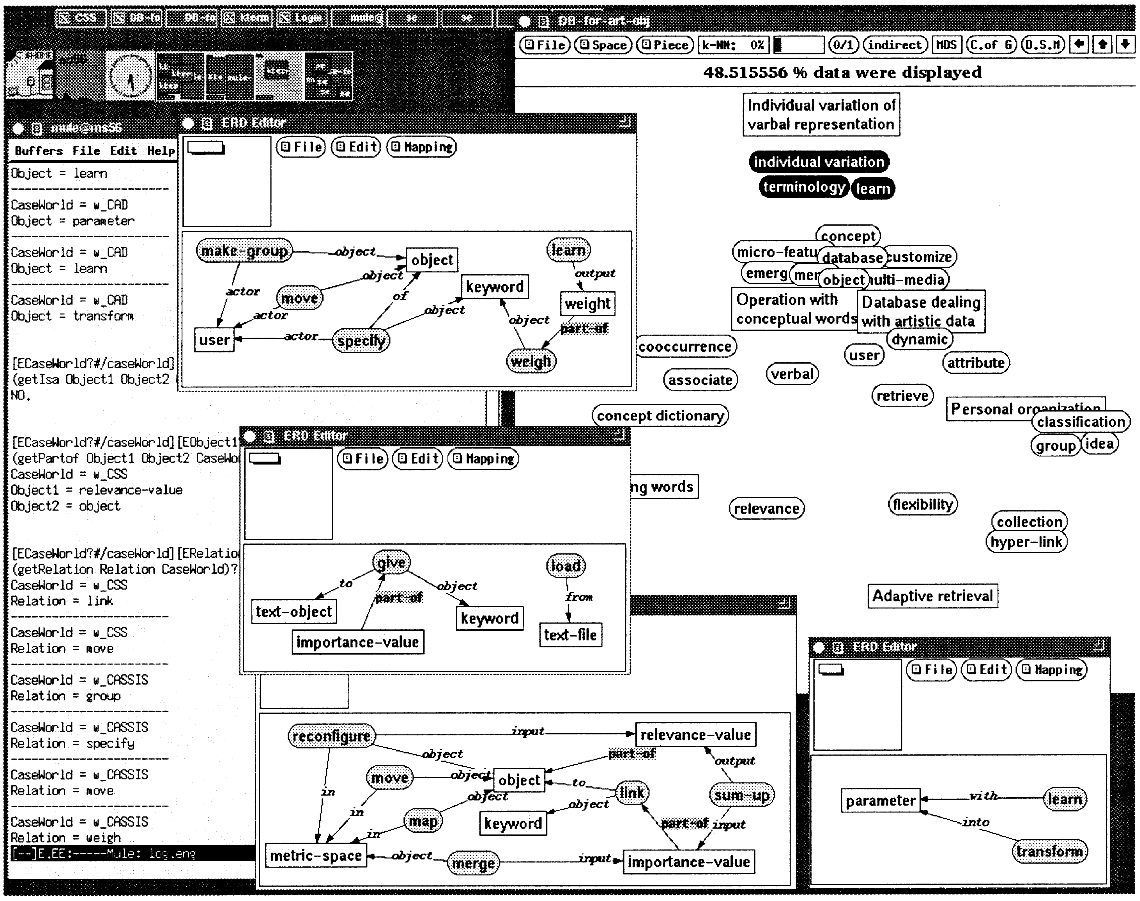
4.2. Extracting partial requirement models from past cases

This section describes the implementation of a task to extract a partial requirement model expressed in terms of previous cases, meeting the requirement concepts that emerge in a user's requirement space. Because of the comparatively abstract nature of expressions, keywords which indicate the user's requirements in the requirement space are rarely expressed in the same terms as objects (entities and relationships in the requirement models) in past models. For this reason, we have developed a method to indirectly extract components of a model from past models by exploiting past requirement spaces as mediators between abstract requirement spaces and concrete requirement models explicitly expressed in past cases.

The procedure for extracting a partial requirement model is as follows (refer to Fig. 3):

1. Suppose a user recognizes a requirement concept, such as 'visualizing the structure of$\cdots$is necessary' by noticing a co-occurrence of the two keywords 'visualization' and 'structure' in his/her requirement space, and he/she then wished to obtain a structured requirement model from past similar cases matching the requirement concept.
2. Past cases in which the keywords 'visualization' and/or 'structure' exist in the requirement space are selected, and the keywords mapped around these words are then activated. In Fig. 3, the keywords 'object', 'metric space', and 'mapping' are activated.
3. In the case objects (entities or relationships) use the same expressions as the keywords activated in step 2, the part containing the objects and the links connected to these are extracted.
4. The user judges whether the partial models extracted from several past cases have a sufficiently strong relationship with his/her requirement concept or not.

This procedure is implemented as an automatic program with KAUS, whose input is a set of keywords indicating a user's requirement concept and a threshold value for the propagation of activation, and whose output is a set of names of the past cases selected and the partial requirement models extracted.



```
Object = learn
------------------------
CaseWorld = w_CAD
Object = parameter
------------------------
CaseWorld = w_CAD
Object = learn
------------------------
CaseWorld = w_CAD
Object = transform


[ECaseWorld?#/caseWorld]
(getIsa Object1 Object2
NO.


[ECaseWorld?#/caseWorld][EObject1
(getPartof Object1 Object2 CaseWo
CaseWorld = w_CSS
Object1 = relevance-value
Object2 = object


[ECaseWorld?#/caseWorld][ERelatio
(getRelation Relation CaseWorld)?
CaseWorld = w_CSS
Relation = link
------------------------
CaseWorld = w_CSS
Relation = move
------------------------
CaseWorld = w_CASSIS
Relation = group
------------------------
CaseWorld = w_CASSIS
Relation = specify
------------------------
CaseWorld = w_CASSIS
Relation = move
------------------------
CaseWorld = w_CASSIS
Relation = weigh
[--]E.EE:-----Mule: log.eng
```

Fig. 4. Example of partial requirement models extracted from past cases.

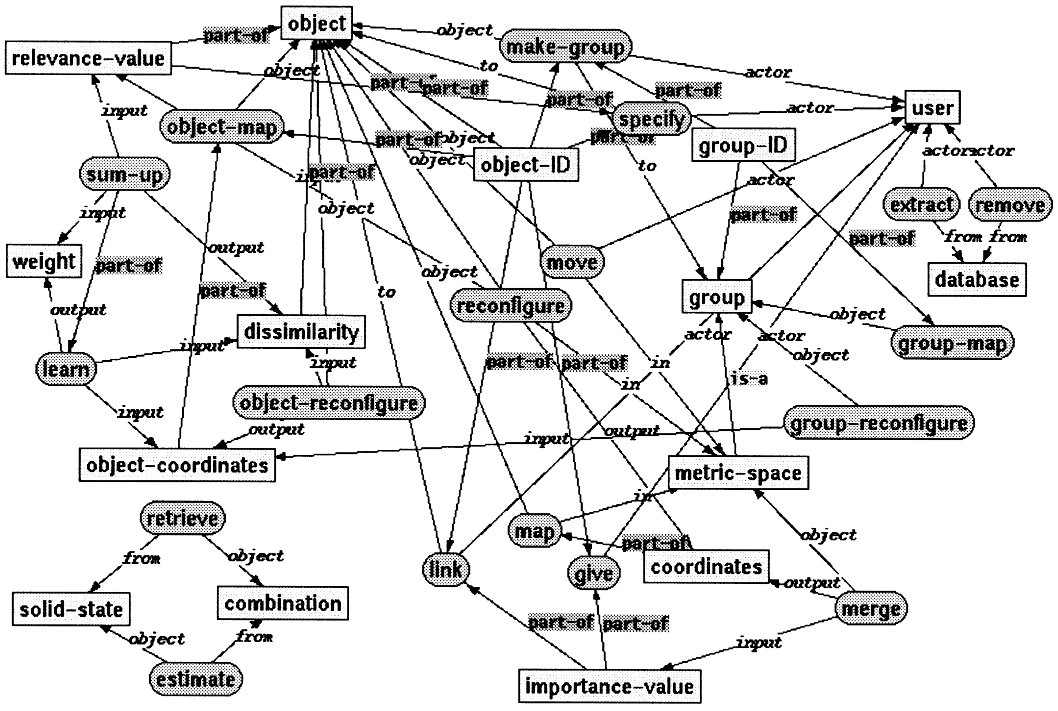


Fig. 5. Example of an automatically-generated requirement model.

Fig. 4 shows an example of the extraction of partial models form several past cases. The window on the right side is the requirement space being constructed by a user using CSS. In this case, the user pays attention to the keywords ‘individual variation’, ‘terminology’ and ‘learn’, and extracts requirement model parts corresponding to this keyword set from past cases. The window on the left side displays the states of inferences determined by KAUS. The results inferred by KAUS are output as partial models described with KAUS, and visualized with a graphical structure browser. In this example, four cases were found from which partial models were extracted. Users can utilize the results for the construction of their requirement model by selecting any suggestion from the system and modifying it. Even if the results themselves are not reusable, they may be useful for prototypes to further analyze their requirements. It can be said that the system enables users to obtain a concrete partial requirement model from their requirement concept even with abstract expressions.

4.3. Composing a tentative model from extracted parts

We also utilized an alternative method to compose new tentative models corresponding to a user’s entire requirement space from partial models extracted from multiple past cases.

The task examined in the previous section was merely to extract parts from past cases, while the task to be described here is to synthesize the parts extracted from multiple cases. The former guarantees the consistency of the context inside each result, while the latter does not do so in the model as a whole. Conversely, we can say that the latter task has the potential to cause a collision of multiple contexts, which may lead to *the emergence of a new structure*.

When solving problems in composition, the avoidance of conflict in a multiple structure is an important issue. Here, we need to resolve any inconsistency that results from the synthesis of components from different sources. For instance, in the case where a certain relationship object is accidentally extracted from multiple cases with different frame structures, one must be selected. Such a case can be observed in Fig. 4 where an object named ‘move’ is extracted from two different contexts. The implemented task can avoid a conflict of this sort by implementing a plausible heuristic criterion, for example, selecting the candidate that has the greatest number of slots of the same expression as any keyword in the user’s requirement space.

Fig. 5 shows an example of a requirement model automatically generated according to the requirement space shown in Fig. 2. This is the result of extracting several partial models from multiple past cases which are inferred to meet the requirement space, synthesizing these models using several heuristics, and visualizing them with a structure browser. In this way, users can instantly obtain a tentative requirement model by simply building their requirement space with CSS. This enables them to carry out *requirement acquisition*, *analysis*, and *modeling*, concurrently.

5. Experiments and evaluation

5.1. Experimental use for a new software design

One of the authors experimentally evaluated our system to externalize his ideas on a software design concerned with a new research topic and to make a prototype of its requirement model.

First, he vaguely supposed ‘a database system dealing with art’ as his subject. Then, he wrote memos on fragmentary ideas and requirements for that system, put them into six text-objects, and structured them as an externalized requirement space using CSS. Fig. 2 shows a snapshot of the space.

Next, he extracted and evaluated partial models, from previous requirement models, using the method described in Section 4.2 to obtain requirement models meeting requirement concepts emerging in the requirement space (see Fig. 4). At that time, viewing the requirement spaces of past cases made it easier for him to understand their contexts with which the extracted requirement models were formed, and this was useful for him to reconsider his requirements. Moreover, the simultaneously extracted partial models made him aware of the relationships among the different cases and supported his analogical thinking. In this experiment, more than 10 past cases were gathered and their requirement spaces were prepared using their requirement specification documents as cases stored in the case-base. In order to confirm the system behavior for modeling using previous models, four requirement models of cases selected from them were formed.

Moreover, the user tried to prototype an initial requirement model using knowledge on past cases; by the method described in Section 4.3. Fig. 5 is the result. The result enabled him to notice contradictions in the requirements and the inadequacy of term definitions; and hence encouraged him to restructure the requirement spaces using CSS. Additionally, by finding out that some parts of his requirement space did not have corresponding parts in the prototyped model, he realized that these parts were potentially original ideas, which were not seen in previous cases.

Note that users of our system can reuse partial requirement models extracted from past cases while being aware of their contexts; since our system provides users with not only computable models of past cases selected by simple keyword matching but also their corresponding requirement spaces, which have representations close to the users’ mental representation and which mediate between the users’ mental thoughts and the computational requirement models.

Moreover, collision among the contexts of past cases encourages new ideas and facilitates further evolution of the users' requirement spaces. The externalized words and concepts in the requirement spaces and models, by repeating such a cycle, are an invaluable repository for groups of people engaged in a collaborative project as a domain-dependent ontology.

5.2. Application to the evaluation of existing software specifications

In order to evaluate the role of requirement models in the development and management of software by groups, we performed the following experiment. In a research laboratory to which the authors had previously belonged, information services were provided using the world-wide web (WWW). Plans were being made to develop another program to manage the creation of home pages, to be made available to multiple users even outside their local network of computers. Features of the case were as follows: computer systems in the laboratory were administrated by multiple users; the software specifications were often modified because the software used a developing technology, namely, the WWW; and it was difficult to grasp the whole concept of the software and estimate its behavior because the software was open to computer networks.

First, we formed a requirement model from a requirement specification document written by a designer of the software. His requirement concepts were already clear since he was an expert programmer; hence, it was not hard to directly build the requirement model. By using the requirement model together with the original specification document, mutual understanding and the sharing of ideas and requirements of the software in the group were facilitated.

Inconsistencies in the software requirements not noticed before the building of the requirement model were observed, because the requirement model made it easier to grasp the data flow and the mutual dependencies among the functional modules. Modification of the requirement specifications was done by operating the requirement model with KAUS. This enabled the systematic modification of the requirements, for example, the system automatically removed a data structure no longer referred to by any functional module due to the removal of an unnecessary functional module.

It is certainly tiresome to form requirement spaces and models for individual projects. However, computerization of the upper stream of design, with methods like the one proposed in this paper, is inevitable for decreasing human errors during the creation of requirement specifications and the development and maintenance of a large-scale system by groups, which cannot be done by individuals. We believe the accumulation of machine-operatable records and know-how concerning past work brings benefits to a global cycle, including the development of similar systems and the modification of existing systems.

6. Concluding remarks and future work

In this paper, we described a system for aiding the construction of software requirement models by integrating techniques that support idea formation and knowledge-based modeling. This system enables users to effectively reutilize by-products produced during software development and research work such as informal documents, which would have otherwise been unusable.

Once a user's requirements have been sufficiently assembled in terms of a comptable model, they can be semi-automatically transformed into executable program code using existing automated software design technology (e.g. Ref. [13]). This means that the methodology proposed in this paper can shift the balance of the interface between humans and machines in software development toward the human side.

The approach described in this paper can be classified in terms of experience-based paradigms, such as analogical reasoning and case-based reasoning (CBR). These methods generally make the implicit, explicit, by allowing us to recognize something that we would not have encountered before by associating it with something we had. We also observed such an effect in the system proposed in this paper, i.e. users of the system can extract a concrete representation of requirement models from their abstract requirement spaces, and then recognize their own requirements more explicitly.

One of the important advantages of experience-based paradigms is that they store and provide a process model for the problem as well as for the solution. Although both CBR and expert systems rely on the explicit symbolic representation of experience-based knowledge to solve a new problem, expert systems only use generalized heuristics, i.e. a compilation of experiences. On the other hand, CBR's representation includes the justification of the solution or the requirements of the problem as well as its solution [14]. We believe that information representing the process for solving a problem is essential for effectively reusing experience-based knowledge, as well as its solution.

The method presented in this paper is a general paradigm for conceptual design, which is not limited to software design. For future work, we intend to: (1) apply and improve the paradigm for general usage by integrating it with other weak and general technologies; and (2) specify it for particular tasks, such as software design, by adding more sophisticated knowledge and methods. Related to point (1) above, Ref. [15] applies a method for visualizing the structure of conceptual spaces to daily conversations in communities for sharing knowledge and helping the mutual understanding among people engaged in collaborative work. For point (2), we need to compile and represent more sophisticated modeling knowledge and store experiences in a case-base according to the specific domain. Empirical experiments will also be indispensable for this.

Acknowledgements

The authors would like to thank Hiroyuki Yamauchi, who has been developing KAUS, for his valuable comments and encouragement. The authors would also like to thank Ryuta Ogawa for implementing CSS.

References

[1] C. Rich, R.C. Waters (Eds.), Readings in Artificial Intelligence and Software Engineering, Morgan Kaufmann, 1986.
[2] M.R. Lowry, R.D. McCartney (Eds.), Automating Software Design, AAAI Press/MIT Press, Cambridge, MA, 1991.
[3] A. Borgida, S. Greespan, J. Mylopoulos, Knowledge representation as the basis for requirements specifications, IEEE Comp. 18 (4) (1985) 82–90.
[4] D.A. White, The knowledge-based software assistant, a program summary, The 6th Annual Knowledge-Based Software Engineering Conference, IEEE Computer Society Press, 1991, pp. 2–6.
[5] K. Hori, S. Ohsuga, Computer-aided thinking for software development, Proceedings of the 2nd Pacific Rim International Conference On Artificial Intelligence, Seoul, 1992, pp. 203–208.
[6] G. Arrango, R. Prieto-Diaz, Introduction and overview, domain analysis concepts and research directions, in: R. Prieto-Diaz, G. Arrango (Eds.), Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, 1991, pp. 9–26.
[7] K. Hori, A system for aiding creative concept formation, IEEE Trans. Systems Man Cybernetics 24 (6) (1994) 882–894.
[8] Y. Sumi, K. Hori, S. Ohsuga, Computer-aided thinking by mapping text-objects into metric spaces, Artif. Intell. 91 (1) (1997) 71–84.
[9] M. Sugimoto, K. Hori, S. Ohsuga, Method to assist the building and expression of subjective concepts and its application to design problems, Knowledge-Based Syst. 7 (4) (1994) 233–238.
[10] Y. Sumi, R. Ogawa, K. Hori, S. Ohsuga, K. Mase, Computer-aided communications by visualizing thought space structure, Electronics Commun. Japan Part 3 79 (10) (1996) 11–22.
[11] S. Ohsuga, Framework of knowledge-based systems — multiple meta-level architecture for representing problems and problem-solving processes, Knowledge-Based Syst. 3 (4) (1990) 204–214.
[12] H. Yamauchi, S. Ohsuga, Modelling objects by extensions and intentions — a theoretical background of KAUS, in: S. Ohsuga et al. (Eds.), Information Modelling and Knowledge Bases III, IOS Press, 1992, pp. 160–173.
[13] C. Li, S. Ohsuga, A meta knowledge structure for program development support, Proceedings of the 5th International Conference On Software Engineering and Knowledge Engineering, 1993.
[14] M.L. Maher, A.G. de Silva Garza, Case-based reasoning in design, IEEE Expert 12 (2) (1997) 34–41.
[15] Y. Sumi, K. Nishimoto, K. Mase, Personalizing shared information in creative conversations, Proceedings of the IJCAI-97 Workshop on Social Interaction and Communityware, Nagoya, 1997, pp. 31–36.