정보기술응용연구
제 3 권 제 1 호
2 0 0 1 년 3 월

Guiding Requirement Formulation Using Scenarios : Grammar-based Convergent Approach

Gyeung-min Kim*

Abstract

Participatory Design (PD) community recognized that identification of use case scenarios describing possible uses of the future system is beneficial for users to identify their system requirements. However, brainstorming is a typical methodology for users to create use case scenarios during PD session, which heavily depend on the people skill and experiences of the analysts. The objective of this study is to develop a theoretical framework for automatic generation of requirement scenarios. Automatically generated scenarios serves as a menu of the possible user requirements from which user group can start to generate ideas about their requirements. The convergent approach taken here is novel in that the generated scenarios describe system requirements as well as the business process requirements in which the system operates. A context-sensitive grammar is used to generate the context relevant requirement scenarios.

Keywords: user requirement, requirement scenarios, idea generation, context-sensitive grammar.

*) School of Business Administration, Ewha Womans University, Seoul, Korea.

1. Introduction

Failure in identifying user requirements at the early stage of system development leads to resistance from the user group in use of the system. It has been reported that failure in Business Process Re-engineering (BPR) projects is resulted from lack of consideration for human factors issues involved in the introduction of a new technology [8]. Since BPR is initiated by top-management with assistance from BPR consultants, the requirement of the system is determined based not on user's job satisfaction needs, but on corporate BPR objectives.

Participative Design (PD) methodology has a long history in Scandinavia as an approach to developing user acceptable systems [4]. Regardless of whether the system is an information system or an electronic device, the purpose of PD is to improve usability of the system by accommodating users' perspectives on system uses. Even though PD receives attention from academic circles in Information Systems (IS) area, PD acceptance by North American IS practitioners is relatively low [4]. One of the difficulties in use of PD is attributed to lack of tools and techniques that ease the task of participating users and analysts [13, 18].

PD community recognized that identification of use case scenarios describing possible uses of future systems at the front end of the system development is beneficial for developing right system requirements [3, 14, 17]. However, brainstorming is a typical methodology for users to create use case scenarios during PD session, which heavily depend on the people skill and experiences of the analysts [3]. A formal methodology such as goal modeling in the requirement engineering research is known to be difficult to use in practice [29].

According to Nielsen [24, pp. 88-89], It is important to realize that users are not designers, so it is not reasonable to expect them to come up with design ideas from scratch it is important to realize that participatory design should not just consist of asking users what they want, since users do not know even what the possibilities are. Nelson indicates the need for tools and techniques to help users identify what they want.

The objective of this study is to develop a theoretical framework for generating requirement scenarios automatically in order to help users identify

their requirements. This paper takes the position of the previous research [3, 14, 17] that the requirement scenarios provide cues to foster idea generation during PD session. That is, users compare and reason about automatically generated requirement scenarios, which results in revising/eliminating the alternative user requirements or devising new ones. The revised and devised requirements are included in the final user requirements. During this process, various users' needs and issues are addressed and reflected in the requirement specification. As results, users will be more satisfied with the system; and the system will be more usable.

In order to fulfill the objective of this research, the following research questions are put forth:

(1) What are the underlying concepts and techniques of the idea generation tools?

(2) Given understanding of idea generation, what are the requirement elements that must be represented and connected to generate requirement scenarios?

(3) What is an appropriate representation/connection mechanism for automatic generation of the alternative scenarios?

The following section investigates the first question. Then, the last two questions will be addressed. Finally, conclusion of this study is presented.

2. Idea Generation

Referred to as problem formulation process in decision sciences, understanding problem domain consists of two complementary subprocess of information search and equivocality reduction [7, 30, 33]. Through information search, different viewpoints and issues of the problem are uncovered and broad understanding of the problem domain is generated [23]. Consensus on the problem is made through equivocality reduction. As results, what should be considered and excluded from the subsequent steps of problem solving is determined [23, 26].

Compared to heuristics and tools to support problem formulation in general, the tools designed to support information search stage of problem formulation are categorized as idea generation tool. The idea generation tool

simulates human's divergent thinking mode, making many connections among problem elements and generating alternative ideas [2, 9, 16, 21, 32]. In the example of a design of a transportation system [32], generic dimensions of the system are first identified: Power Source, Load Container and Control system. Then, possible alternatives for each dimension are identified below.

Power Source: Coal-Steam, Electric.

Load Container: Conveyor Belt, Flat Car, Enclosed vehicle.

Control system: Manual, Automatic.

A combination of each dimension comprises a total design concept:

(1) Coal-Steam/Conveyor Belt/Manual.

(2) Coal-Steam/Conveyor Belt/ Automatic.

(3) Electric /Conveyor Belt/Manual, etc

The presentation of the alternatives triggers human's thinking; and results in either identification of the new ideas or modification/elimination of the existing ideas in the alternatives.

According to MacCrimmon and Wagner [21], problems can be analyzed in many different perspectives such as attributes, functions, or purposes-means of the problem. Connections of these problem elements can also come in many different ways. In relational combinations [6, 21], the problem elements are combined by means of randomly selected relational words such as above. The purpose of this technique is to show familiar problem elements in a randomly constructed sentence to induce thought about new connections between problem elements.

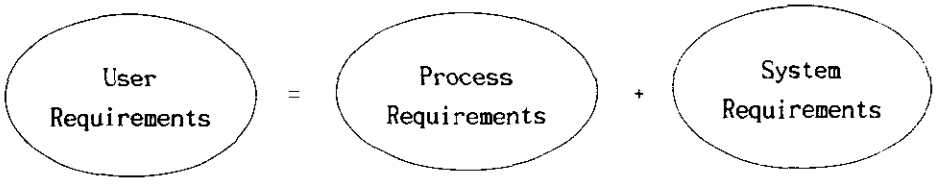
In summary, two issues in design of an idea generation tool are (1) identification of the problem elements and (2) identification of a mechanism to represent and connect the elements to generate alternatives.

Like problem formulation in decision making, requirement formulation comprises two complementary subprocesses of information search and equivocality reduction. First, information is searched to understand and analyze the problem domain. During this process, the general requirements are refined and clarified; and the potential requirements are uncovered. Then, consensus is made on what should be developed as well as what should not be developed. In this regard, some of concepts and techniques used in idea generation tools to support problem formulation can be applied to the requirement formulation.

The following section investigates the two issues involved in design of idea generation tool from the perspective of system requirement determination. First, requirement elements are identified. Then, how these elements can be represented and connected to generate the requirement scenarios is discussed.

3. Framework for Scenario Generation

When a new business system is developed, focus should be not only on system design but also on process design that the system supports [31]. This research takes a similar position that the business design should be implemented directly in software. Thus, the requirements formulation is viewed to consist of the identification of business process requirements and system requirements. This approach is called convergent approach.



[Figure-1] Convergent Approach

3.1 Convergent Approach

In object-oriented analysis, user requirements are defined in terms of objects and their responsibilities. The objects are people, computer systems or other business objects representing contemporary networked enterprise. Requirements formulation focuses on identifying objects and allocating responsibilities between people objects and system objects.

Exemplary user requirements for Order Management System are shown below:

Examples of User Requirements

1. Sales person receives an order from a customer.
2. Sales person reserves inventory in the database.
3. System issues an estimated delivery date for the order.
4. Shipping department delivers the order.

In these examples, main objects are: sales person, system, order, inventory and customer. Responsibilities of the objects are: "receives orders *from customer*, reserves *inventory* in the database, and issues an estimated delivery date for the orders. While the first two responsibilities are performed by the sales person, the third one is performed by the system.

In this study, Requirement Element (RE) that is a building block of user requirements is defined as a combination of an object and its responsibilities. A RE is denoted as *object.responsibility* representing that the *object* provides the *responsibility*. For example, the combination of an object, "sales person" and its responsibilities, " receives orders from customer " results in a RE, " sales person receives orders from customer " that is denoted as *Sales_Person.Receive_Orders_From_Customer*.

The REs are categorized according to the type of responsibility that each RE falls into. Consider the following exemplar REs. While the first two REs are categorized as the responsibility type, RECEIVE_ORDER, the next two are categorized as the responsibility type, RESERVE_INVENTORY.

Responsibility Type: RECEIVE_ORDER

RE1: *System.Receive_Orders_Through_Internet*.

RE2: *Sales_person.Receive_Orders_From_Floor*.

Responsibility Type: RESERVE_INVENTORY

RE3: *Sales_person.Reserve_Inventory_in_Database*.

RE4: *Sales_person.Reserve_Inventory_By_Calling_To_Warehouse*.

Responsibility Type: DELIEVR_ORDER

RE5: *Shipping_department.Deliver_order*

Each RE represents an alternative way of providing a certain type of responsibility by an object. Process is defined as a specific ordering of work activities across time and place, with a beginning and an end. A set of REs, each of which has a distinct responsibility type can be arranged in an order by which responsibilities must be carried out. Each series of RE is called Convergent Requirement Scenario (CRS). Examples are shown below:

CRS1: RE1, RE3, RE5

CRS2: RE1, RE4, RE5

CRS3: RE2, RE3, RE5

CRS4: RE2, RE4, RE5

The CRS1 consists of RE1 of RECEIVE_ORDER type, RE3 of RESERVE_INVENTORY type and RE5 of DELIVER_ORDER.

RECEIVE_ORDER is prerequisite of RESERVE_INVENTORY. This temporal constraint between business process steps is reflected in each CRS. Each CRS represents an alternative scenario for convergent requirement. In this example, each CRS depicts an alternative requirement for Order Management Systems. CRS1 describes the following case:

System.Receive_Orders_Through_Internet,
Sales_person.Reserve_Inventory_in_Database,
Shipping_department.Deliver_order.

3.2 The Source of Requirement Scenarios

To generate CRS automatically, an inventory of REs and their relationships need to be identified. They can be obtained from the cases of other companies in the similar situations to the user's domain. Many companies have similar kinds of business processes, each of which consists of the similar types of responsibilities. However, each company implements its responsibilities with a different set of means.

The first column in [Table 1] lists the types of responsibilities comprising the customer order process. The table indicates that every order process consists of a generic sequence of responsibilities such as RECEIVE_ORDER, CHECK_INVENTORY, and PROCESS_ORDER [20]. For each responsibility type, each of the second and third columns shows a specific means to implement the responsibility type. Case 1 illustrates the process where the responsibility type, RECEIVE_ORDER is accomplished by *receive_order_from_Internet.* Case 2 describes the process where RECEIVE_ORDER, is implemented by *receive_order_from_floor.*

Given a pool of the REs, a number of valid CRSs can be composed. The framework used to generate the valid CRS is a grammar. A grammar provides a framework for generating new instances of a set. In linguistic grammar, a number of valid sentences can be generated from a pool of words. Each sentence is an instance of a certain grammatical rule. For example, from a set of verbs {eats, washes} and a set of nouns {dog, apple}, the following sentences can be generated: "dog eats apple", "dog washes apple", "apple eats dog" and "apple washes dogs". Each sentence is a distinct

case of the rule, noun verb noun. As linguistic grammar is used to represent the sequential relationships of words, Process Grammar [27] is used in this study to represent the temporal relationships among REs.

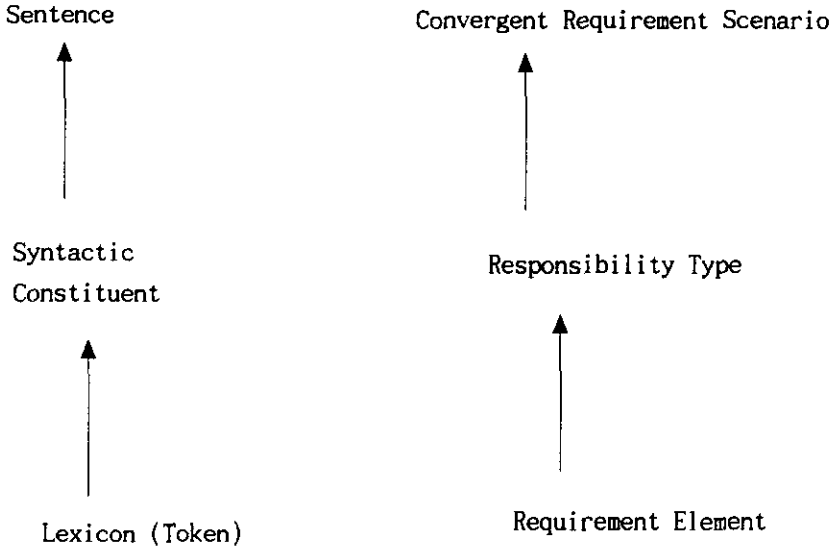
[Table-1] Requirement Elements (REs) in Customer Order process
S: Sales department, W: Warehouse, SH: Shipping department

Responsibility Types	Case 1	Case 2	...
RECEIVE ORDER	S.Receive_order_from_Internet	S.Receive_order_from_floor	...
CHECK_ INVENTORY	S.Look_up_computer	S.Look_up_computer	...
PROCESS_ORDER	S.Send_copy_to_warehouse S.Notify_to_shipping SH.Schedule_delivery SH.Call_to_customer SH.Deliver_product	S.Put_order_into_computer S.Schedule_delivery W.Check_order SH.Deliver_product	...

3.3 Process Grammar

Although the most familiar type of grammar is English grammar, grammar has been used in many areas to describe a set of possible patterns such as circuit patterns in semi-conductor wafer. While linguistic grammar defines a set of valid sentences in a language, circuit grammar defines a set of valid shapes of electronic circuits.

The basic elements of a grammar is called a lexicon [22] or token. They are treated analytically as the most detailed level of description necessary for the problem at hand. An example of the lexicon is a word of a language. The next level of the grammatical component is called syntactic constituents. In linguistics example, words or phrases are categorized into different syntactic constituents such as noun, verb, noun phrases or verb phrases according to a particular function that each category serves in the syntax of sentence. These constituents can be combined according to grammatical rules to create sentences. [Figure 2] shows hierarchical relationships among grammatical components in linguistics.



[Figure-2] Relationships among Grammatical Components

In this study, a Requirement Element (RE) is considered as a token. As lexicons are categorized into syntactic constituents based on their functionality, REs are classified into Responsibility Type according to their responsibilities. Thus, Responsibility Type corresponds to syntactic constituents in the linguistics. Syntactic constituents have the following characteristics [27]:

1) provide a way of describing the structural features of a pattern without elaborating it all the way down to the specifics of the token.
2) can be nested together.

The responsibility type such as RECEIVE_ORDER meets the first characteristics since it generalizes *receive_order_by_mail*, *receive_order_from_floor* and other ways of receiving orders. The responsibility type, PROCESS_ORDER is a nested with another responsibility type, FILL_ORDER_FROM_STORE, FILL_ORDER_FROM_OTHER_STORE, or FILL_ORDER_FROM_ON_ORDER.

The 'responsibility types' are combined according to the various constraints such as temporal and job-control constraints [28]. For example, CHECK_INVENTORY must be performed later than RECEIVE_ORDER. The constraints govern the way REs are arranged in order to create Convergent Requirement Scenarios (CRSs). [Figure 2] describes relationships among components of the process grammar. The following rules, numbered as 1 through 20, describe a grammar for Customer Order Process (COP):

1. *CUSTOMER_ORDER_PROCESS* -> RECEIVE_ORDER, CHECK_INVENTORY, PROCESS_ORDER.
2. RECEIVE_ORDER -> METHOD, RECEIVE_ORDER.
3. METHOD -> **human.**
4. METHOD -> **automatic.**
5. **human,** RECEIVE_ORDER -> *S.receive_order_by_mail, human .*
6. **human,** RECEIVE_ORDER -> *S.receive_order_from_floor, human.*
7. **automatic,** RECEIVE_ORDER -> *System.receive_order_from_internet, automatic.*
8. **human,** CHECK_INVENTORY -> *S.look_up_computer, human.*
9. **automatic,** CHECK_INVENTORY -> *Sytem.check_inventory_database, automatic.*
10. **human,** PROCESS_ORDER -> *S.return_order.*
11. **human,** PROCESS_ORDER -> **human,** FILL_ORDER_FROM_STORE.
12. **human,** PROCESS_ORDER -> **human,** FILL_ORDER_FROM_OTHER_STORE.
13. **human,** PROCESS_ORDER -> **human,** FILL_ORDER_FROM_ON_ORDER.
14. **automatic,** PROCESS_ORDER -> *System.send_out_of_stock_message.*
15. **automatic,** PROCESS_ORDER -> **automatic,** FILL_ORDER_FROM_STORE.
16. **human,** FILL_ORDER_FROM_STORE -> *S.send_copy_to_warehouse, S.notify_to_shipping, SH.schedule_delivery, SH.call_to_customer, SH.deliver_product.*
17. **human,** FILL_ORDER_FROM_STORE -> *S.put_order_into_computer, S.schedule_delivery, W.check_order, SH.deliver_product.*

18. **human,** FILL_ORDER_FROM_OTHER_STORE -> *S.dial_to_store, S.reserve_stock_for_pick_up.*
19. **human,** FILL_ORDER_FROM_ON_ORDER -> *S.reserve_from_on_order.*
20. **automatic,** FILL_ORDER_FROM_STORE -> *System.schedule_delivery, System.send_notice_to warehouse, SH.deliver_product.*

Each rule in the grammar has a number for reference. The symbol -> is read as consists of. While an italic element (e.g. *S.receive_order_from_floor*) is a RE, an element in upper case fonts (e.g. RECEIVE_ORDER) is a responsibility type. The element in italic upper case (e.g. *CUSTOMER_ORDER_PROCESS*) represents the domain of the requirement scenarios. Rule 1 states that the Convergent Requirement Scenarios (CRSs) for *CUSTOMER_ORDER_PROCESS* consist of a series of responsibility types, RECEIVE_ORDER, CHECK_INVENTORY and PROCESS_ORDER. Each nonterminal in the right-hand side of the rule is further elaborated by applying the corresponding rules. For example, RECEIVE_ORDER in the rule 1 can be elaborated with right-hand side of the rule 2. The nonterminal METHOD is further elaborated with the rule 3 or rule 4. [Figure 3] demonstrates the process of elaborating nonterminals. The nonterminal METHOD in the rule 2 provides a context to determine when different rules can be applied. Depending on whether the responsibility in the previous step is delivered automatically or by human, the mode of the subsequent responsibility is determined. This ensures semantic agreement among process steps. The COP grammar is a context-sensitive grammar (Type 1) [5, 19]. Context-sensitive grammar allows more than one symbol to be on the left-hand side of a rule and makes it possible to define a context in which the rule can be applied. Compared to the Pentland's context-free grammar [28], COP context-sensitive grammar can perform checks for such constraints.

[Figure 3] shows the generation of the CRS1:

CRS1: *S.receive_order_from_floor, S.look_up_computer, S.return_order* (Rules applied: 1, 2, 3, 5, 8 and 10)

Depending upon which rule is applied at the step of 6, the following CRSs are generated:

CRS2: *S.receive_order_from_floor, S.look_up_computer, S.send_copy_to_warehouse, S.notify_to_shipping, SH.schedule_delivery, SH.call_to_customer, SH.deliver_product* (Rules applied: 1, 2, 3, 5, 8, 11 and16)

.

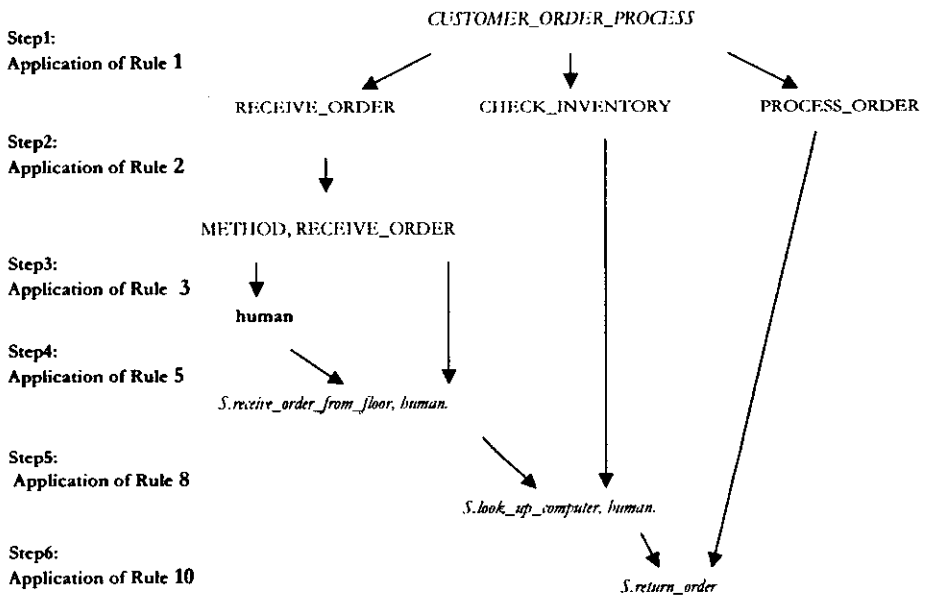
CRS3: *S.receive_order_from_floor, S.look_up_computer, S.put_order_into_computer, S.schedule_delivery, W.check_order, SH.deliver_product* (Rules applied: 1, 2, 3, 5, 8, 11 and 17)

.

CRS4: *S.receive_order_from_floor, S.look_up_computer, S.dial_to_store, S.reserve_stock_for_pick_up* (Rules applied: 1, 2, 3, 5, 8, 12 and 18)

.

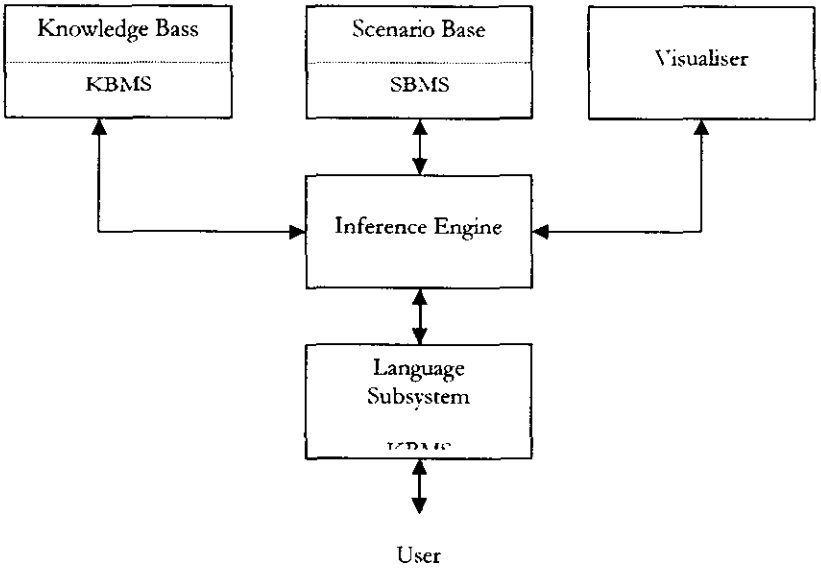
CRS5: *S.receive_order_from_floor, S.look_up_computer, S.reserve_from_on_order* (Rules applied: 1, 2, 3, 5, 8, 13 and 19)



[Figure-3] Convergent Requirement Scenario Generation

4. Development of the Prototype System

The development of the prototype system is under way. An architecture of the prototype system is shown in [Figure 4]. The prototype system consists of knowledge base, scenarios base, their management systems, inference engine and language subsystem. The language subsystem provides user interface to accept the user requests and display the scenarios of a domain in which the Participatory Design (PD) team is interested. The knowledge base is a repository for the process grammar. Inference engine generates the CRS based on the user request and the grammar stored in the knowledge base. The CRS generated are stored in the scenario base. The visualizer visualizes the generated scenarios. The scenarios can be displayed on a big screen that can be seen by PD team members.



KBMS: Knowledge Base Management Systems
SBMS: Scenario Base Management Systems

[Figure-4] An Architecture of the Prototype System

Based on the visualized requirement scenarios, the PD team can compare and reason about their requirements; identify missing or unnecessary user requirements; and correct misunderstood requirements. For example, presentation of CRS3 in the previous section could trigger the shipping department to realize that they want the authority of scheduling the product shipping instead of the delivery schedule being given by the sales department (note that scheduling is done by the sales department). Thus, CRS3 could enable PD group to identify needs of the job autonomy in shipping department. In this way, user oriented issues involved in system requirements can be minimized.

5. Conclusion

When novice users are not certain about what to do, they need help. User participation in system development requires techniques and tools that enable end users to understand the possibilities for computer support [15].

This paper takes the position of the previous research that the requirement scenarios provide cues to foster user creativity during PD session. Although, empirical studies on effectiveness of the CRS in requirement formulation need to be done in the future, the contribution of this study is to develop a theoretical framework of generating requirement scenarios automatically. This study is novel in the following ways: (1) the convergent approach is used to generate the system requirement scenarios as well as the business process scenarios in which the system operates; (2) a context-sensitive grammar is used to generate the context relevant requirement scenarios.

The use of the CRS can complement the weakness of the existing PD techniques that is qualitative in nature and relies mainly on analyst's experience and low technology methods. Generating requirement scenarios is a complex sequence of activities. An automated tool to generate the scenarios allows PD to become the norm rather that the exception.

6. Reference

[1] Abell, P., The Syntax of Social Life: The Theory and Method of Comparative Narratives, New York, Clarendon Press, 1987.

[2] Ackoff, R. and Vergara, E., "Creativity in Problem Solving and Planning: A Review", European Journal of Operational Research, Vol. 7, No.1, 1981, pp.1-13.

[3] Brown, D, An Introduction to Object-Oriented Analysis, Objects in Plain English, John Wiley and Sons Inc., 1997.

[4] Carmel, E, Whitaker, R, and George, J, "PD and Joint Application Design: A Transatlantic Comparison", Communications of the ACM, Vol. 36, No.4, June 1993, pp. 40-48.

[5] Chomsky, N, Aspects of the Theory of Syntax, MIT Press, Cambridge, MA, 1965.

[6] Crovits, H, Galton's Walk Haper & Row, New York, 1970.

[7] Daft, R and Lengel, R "Organizational Information Requirements, Media Richness and Structural Design", Management Science, Vol. 32, No. 5, May 1986, pp. 554-571.

[8] Davenport, T, "Will Participative Makeovers of Business Process Succeed Where reengineering Failed?", Planning Review, Vol. 23, No.1, 1995, pp. 24-29.

[9] De Bono, E, De Bono's Thinking Course, Facts on file, New York, NY, 1993.

[10] Ehan, P., Mlleryd, B. and Sjgren, D., "Playing in reality: A paradigm case", Scandinavian Journal of Information Systems, No. 2, 1990, pp. 101-120.

[11] Ehninger, D and Brockreide, W., Decision by Debate, Harper & Row, New York, 1978.

[12] Goffman, E., Forms of Talk, University of Pennsylvania Press, Philadelphia, PA, 1981.

[13] Greenbaum, J., "Panel Presentation-Author's notes", Conference on Participatory Design-PDC'90, Seattle, Wa. 1990.

[14] Greenbaum, J. and Kyng, M., "Design at Work: Cooperative Design of Computer Systems", Lawrence Erlbaum Associates, Hillsdale, NJ,

1991.

[15] Grnbaek, K., Kyng, M. and Mogensen, P., "CSCW Challenges: Cooperative Design in Engineering Project", Communications of the ACM, Vol. 36, No.4, 1993.

[16] Guilford, J., The Nature of Human Intelligence, McGraw-Hill, New York, 1967.

[17] Jacobson, I., Christerson, M., Jonsson, P. and vergaard, G., Object-Oriented Software Engineering, Addison-Wesley, New York, 1992.

[18] Kensing, F. and Munk-Masden, A., "PD: Structure in the Toolbox", Communications of the ACM, Vol. 36, No.4, 1993, pp.78-85.

[19] Luger, G. and Stubblefield, W., Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Addison-Wesley, New York, 1997.

[20] Malone, T., Crowston, K., Lee, J. and Pentland, B., "Tools for Inventing Organizations: Towards a Handbook of Organizational Processes, Proceedings of the Second IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises", Morgantown, WV, April 1993.

[21] MacCrimmon, K. and Wagner, C., "Stimulating Idea through Creativity Software", Management Science, Vol. 40, No. 11, 1994, pp. 1514-1532.

[22] Miclet, L., Structural Methods in Pattern Recognition, Springer-Verlag, New York, 1986.

[23] Niederman, F. and Desanctis, G. "The Impact of a Structured-Argument Approach on Group Problem Formulation", Decision Sciences, Vol. 26, No. 4, 1995, pp. 451-474.

[24] Nielsen, J., Usability Engineering, AP Professional, Chestnut Hill, MA, 1993.

[25] Norman, R., Object-Oriented Systems Analysis and Design, Prentice Hall, Upper Saddle River, NJ, 1996.

[26] Nutt, P., "Formulation Tactics and the Success of Organizational Decision Making", Decision Sciences, Vol. 23, No. 3, 1992, pp. 519-540.

[27] Pentland, B., "Organizing Moves Software Support Hot Lines",

Administrative Science Quarterly, Vol. 37, No. 4, 1992, pp.527-548.

[28] Pentland, B., "Grammatical Models of Organizational Process", Organizational Science, Vol. 6, No. 5, 1995, pp. 541-556.

[29] Rolland, C., Souveyet, C., and Achour, C., "Guiding Goal Modeling Using Scenarios", IEEE Transactions on Software Engineering, Vol. 24, No. 12, 1998, pp. 1055-1070.

[30] Simon, H., The New Science of Management Decisions, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[31] Taylor, D., Business Engineering with Object Technology, John Wiley & Sons, Inc., 1995.

[32] Young, L., "Knowledge-Based Systems for Idea Processing Support", Data Base Vol. 22, No. 1/2, 1991, pp. 46-50.

[33] Weick, K., The Social Psychology of Organizing, Addison-Wesley, Mass., 1979.

◆ 저자소개 ◆

김 경민 (Kim, Gyeung-min)

이화여자대학교 컴퓨터학과를 졸업하고, Texas Tech University에서 경영정보학 석사와 박사학위를 취득하고, 현재 이화여자대학교 경영학부에서 조교수로 재직 중이다. 주요 연구분야로는 비즈니스 프로세스 혁신, 사용자 참여 시스템 디자인 (participatory design), 프로세스 시뮬레이션 등이며, *Journal of Organizational Computing and Electronic Commerce, Journal of Systems and Software, Journal of End-user Computing, Cycle Time Research* 등 다양한 국내외 학술지 및 학회에 논문을 발표하였다.

E-mail: gkim@mm.ewha.ac.kr.

TEL: (02) 3277-3581