We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

2

Knowledge Representation and Validation in a Decision Support System: Introducing a Variability Modelling Technique

Abdelrahman Osman Elfaki[1], Saravanan Muthaiyah[2]
Chin Kuan Ho[2] and Somnuk Phon-Amnuaisuk[3]
[1]*Management and Science University*
[2]*Multimedia University Cyberjaya,*
[3]*Tunku Abdul Rahman University*
Malaysia

1. Introduction

Knowledge has become the main value driver for modern organizations and has been described as a critical competitive asset for organizations. An important feature in the development and application of knowledge-based systems is the knowledge representation techniques used. A successful knowledge representation technique provides a means for expressing knowledge as well as facilitating the inference processes in both human and machines [19]. The limitation of symbolic knowledge representation has led to the study of more effective models for knowledge representation [17].

Malhotra [14] defines the challenges of the information-sharing culture of the future knowledge management systems as the integration of decision-making and actions across inter-enterprise boundaries. This means a decision making process will undergo different constraints. Therefore, existence of a method to validate a Decision Support System (DSS) system is highly recommended. In the third generation of knowledge management, the knowledge representation acts as boundary objects around which knowledge processes can be organized [26]. Knowledge is viewed in a constructionist and pragmatic perspective and a good knowledge is something that allows flexible and effective thinking and construction of knowledge-based artifacts [26].

This paper answers the two questions of [26] and [14] in the context of a DSS: 1) how to define and represent knowledge objects and 2) how to validate a DSS.

For any decision, there are many choices that the decision maker can select from [7]. The process of selection takes place at a decision point and the selected decision is a choice. For example, if someone wants to pay for something, and the payment mode is either by cash or by credit card, the payment mode is the decision point; cash and credit card are choices. Now, we can conclude that the choices and decision points represent the knowledge objects in DSS. Choices, decision points and the constraint dependency rules between these two are collectively named as variability. Task variability is defined in [5] as the number of exceptions encountered in the characteristics of the work. The study in [5] tested the importance of variability in the system satisfaction. Although there are many existing approaches for representing knowledge DSS, the design and implementation of a good and useful method that considers variability in DSS is much desired.

The term variability generally refers to the ability to change; to be more precise, this kind of variability does not occur by chance but is brought about on purpose. In other words, variability is a way to represent choices. Pohl et al. [20] suggest the three following questions to define variability.

What does it vary? : Identifying precisely the variable item or property of the real world. The question leads us to the definition of the term variability subject (A variability subject is a variable item of the real world or a variable property of such an item).

Why does it vary? : There are different reasons for an item or property to vary: different stakeholders' needs, different countries laws, technical reasons, etc. Moreover, in the case of interdependent items, the reason for an item to vary can be the variation of another item.

How does it vary? : This question deals with the different shapes a variability subject can take. To identify the different shapes of a variability subject, we define the term variability object (a particular instance of a variability subject).

Example of variability *Subject and Objects for "Car":*

The variability subject "car" identifies a property of real-world items. Examples of variability objects for this variability subject are Toyota, Nissan, and Proton.

The problem of representing variability in a DSS requires a complex representation scheme to capture static and dynamic phenomena of the choices that can be encountered during the decision process. We believe that the key feature of such knowledge representation (for variability in a DSS) is its capability of precise representation of diverse types of choices and associations within them. This involves: i) qualitative or quantitative description of choices and their classification, ii) representation of causal relationships between choices and iii) the possibility of computerizing the representation.

The main aim of variability representing in DSS is to create a decision repository that contains decision points, its related choices and the constraint dependency relations between decision points-choices, choices-choices, or decision points-decision points.

Nowadays, Feature Model (FM) [12] and Orthogonal Variability Model (OVM) [20] are the well-known techniques to represent variability. Although, FM and OVM are successful techniques for modeling variability, some challenges still need to be considered such as logical inconsistency, dead features, propagation and delete-cascade, and explanation and corrective recommendation. Inconsistency detection is defined as a challenging operation to validate variability in [2]. In [27] the source of logical inconsistency is defined from a skill-based or rule-based errors which would include errors made in touch-typing, in copying values from one list to another, or other activities that frequently do not require a high level of cognitive effort. One of the main drawbacks coming from the fusion of several different and partial views is logical inconsistency [9]. Dead feature is defined in [6] as a frequent case of error in feature model- based variability. Instead of dead feature, we called it dead choice. Modeling variability methods must consider constraint dependency rules to assure the correctness of the decision. Propagation and delete cascade operation is proposed to support auto selection of choices in the decision making process. Propagation and delete cascade operation is a very critical operation in the semi-auto environment.

This paper defines a rule-based approach for representing and validating knowledge in DSS. In addition to representing variability to capture knowledge in DSS, intelligent rules are defined to validate the proposed knowledge representation. The proposed method validates two parts. The first part is validating a decision repository in which a logical inconsistency and dead choices are detected. The second part is validating the decision

making process by providing automated constraint dependency checking, explanation and corrective recommendation, and propagation and delete-cascade.
This paper is structured as follows: Literature is surveyed in section two. Knowledge representation of DSS using variability is demonstrated in section three. Knowledge validation is illustrated in section four and implementation is discussed in section five. Section six contains the conclusion and future work.

2. Related work

The aim of knowledge representation is to facilitate effective knowledge management which concerns expressive representation and efficiency of reasoning in human [15]. Related works on this area are summarized as follows:
Haas [10] investigated the feasibility of developing an overarching knowledge representation for Bureau of Labor Statistics information that captured its semantics, including concepts, terminology, actions, sources, and other metadata, in a uniformly applicable way. Haas suggested the (ISO/IEC 11179) standard for metadata, as knowledge representation techniques. Molina [16] reported the advantages of using knowledge modeling software tool to help developers build a DSS. Molina describes the development of DSS system called SAIDA where knowledge is represented as components, which was designed by Knowledge Structure Manager (KSM). KSM is a knowledge-modeling tool that includes and extends the paradigm of task method-domain followed by different knowledge engineering methodologies. KSM provides a library of reusable software components, called primitives of representation that offer the required freedom to the developer to select the most convenient representation for each case (rules, frames, constraints, belief networks, etc.).
Froelich and Wakulicz-Deja [8] investigated problems of representing knowledge for a DSS in the field of medical diagnosis systems. They suggested in [8] a new model of associational cognitive maps (ACM). The ability to represent and reason with the structures of causally dependant concept is the theoretical contribution of the proposed ACM. Antal [1] proposed the bayesian network as a knowledge representation technique to represent multiple-point-of views. The proposed technique in [1] serves as a refection of multiple points of view and surpasses bayesian network both by describing dependency constraint rules and an auto-explanation mechanism. Lu et al. [13] developed a knowledge-based multi-objective DSS. The proposal in [13] considers both declarative and procedural knowledge. Declarative knowledge is a description of facts with information about real-world objects and their properties. Procedural knowledge encompasses problem-solving strategies, arithmetic and inferential knowledge. Lu et al. [13] used text, tables and diagrams to represent knowledge.
Brewster and O'Hara [3] prove difficulties of representative skills, distributed knowledge, or diagrammatic knowledge using ontologies. Pomerol et.al [21] used artificial intelligence decision tree to represent operational knowledge in DSS. Christiansson [4] proposed semantic web and temporal databases as knowledge representation techniques for new generation of knowledge management systems. One of the most sophisticated knowledge modeling methodologies is Common KADS [24]. Common KADS explains how to model a knowledge application through structural top-down analysis of the problem domain. The outcome of modeling process according to Common KADS consists of three layers that are called contextual model, conceptual model and design model. Common KADS model did not provide mechanism to define relation between objects or between layers. Padma and

Balasubramanie [18] used traditional knowledge-based tool to define DSS. Williams [28] described the benefits of using Ontologies and Argumentation for DSS. Suh in [25] applied Database Management System (DBMS) in two-phased decision support system for resource allocation.

To the best of our knowledge, there is no one particular or specific method for handling variability as a knowledge representation technique in DSS. In addition to variability representation, our proposed method could be used to deal with main challenges in variability representation such as: constraint dependency rules, explanation, propagation and delete-cascade, logic inconsistency detection and dead decision detection. Table 1 summarized the previous works in knowledge representation and validation regarding a DSS. The columns are denoted as following: KR for Knowledge Representation, CDR for Constraint Dependency Rules, Expl for Explanation, Pro and DC for Propagation and delete-cascade, LID for Logic Inconsistency Detection and DDD for Dead Decision Detection.

Technique	Ref.	KR	Reasoning	CDR	Expl	Pro and DC	LID	DDD	Gap
ISO/IEC 11179	10	Yes	No	No	No	No	No	No	6/7
Traditional artificial intelligence knowledge representation techniques such as frames, decision trees, belief networks, etc	16,1,12	Yes	Yes	No	No	No	No	No	5/7
Associational cognitive maps(ACM)	8	Yes	Yes	No	No	No	No	No	5/7
Bayesian network	1	Yes	Yes	Yes	Yes	No	No	No	3/7
Text, tables and diagrams	13	Yes	No	No	No	No	No	No	6/7
Ontologies	3, 28	Yes	Yes	No	No	No	No	No	5/7
Temporal database and Semantic Web	4	Yes	Yes	Yes	Yes	No	No	No	3/7
Three layer modeling(KADS)	24	Yes	Yes	No	No	No	No	No	5/7
DBMS	25	Yes	Yes	No	No	No	No	No	5/7

Table 1. Summary of Literature Review

Table 1 clarifies the need for new method of representation and validating knowledge in DSS. Although the logic inconsistency and dead decision problems are cleared in variability representation methods, some works in literature expressed these problems in a DSS [27].

3. Knowledge representation in DSS using variability

In this section, we defined variability is DSS, and then described how to represent variability in a DSS using First Order Logic (FOL).

3.1 Define variability in DSS

By variability in DSS, we mean choices representation (considering dependency constraint rules between them). In the choice phase, the decision maker chooses a solution to the problem or opportunity. DSS help by reminding the decision maker what methods of choice are appropriate for the problem and help by organizing and presenting the information [7]. Hale [11] states that "relationships between knowledge objects such as kind-of, and part-of become more important than the term itself". We can look for choices as knowledge objects. The proposed method defines and deals with these types of relationship and with dependency constraints between choices such as require and exclude.
There is no standard variability representation for a DSS [21]. The proposed method can enhance both readability and clarity in representation of variability in DSS. Consequently, the proposed method represents variability in high degree of visualization (using graph representation) considering the constraints between choices. As we mentioned before in the definition of variability, there are two items: variability subject and variability object. We suggest variability subject as a *decision point* and variability object and a *choice*. As example from figure 1: the variability subject "Promotion" identifies a *decision point*. Examples of variability objects for this variability subject are "Experience, Qualifications, or Special Report". This example illustrated three choices: "Experience, Qualifications, or Special Report" that the decision maker can select from in the decision point "Promotion".

A reward system as an example:

Rewards systems can range from simple systems to sophisticated ones in which there are many alternatives. It is closely related to performance management. Both rewarding and performance measuring are difficult tasks due to the decision variability that takes place in different activities of human resources cycle as it can be seen in figure 1.

3.2 Representing variability in DSS using first order logic

In this sub-section, the notations of the proposed method are explained. Syntaxes and semantics (the most important factors for knowledge representation methods) for the proposed method are defined. The proposed methods composed of two layers. The upper layer is a graphical diagram. Providing visual picture is the function of the upper layer. The lower layer is a representation of the upper layer in forms of predicates. Providing a reasoning tool is the aim of the lower layer. You can imagine the upper layer as a user-interface while the lower layer as a source code. In the lower layer, decision point, choices, and constraint dependency rules are represented using predicates. The output of this process is a complete modeling of variability in DSS as knowledge- based. In other words, this process creates a decision-repository based on two layers. This decision-repository contains all decisions (choices) grouped by decision points. The proposed method validates both decision-repository and decision making process.

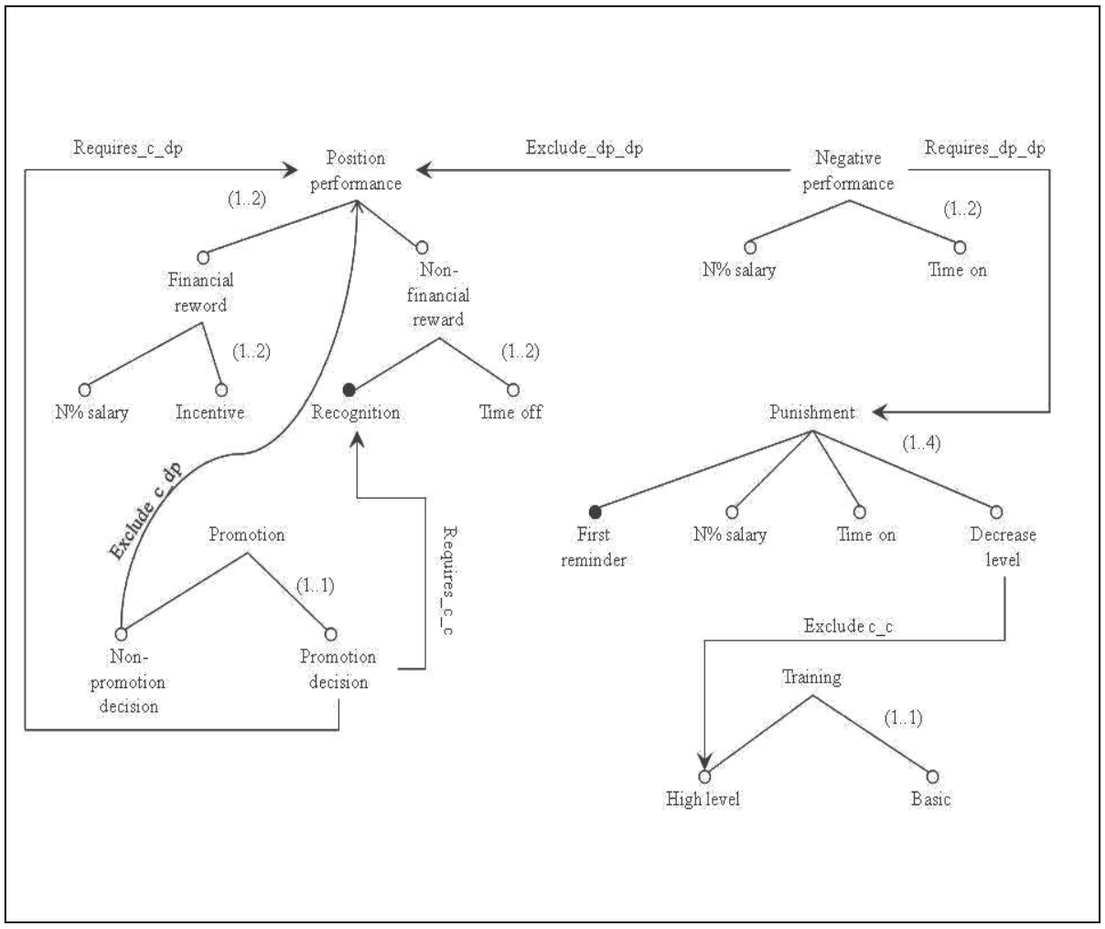


Fig. 1. Variability in Reward System: Upper layer representation

3.1.1 Upper layer representation of the proposed method (FM-OVM)

In this layer, we combined FM diagram with OVM notations. Figure 1 represents the upper layer of our proposed method. Optional and mandatory constraints are defined in Figure 1 by original FM notations [9], and constraint dependency rules are described using OVM notations. OVM and FM can easily become very complex for validating a medium size system, i.e., several thousands of decision points and choices are needed. This reason motivates us to develop an intelligent method for representing and validating variability in DSS.

3.1.2 Lower layer of the proposed method

Decision points, choices, and constraint dependency rules are used to describe variability [20]. Constraint dependency rules are: decision point requires or excludes decision point, choice requires or excludes choice, and choice requires or excludes decision point. In this sub-section, decision points, choices, and dependency constraint rules are described using predicates as a low level of the proposed method. Examples are based on Figure 1. Terms beginning with capital letters represent variables and terms beginning with lower letter represent constants. Table 2 shows the representation of *Negative Performance* decision point using the lower layer of the proposed method.

DECISION POINT

A decision point is a point that the decision maker selects one or more choices. In figure 1, *training* represents a decision point and *Basic* and *High level* represent its choices. The following five predicates are used to describe each decision point:

type:

Syntax: type(Name1,decisionpoint).
Semantic: Identify the type, *Name1* defines the name of decision point, e.g. type (promotion, decisionpoint).

choiceof:

Syntax: choiceof(Name1, Name2).
Semantic: Identifies the choices of a specific decision point. *Name1* represents name of decision point and *Name2* represents the name of a choice, e.g. choiceof (promotion, promotion decision)

max:

Syntax: max(Name, int).
Semantic: Identifies the maximum number of allowed selection at a decision point. *Name* defines the name of the decision point and *int* is an integer,e.g. max(positive performance, 2).

min:

Syntax: min(Name, int).
Semantic: Identifies the minimum number of allowed selection at a decision point. *Name* defines the name of the decision point and *int* is an integer, e.g. min(positive performance,1). The common choices(s) in a decision point is/are not included in maximum-minimum numbers of selection.

Common:

Syntax: common(Name1, yes). common(Name2, no).
Semantic: Describes the commonality of a decision point. *Name1* and *Name2* represent the names of decision points, e.g. common(promotion, yes) If the decision point is not common, the second slot in the predicate will become no e.g. common(punishment, no).

CHOICE

A choice is decision belonging to a specific decision point. For instance, in Figure 1, time *on* is a choice that belongs to the *negative performance* decision point. The following two predicates are used to describe a choice

Type:

Syntax: type(Name1,choice).
Semantic: Define the type. *Name1* represents the name of variant, e.g. type(recognition, choice).

Common

Syntax: common(Name1, yes). common(Name2, no).
Semantic: Describes the commonality of a choice, e.g. common(first reminder, yes). If the choice is not common, the second slot in the predicate will become no -as example-common(time on ,no).

Constraint dependency rules

The following six predicates are used to describe constraint dependency rules:

requires_c_c:

Syntax: requires_c_c(Name1, Name2).
Semantic: choice requires another choice. *Name1* represents the first choice and *Name2* represents the required choice, e.g. requires_c_c(promotion decision, recognition).

excludes_c_c:

Syntax: excludes_c_c (Name1, Name2).
Semantic: choice excludes choice. *Name1* represents the first choice and *Name2* represents the excluded choice, e,g. excludes_c_c(decrease level, high level).

requires_c_dp:

Syntax: requires_c_dp(Name1, Name2).
Semantic: Choice requires decision point. *Name1* represents the choice and *Name2* represents the required decision point, e.g. requires_c_dp(promotion decision, positive performance).

excludes_c_dp:

Syntax: excludes_c_dp(Name1, Name2).
Semantic: Choice excludes decision point. *Name1* represents the choice and *Name2* represents the excluded decision point, e.g. excludes_c_dp(non promotion decision, positive performance).

requires_dp_dp:

Syntax: requires_dp_dp(Name1, Name2).
Semantic: Decision point requires another decision point. *Name1* represents the first decision point and *Name2* represents the required decision point, e.g. requires_dp_dp(negative performance, punishment).

excludes_dp_dp:

Syntax: excludes_dp_dp(Name1, Name2).
Semantic: Decision point excludes another decision point. Name1 represents the first decision point and *Name2* represents the excluded decision point, e.g. excludes_dp_dp(negative performance, positive performance)

type(negative performance, decisionpoint). choiceof(negative performance, time on). choiceof(negative performance, N% salary). common(negative performance, no). min(negative performance, 1). max(negative performance, 2). requires_dp_dp(negative performance, punishment). excludes_dp_dp(negative performance, positive performance).

Table 2. Representation of Negative Performance

In addition to these predicates, we define two more predicates *select* and *notselect*. The *select* predicate is assigned for all selected choices. The *notselect* predicate prevents the choice from

the selection process, i.e. validate decision-making process. At the end, the choices that represent the final decisions are assigned by *select* and not(*notselected*) predicates.

4. Variability validation in DSS

Although variability is proposed as a technique of knowledge representation that provides a decision repository; validating this repository and decision making process is important.
In a decision making process, a decision maker selects the choice(s) from each decision point. The proposed method guides the decision maker by: 1) validating the constraint dependency rules, 2) automatically selecting (propagation and delete-cascade) decisions, and 3) provide explanation and corrective recommendation. In addition, the proposed method validates the decision-repository by detecting dead choices and logical inconsistency. In this section, six operations are illustrated. These operations are implemented using Prolog [29].

4.1 Validating the decision making process

4.1.1 Constraint dependency satisfaction

To validate the decision-making process, the proposed method triggers rules based on constraint dependencies. Based on the constraint dependency rules, the selected choice is either accepted or rejected. After that, the reason for rejection is given and correction actions are suggested. When the decision maker selects a new choice, another choice(s) is/are assigned by the *select* or *notselect* predicates. As example, in table 3: number 1, the choice *x* is

1. $\forall x, y: type(x, choice) \wedge type(y, choice) \wedge requires\_c\_c(x, y) \wedge select(x) \Longrightarrow select(y)$
2. $\forall x, y: type(x, choice) \wedge type(y, choice) \wedge excludes\_c\_c(x, y) \wedge select(x) \Longrightarrow notselect(y)$
3. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge requires\_c\_dp(x, y) \wedge select(x) \Longrightarrow select(y)$
4. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge excludes\_c\_dp(x, y) \wedge select(x) \Longrightarrow notselect(y)$
5. $\forall x, y: type(x, decisionpoint) \wedge type(y, decisionpoint) \wedge requires\_dp\_dp(x, y) \wedge select(x) \Longrightarrow select(y)$
6. $\forall x, y: type(x, decisionpoint) \wedge type(y, decisionpoint) \wedge excludes\_dp\_dp(x, y) \wedge select(x) \Longrightarrow notselect(y)$
7. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge select(x) \wedge choiceof(y, x) \Longrightarrow select(y)$
8. $\exists x \ \forall y: type(x, choice) \wedge type(y, decisionpoint) \wedge select(y) \wedge choiceof(y, x) \Longrightarrow select(x)$
9. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge notselect(y) \wedge choiceof(y, x) \Longrightarrow notselect(x)$
10. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge common(x, yes) \wedge choiceof(y, x) \wedge select(y) \Longrightarrow select(x)$
11. $\forall y: type(y, decisionpoint) \wedge common(y) \Longrightarrow select(y)$
12. $\forall x, y: type(x, choice) \wedge type(y, v\ decisionpoint) \wedge choiceof(y, x) \wedge select(x) \Longrightarrow sum(y,(x)) \leq max(y,z)$
13. $\forall x, y: type(x, choice) \wedge type(y, decisionpoint) \wedge choiceof(y, x) \wedge select(x) \Longrightarrow sum(y,(x)) \geq min(y,z)$

Table 3. The main rules in the proposed method

selected (assigned by *select* predicate (select(x)) and *x* requires the choice *y*. Then the system automatically assigns *y* by *select* predicate (select(y)). That means *y* is selected also. At the end, *select* and not *notselect* predicates represent the selections of decision-making process.
Table 3 shows the main rules in our proposed method. The proposed method contains thirteen main rules to validate the decision-making process. Rules 1 through 6 are used to validate constraint dependency rules. Rules 7 and 8 deal with relationships between decision point and their choice(s). Rules 10 and 11 satisfy the common property of decision points and choices. Rules 12 and 13 validate the maximum and minimum property of decision points. Appendix 1 describes the proposed rules in details.

4.1.2 Propagation and delete-cascade

This operation defines how some choices are selected automatically as a reaction to previous selection of other choices.

Definition 1: Selection of the choice *n*, *select(n)*, is propagated from selection of the choice *x*, *select(x)*, in three cases:

i. $\forall x,y,z,n:type(x,choice) \wedge choiceof(y,x) \wedge select(x) \wedge requires\_dp\_dp(y,z) \wedge type(n,choice) \wedge choiceof(z, n) \wedge common(n,yes) \Longrightarrow select(n)$.

If *x* is a choice and *x* belongs to the decision point *y* and *x* is selected, that means *y* is selected (rule 7), and the decision point *y* requires a decision point *z*, that means *z* is also selected (rule 5), and the choice *n* belongs to the decision point *z* and the choice *n* is common. It means the choice *n* is selected (rule 10).

ii. $\forall$ x,n: type(x,choice) $\wedge$ type(n,choice) $\wedge$ select(x)$\wedge$ requires_c_c(x,n) $\Longrightarrow$ select(n).

If the choice *x* is selected and it requires the choice *n*, it means the choice *n* is selected, (rule 1). The selection of choice *n* propagated from the selection of *x*.

iii. $\forall x,z,n:type(x,choice) \wedge select(x) \wedge type(z,decisionpoint) \wedge requires\_c\_dp(x,z) \wedge type(n,choice) \wedge choiceof(z,n) \wedge common(n) \Longrightarrow select(n)$.

If the choice *x* is selected and it requires the decision point z, that means the decision point *z* is selected (rule 3), and the choice *n* is common and belongs to the decision point *z* and that means the choice *n* is selected (rule 10). The selection of the choice *n* is propagated from the selection of *x*.

Delete-cascade operation

This operation validates the automated decision-making process during execution time. The following scenario describes the problem:
If choice *x* is selected in time *N* and the two choices *y* and *k* are propagated due to selection of *x*, then the decision list (at time *N*) = {*x*, *y*, *k*}. In time (*N* + 1), the choice *m* is selected, and *m* excludes *x*, then *x* is removed from the decision list. The decision list at time (*N* + 1) = {*m*, *y*, *k*}. The presence of the choices *y* and *k* is not correct. The choices *y* and *k* are not decision maker's choices. The following rule implements the delete-cascade operation.

$\forall x,y:type(x,choice) \wedge type(y,choice) \wedge requires\_c\_c(y,x) \wedge select(x) \wedge notselect(y) \Longrightarrow notselect(x)$.

For all choices *x*, and *y*; if the choice *y* requires *x* and *x* is selected and *y* is assigned by *notselect* predicate, that means *y* is excluded in the configuration process, and *x* was selected according to selection of *y (y requires x)*, then the presence of *x* after exclusion of *y* is not true. The output for this operation is the assignment of the choice *x* with *notselect* predicate. This assignment permits the proposed method to perform *delete-cascade* operation to verify the selections.

4.1.3 Explanation and corrective recommendation

This operation is defined (in this paper) for highlighting the sources of errors within decision- making process. The errors happened due to dissatisfaction of constraint dependency rules.

The general pattern that represents failure due to dissatisfaction of constraint dependency rules is:

Decision A excludes Decision B and Decision A is selected then Decision B fails to select.

In the proposed method, there are two possibilities for the decision: decision point or choice. Three possibilities for the exclusion constraint: choice excludes choice, choice excludes decision point, or decision point excludes decision point. We assign the predicate *notselect* to the excluded choice for preventing future select.

The following definition describes these possibilities in the form of rules:

Selection of choice *n*, *select (n)*, fails due to selection of choice *x*, *select(x)*, and assign by *notselec*t predicate in three cases:

i. $\forall x,y,n:type(x,choice) \wedge select(x) \wedge type(y,decisionpoint) \wedge choiceof(y,x) \wedge type(n,choice) \wedge excludes\_c\_dp(n,y) \Longrightarrow notselect(n)$.

If the choice *x* is selected, and it belongs to the decision point *y*, this means *y* is selected (Rule 7), and the choice *n* excludes the decision point *y*, this means *n* is assigned by *notselect* predicate.

ii. $\forall x,y,z,n:type(x,choice) \wedge select(x) \wedge type(y,decisionpoint) \wedge type(z,decisionpoint) \wedge type(n,choice) \wedge choiceof(y,x) \wedge choiceof(z,n)\ excludes\_dp\_dp(y,z) \Longrightarrow notselect(n)$.

If the choice *x* is selected and *x* belongs to the decision point *y*, that means *y* is selected (Rule 7), and if the decision point *y* excludes the decision point *z*, this means *z* is assigned by *notselect* predicate (rule 6), and the choice *n* belongs to decision point *z*, this means *n* is assigned by *notselect* predicate (rule 9).

iii. $\forall x,n:\ type(x,choice) \wedge select(x) \wedge type(n,choice) \wedge excludes\_c\_c(x,n) \Longrightarrow notselect(n)$.

If the choice *x* is selected, and *x* excludes the choice *n*, which means *n* is assigned by notselect predicate (rule 2).

Two examples are presented to express how the proposed method could be used for guiding the decision maker by explanation and corrective recommendation. Example 1 shows an interactive corrective recommendation mechanism. Example 2 shows how the proposed method validates decision maker in future based on his current selections.

Example 1

Suppose the decision maker selected *decrease level* as a punishment for one employee. After that, the decision maker selects *high level* as training for the same employee; the system rejects the choice and directs the decision maker to deselect *decrease level* first. Table 4 describes Example 1. This example represents rule (iii). The example illustrates how the proposed method guides decision makers to solve the rejection reason.

Example 2

The decision maker asks to select the choice *non promotion decision*, which is excludes *positive performance* decision point. The system accepts the choice and assigns the decision point *positive performance* by *notselect* predicate to validate future selections. Table 5 describes Example 2. The predicate *notselect (positive performance)* prevents the selection of its choices, Rule 9.

The proposed method guides the decision maker step by step (in each choice). If the decision maker's choice is invalid, the choice is immediately rejected and corrective actions are suggested, see Example 1. Moreover, *notselect* predicate can be assigned to some choices according to decision maker's selection, see Example 2. The *notselect* predicate prevents the decision maker from future errors; see Table 3: Rule 9.

? select (decrease level). You have to deselect high level

Table 4. Example 1

? select (non promotion decision). Yes notselect (positive performance) added to knowledge base.

Table 5. Example 2

4.2 Validate decision repository

4.2.1 Logical inconsistency detection

Inconsistency occurs from contradictions in constraint dependency rules. It is a very complicated problem. Inconsistency has different forms and it can occur between: groups (as example: (A and B) require (D and C) and (D and C) exclude (A and B)), group and individual (as example: (A and B) require D and D excludes (A and B)), or between individuals only (as example**:** (A require**s** B and B requires C and C excludes A)). A, B, C and D could be choices or decision points.

In this paper, we suggest rules to detect logical inconsistency between individuals. The rules that can be used to detect logical inconsistency (between individuals) are categorized in three groups. Each group contains two rules.

Group 1

In this group, we discuss the constraint dependency relation between two decisions from the same type (decision point or choice).

The first decision requires the second one while the second decision excludes the first one. The logical inconsistency between two decisions could be indirect, e.g. A requires B and B requires C and C excludes A. Therefore, to transfer the logical inconsistency to be directly within two decisions, we define these transfer rules:

i. $\forall x,y,c:type(x,choice) \wedge type(y,choice) \wedge type(c,choice) \wedge requires\_c\_c(x,y) \wedge requires\_c\_c(y,c) \Longrightarrow requires\_c\_c(x,c).$

ii. $\forall x,y,c:type(x,decisionpoint) \wedge type(y,decisionpoint) \wedge type(c,decisionpoint) \wedge requires\_dp\_dp(x,y) \wedge requires\_dp\_dp(y,c) \Longrightarrow requires\_dp\_dp(x,c).$

The following rules detect inconsistency in group 1:

i. $\forall x,y:type(x,choice) \wedge type(y,choice) \wedge requires\_c\_c(x,y) \wedge excludes\_c\_c(y,x) \Longrightarrow error.$

If the choice x requires the choice y which means selection of x leads to selection of y (Rule 1). In addition, choice y excludes the choice x which means if y selected, x must not be selected (Rule 2). This is an error.

ii. $\forall x,y:type(x,decisionpoint) \wedge type(y,decisionpoint) \wedge requires\_dp\_dp(x,y) \wedge excludes\_dp\_dp(y,x) \Longrightarrow error.$

If the decision point *x* requires the decision point *y* that means selection of *x* leads to selection of *y* (Rule 5), and the decision point *y* excludes the decision point x which means if *y* is selected *x* must not be selected (Rule 6). This is an error.

Group 2

In this group, we discuss the constraint dependency relation between two decision points. At the same time, there is a contradictory relation between one choice (belongs to the first decision point) and the second decision point. The following rules illustrated group 2:

i. $\forall x,y,z:type(x,choice) \wedge common(x,yes) \wedge type(y,decisionpoint) \wedge choiceof(y,x) \wedge type(z,decisionpoint) \wedge requires\_dp\_dp(y,z) \wedge excludes\_c\_dp(x,z) \Longrightarrow error.$

If the common choice *x* belongs to the decision point *y*, and *x* excludes the decision point *z*, which means if *x* selected, no choice belonging to *z* must be selected (Rule 4, and Rule 9), and the decision point *y* requires the decision point *z* which means if *y* is selected *z* must also be selected(Rule 5). Selection of the decision point *y* means selection of the common choice *x* (Rule 10) but *x* excludes *z*. This is an error.

ii. $\forall x,y,z:type(x,choice) \wedge type(y,decisionpoint) \wedge choiceof(y,x) \wedge type(z,decisionpoint) \wedge excludes\_dp\_dp(y,z) \wedge requires\_c\_dp(x,z) \Longrightarrow error.$

If the choice *x* belongs to the decision point *y* and *x* requires the decision point *z* that means if *x* selected *z* should be selected (Rule 3). The decision point y excludes the decision point *z* that means if one of the choices belongs to *y* is selected none belongs to *z* should be selected (Rules 6, 7, and 9). *X* requires *z* is an error.

Group 3

In this group, we discuss the constraint dependency relation between two decision points. At the same time, there is a contradictory relation between their choices. The following rules illustrates group 3:

i. $\forall x,y,n,z:type(x,choice) \wedge type(y,decisionpoint) \wedge choiceof(y,x) \wedge type(n,choice) \wedge type(z,decisionpoint) \wedge choiceof(z,n) \wedge common(n,yes) \wedge excludes\_c\_c(x,n) \wedge requires\_dp\_dp(y,z) \Longrightarrow error.$

The common choice *x* belongs to the decision point *y* and the common choice *n* belongs to *z*. The decision point *y* requires the decision point *z* that means if *y* selected then *z* must be selected, (Rule 5), and selection of *y* and *z* means selection of *x* and *n*, (Rule 10). *X* excludes *n* is an error.

ii. $\forall x,y,n,z:type(x,choice) \wedge type(y,decisionpoint) \wedge choiceof(y,x) \wedge type(n,choice) \wedge type(z,decisionpoint) \wedge choiceof(z,n) \wedge requires\_c\_c(x,n) \wedge excludes\_dp\_dp(y,z) \Longrightarrow error.$

If choice *x* belongs to the decision point *y*, and the choice *n* belongs to the decision point *z*, and *x* requires *n* which means if *x* is selected, *n* should also be selected (Rule1). Selection of the choice *x* means selection of the decision point *y*, and selection of choice *n* means selection of decision point *z* (Rule 7). The decision point y excludes the decision point *z* t which means if one of the choices belonging to *y* is selected, then none belonging to *z* must be selected (Rules 6, 7, and 9). *X* requires *n* is an error.

4.3 Dead decision detection

A dead decision is a decision that never appears in any legal decision process. The only reason to prevent a decision from being included in any decision process is that there is a

common choice/decision point that excludes this decision. The general form for describing a dead decision is:

Decision A excludes decision B and decision A is common then decision B is a dead decision.

According to the proposed method, there are two possibilities for a decision: decision point or choice, two possibilities for decision (A) to be common and three possibilities for the exclusion.

Two possibilities for decision (A) to be common:

1. Common decision point:
$\exists A$ *:type(A, decisionpoint)* $\wedge$ *common(A,yes).*
2. Common choice belongs to common decision point:
$\exists A,C$*:type(A, choice)* $\wedge$ *type(C, decisionpoint)* $\wedge$ *choiceof(C,A)* $\wedge$*common(C,yes)* $\wedge$*common(A,yes)***.**

Three possibilities for the exclusion constraint:

3. Choice excludes choice:
$\exists A,B$*: type(A, choice)* $\wedge$ *type(B, choice)* $\wedge$ *excludes_c_c(A,B).*
4. Choice excludes decision point:
$\exists A,C$ *:type(A,choice)* $\wedge$ *type(C, decisionpoint)* $\wedge$ *excludes_c_dp(A,C).*
5. 5. Decision point excludes decision point:
$\exists A,C$*: type(A, decisionpoint)* $\wedge$ *type(C, decisio point)* $\wedge$ *excludes_dp_dp(A,C).*

If we apply the two possibilities of common decision to the three possibilities of the exclusion constraint then we have six possibilities for satisfying the general form of the dead decision. These possibilities are (1, 3),(1, 4),(1, 5),(2, 3),(2, 4),(2, 5). The possibilities (1, 3), (1, 4) and (2, 5) are excluded because decision *A* cannot be decision point and choice at the same time. Therefore, all the possible scenarios for the dead decision are: (1, 5), (2, 3), (2, 4). These scenarios are represented by the following rules:

i. $\forall A,B,C$*:type(A,decisionpoint)*$\wedge$*common(A,yes)*$\wedge$*type(C,decisionpoint)*$\wedge$*excludes_dp_dp(A,C)*$\wedge$ *type(B,choice)* $\wedge$ *choiceof(C ,B)* $\Longrightarrow$ *dead_ decision (B).*

The decision point *C* in the above rule represents a dead decision point. All choices belonging to a dead decision point are dead decisions.

ii. $\forall A,B,C$*:type(A,choice)*$\wedge$*type(C,decisionpoint)*$\wedge$*choiceof(C,A)*$\wedge$*common(C,yes)*$\wedge$*common(A,yes)*$\wedge$ *type(B, choice)* $\wedge$*excludes_c_c(A,B)* $\Longrightarrow$ *dead_ decision(B).*

iii. $\forall A,B,C,D$*:type(A,choice)*$\wedge$*type(C,decisionpoint)*$\wedge$*choiceof(C,A)*$\wedge$*common(C,yes)*$\wedge$*common(A,yes)*$\wedge$*type(B,choice)*$\wedge$*type(D,decisionpoint)*$\wedge$*choiceof(D,B)*$\wedge$*excludes_c_dp(A,D)*$\Longrightarrow$ *dead_ decision (B).*

Rule (i) represents case (1, 5), rule (ii) represents case (2, 3) and rule (iii) represents case (2, 4). The decision point *D* in rule 3 represents a dead decision point. The choice *B* represents all choices belonging to *D*.

5. Scalability testing

Scalability is approved as a key factor in measuring applicability of the techniques dealing with variability modeling [23]. The output time is a measurement key for scalability. A system considers scalable for specific problem if it can solve this problem in a reasonable time.

In this section, we discuss the experiments, which are developed to test the scalability of the proposed method.

5.1 The experiment

In the following, we describe the method of our experiment:

- **Generate the decision repository:** repository is generated in terms of predicates (Decision points, and choices). We generated four sets containing 1000, 5000, 15000, and 20000 choices. Choices are defined as numbers represented in sequential order, as example: In the first set (1000 choices) the choices are: 1, 2, 3,…, 1000. In the last set (20000 choices) the choices are: 1, 2, 3, …, 20000. The number of decision point in each set is equal to number of choices divided by five, which means each decision point has five choices.
- **Define the assumption:** We have three assumptions: i) each decision point and choice has a unique name, ii) each decision point is orthogonal, and iii) all decision points have the same number of choices.
- **Set the parameters**: The main parameters are the number of choices and the number of decision points. The remaining eight parameters (common choice, common decision point, choice requires choice, choice excludes choice, decision point requires decision point, decision point excludes decision points, choice requires decision point, and choice excludes decision point) are defined as a percentage. Three ratios are defined: 10%, 25%, and 50%. The number of the parameters related to choices (such as; common choice, choice requires choice, choice excludes choice, choice requires decision point, and choice excludes decision point) is defined as a percentage of the number of the choices. The number of parameters related to decision point (such as; decision point requires decision point) is defined as a percentage of the number of decision points. Table 6 represents snapshots of an experiment's dataset, i.e. the decision repository in our experiments.
- **Calculate output**: for each set, we made thirty experiments, and calculated the execution time as average. The experiments were done with the range (1000-20000) choices, and percentage range of 10%, 25%, and 50%.

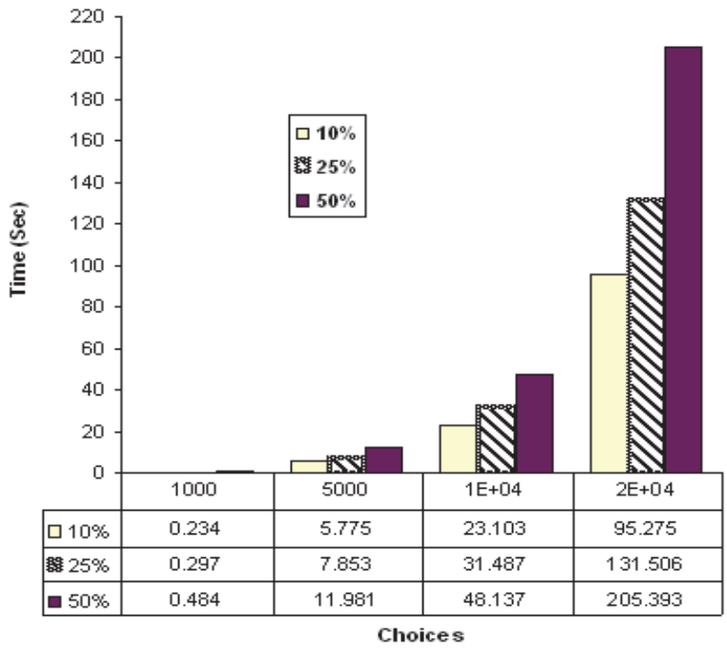
In the following section, the experiments that are done for dead decision detection, explanation, and logical inconsistency detection are discussed. The rest two operations (constraint dependency satisfaction, and propagation and delete-cascade) are working in semi-auto decision environment, where some decisions are propagated automatically according to decisions made. In semi-auto decision environment, the scalability is not a critical issue.

```
type(dp1,decisionpoint).
type(1,choice).
variants(dp1,1).
common(570,yes).
Common(dp123,yes).
requires_c_c(7552,2517).
requires_dp_dp(dp1572,dp1011).
excludes_dp_dp(dp759,dp134).
excludes_c_c(219,2740).
requires_c_dp(3067,dp46).
excludes_c_dp(5654,dp1673).
```

Table 6. Snapshot of experiment's dataset

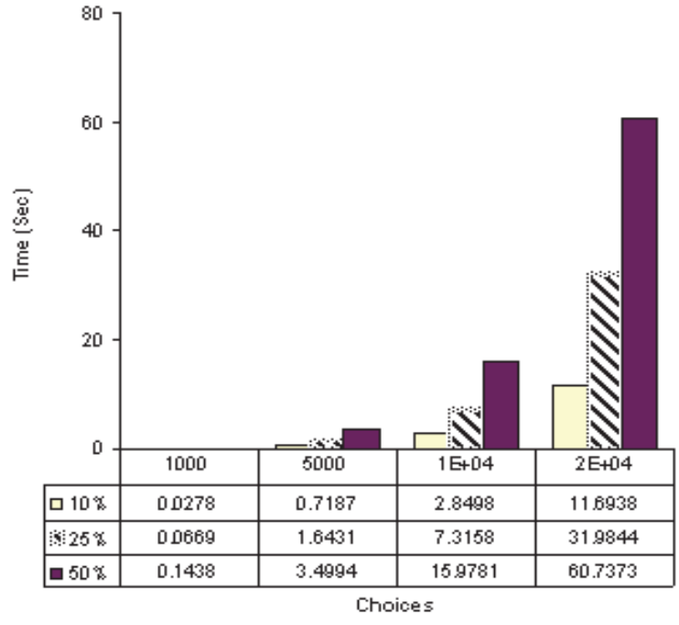
5.2 Empirical results

Dead Decision Detection: Figure 2 illustrates the average execution time. For (20,000) choices and 50% of constraint dependency rules, the execution time is 3.423 minutes which can be considered as a reasonable time. The output of each experiment is a result file containing the dead decisions.



	1000	5000	1E+04	2E+04
10%	0.234	5.775	23.103	95.275
25%	0.297	7.853	31.487	131.506
50%	0.484	11.981	48.137	205.393

Fig. 2. Dead decision detection results



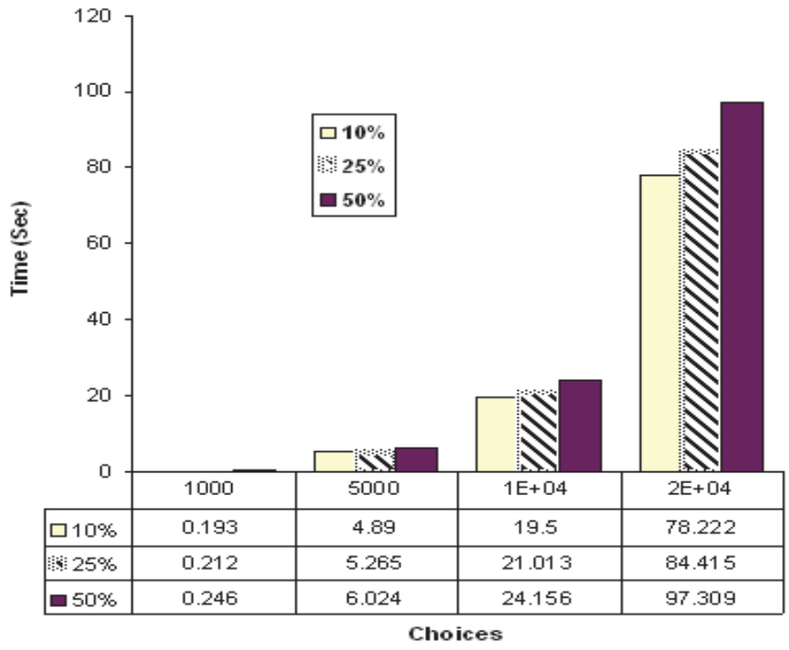
	1000	5000	1E+04	2E+04
10%	0.0278	0.7187	2.8498	11.6938
25%	0.0669	1.6431	7.3158	31.9844
50%	0.1438	3.4994	15.9781	60.7373

Fig. 3. Explanation results

Explanation

This process defines the source of error that might occur when the new choice is selected. To evaluate the scalability of this operation, we define additional parameter, the predicate select(C): where C is random choice. This predicate simulates decision maker selection. Number of select predicate (defined as a percentage of number of choices) is added to the knowledge-based for each experiment, and the choice C is defined randomly (within scope of choices). Figure 3 illustrates the average execution time. The output of each experiment is a result file containing the selected choices and the directive messages.

Logical Inconsistency-Detection: Figure 4 illustrates the average execution time to detect inconsistency in FM Range from 1000 to 20,000 choices



	1000	5000	1E+04	2E+04
10%	0.193	4.89	19.5	78.222
25%	0.212	5.265	21.013	84.415
50%	0.246	6.024	24.156	97.309

Fig. 4. Logical Inconsistency Detection

6. Conclusion and future work

Representing knowledge objects and the relation between them is the main issues of the modern knowledge representation techniques. We suggest variability for representing knowledge objects in DSS. By introducing variability to represent knowledge in DSS we can get both formalized knowledge representation in decision repository and support decision-making process by validation operations. Decision selection processes are validated using constraint dependency rules, propagation and delete cascade, and explanation and corrective recommendation operations. Decision repository is validated by detecting logical inconsistency and dead choices. In [5] it states, "developing and using a mathematical model in a DSS, a decision maker can overcome many knowledge-based errors". For this reason, the proposed method is supported by FOL rules.

We plan to test and validate this work using real data and real life case studies from industry. In addition, new operations are needed to validate DSS.

7. Appendix A

Explanation of the rules in table 3

Rule 1:

For all choice *x* and choice *y*; if *x* requires *y* and *x* is selected, then *y* is selected.

Rule 2:

For all choice *x* and choice *y*; if *x* excludes *y* and *x* is selected, then *y* is assigned by *notselect* predicate.

Rule 3:

For all choice *x* and decision point *y*; if *x* requires *y* and *x* is selected, then *y* is selected. This rule is applicable as well, if the decision point is selected first and it requires a choice:

$\forall$ x, y: type(x, choice) $\wedge$ type(y, decisionpoint) $\wedge$ require_c_dp(x, y) $\wedge$ select(y) $\Longrightarrow$ select(x)

For all choice *x* and decision point *y*; if *x* requires y and *y* is selected, then *x* is selected.

Rule 4:

For all choice *x* and decision point *y*; if *x* excludes *y* and *x* is selected, then *y* is assigned by *notselect* predicate. This rule is applicable as well, if the decision point is selected first and it requires a choice:

$\forall x, y$: *type(x, choice)* $\wedge$ *type(y, decisionpoint)* $\wedge$ *exclude_c_dp(x, y)* $\wedge$ *select(y)* $\Longrightarrow$ *notselect(x)*

For all choice *x* and decision point *y*; if *x* excludes *y* and *y* selected, then *x* is assigned by *notselect* predicate.

Rule 5:

For all decision point *x* and decision point *y*, if *x* requires *y* and *x* selected, then *y* is selected.

Rule 6:

For all decision point *x* and decision point *y*, if *x* excludes *y* and *x* is selected, then *y* is assigned by *notselect* predicate.

Rule 7:

For all choice *x* and decision point *y*, where *x* belongs to *y* and *x* is selected, that means *y* is selected.

This rule determines the selection of decision point if one of its choices was selected.

Rule 8:

For all decision point *y* there exists of choice *x*, if *y* selected and *x* belongs to *y*, *x* is selected.

This rule states that if a decision point was selected, then if there is choice(s) belonging to this decision point , must be selected.

Rule 9:

For all choice *x* and decision point *y*; where *x* belongs to *y* and *y* defined by predicate *notselect(y)*, then *x* is assigned by *notselect* predicate. This rule states that if a decision point was excluded, then none of its choices is selected.

Rule 10:

For all choice *x* and decision point *y*; where *x* is a common, *x* belongs to y, and *y* is selected, then *x* is selected. This rule states that if a choice is common and its decision point selected then it is selected.

Rule 11:

For all decision point *y*; if *y* is common, then *y* is selected. This rule states that if a decision point is common then it is selected in any decision process.

Rule 12:

For all choice *x* and decision point *y*; where *x* belongs to *y* and *x* is selected, then the summation of *x* must not be less than the maximum number allowed to be selected from y.

Rule 13:
For all choice *x* and decision point *y*; where *x* belongs to *y* and *x* is selected, then the summation of *x* must not be greater than the minimum number allowed to be selected from *y*.

Rules 12 and 13 validate the number of choices' selection considering the maximum and minimum conditions in decision point definition (cardinality definition). The predicate sum(y, (x)) returns the summation number of selected choices belongs to decision point y.

8. References

[1] Antal, P." Integrative Analysis Of Data, Literature, And Expert Knowledge By Bayesian Networks",, Phd Thesis , Katholieke University Leuven, 2008.

[2]Batory, D., Benavides, D., Ruiz-Cortés, A., "Automated Analyses of Feature Models: Challenges Ahead", *Special Issue on Software Product Lines , Communications of the ACM*, 49(12) December 2006,pp. 45 – 47.

[3]Brewster, C., O'Hara, K., "Knowledge Representation with Ontologies: The Present and Future", Intelligent Systems, IEEE 1094-7167/04, 19(1), 2004.

[4]Christiansson, P. "Next Generation Knowledge Management Systems For The Construction Industry", Paper w78-2003-80, Proceedings 'Construction It Bridging The Distance. Cib Publication 284. Auckland 2003, pp. 80-87.

[5] Celderman, M. "Task Difficulty, Task Variability and Satisfaction with Management Support systems:Consequences and Solutions", Research Memorandum 1997-53, Vrije University, Amsterdam, 1997.

[6] Czarnecki, K., Hwan, C., Kim, P. "Cardinality-based Feature Modeling and Constraints: A Progress Report", International Workshop on Software Factories at (OOPSLA'05), San Diego California, 2005.

[7] Densham, P. J. "Spatial decision support systems", Geographical information systems: principles and applications, London, Longman, 1991,pp. 403 - 412.

[8] Froelich, W., Wakulicz-Deja, A. " Associational Cognitive Maps for Medical Diagnosis Support", Intelligent Information Systems Conference , Zakopane, Poland, 2008, pp. 387–396.

[9] Grégoire, E. "Logical Traps in High-Level Knowledge and Information Fusion", Specialist Meeting Conference on Information Fusion for Command Support (IST-055/RSM-001), The Hague, The Netherlands,2005.

[10] Haas, S. W. "Knowledge Representation, Concepts, and Terminology: Toward a Metadata Registry for the Bureau of Labor Statistics", Final Report to the United States Bureau of Labor Statistics, School of Information and Library Science, University of North Carolina at Chapel Hill, July 1999.

[11] Hale, P., Scanlan, J., Bru, C. " Design and Prototyping of Knowledge Management Software for Aerospace Manufacturing", 10th ISPE International Conference on Concurrent Engineering, Madeira Island , Portugal, 2003.

[12] Kang, K., Hess, J., Novak, W. Peterson, S. "Feature oriented domain analysis (FODA) feasibility study",Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[13] Lu, J., Quaddus, M.A., Williams, R. "Developing a Knowledge-Based Multi-Objective Decision Support System", the 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, 2000.

[14] Malhotra, Y. Why knowledge management fails? Enablers and constraints of knowledge management in human enterprises, handbook of knowledge management, chapter 30, springer, 2002, pp. 568-576.

[15] Mikulecky, P., Olsevicov´a, K., Ponce, D. "Knowledge-based approaches for river basin management", *the journal Hydrology and Earth System Sciences*, Discuss., 4, 1999-2033, 2007.

[16] Molina, M. "Building a decision support system with a knowledge modeling tool", *Journal of Decision Systems*, Lavoisier, Volume 14 (3), 2005.

[17] Martinez, S.I. A formalized Multiagent decision support in cooperative environments, Doctoral Thesis, Girona Catalonia, Spain, 2008.

[18] Padma, T., Balasubramanie, P., Knowledge based decision support system to assist work-related risk analysis in musculoskeletal disorder, *Knowledge-Based Systems,* Elsevier, 22, 2009 ,72–78.

[19] Peleg M, Tu S. W.: Decision Support, Knowledge Representation and Management in Medicine, 2006 IMIA Yearbook of Medical Informatics, Reinhold Haux, Casimir Kulikowski, Schattauer, Stuttgart, Germany, BMIR-2006-1088, 2006.

[20] Pohl, k., G. Böckle, Linden, F. J. van der. *Software product line engineering, Foundations, Principles, and Techniques*, Springer, Berlin, Heidelberg, 2005, 490.

[21] Pomerol, J., Brezillon, P. Pasquier, L., "Operational Knowledge Representation for Practical Decision Making", The 34th Hawaii International Conference on System Sciences, 2001.

[22] Roth, B.M., Mullen, J. D. *Decision Making: Its Logic, And practice*, Lanham, MD:Rowman & littlefield, 2002.

[23] Segura S. ,"Automated Analysis of Feature Models using Atomic Sets", the 12th international conference of software product line, Limerick Irland,2008.

[24] Schreiber, G., Akkermans, H., Anjewierden, A., Dehoog, R. Shadbolt, N . Vandevelde, W. Wieling, B, *Knowledge Engineering and Management:The CommonKADS Methodology*, MIT Press, 1999.

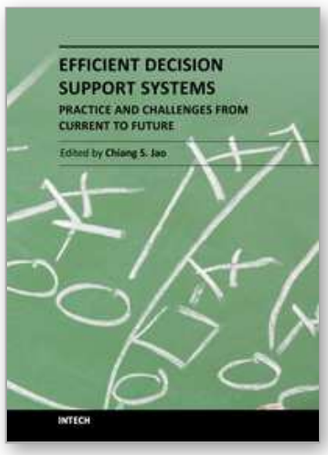
[25] Suh C.K. "An Integrated Two-Phased Decision Support System for Resource Allocation", *Wseas transactions on business and economics*,Volume 4(11), November 2007.

[26] Tuomi I., " The Future of Knowledge Management", Lifelong Learning in Europe (LLinE), vol VII, issue 2/2002, , 2002 ,pp. 69-79.

[27] Williams, M., Dennis, A. R., Stam, A., Aronson, J. "The Impact of DSS Use and Information Load on Errors and Decision Quality", Working Papers on Information Systems, Indiana University, USA, Sprouts: Working Papers on Information Systems, 4(22). http://sprouts.aisnet.org/4-22, 2004.

[28] Williams, M. H. Integrating Ontologies and Argumentation for decision-making in breast cancer, PhD Thesis, University College London, 2008.

[29] Wielemaker J.: SWI-Prolog (Version 5.6.36) free software, Amsterdam, University of Amsterdam, 2007.



Efficient Decision Support Systems - Practice and Challenges From Current to Future

Edited by Prof. Chiang Jao

ISBN 978-953-307-326-2
Hard cover, 542 pages
Publisher InTech
Published online 09, September, 2011
Published in print edition September, 2011

This series is directed to diverse managerial professionals who are leading the transformation of individual domains by using expert information and domain knowledge to drive decision support systems (DSSs). The series offers a broad range of subjects addressed in specific areas such as health care, business management, banking, agriculture, environmental improvement, natural resource and spatial management, aviation administration, and hybrid applications of information technology aimed to interdisciplinary issues. This book series is composed of three volumes: Volume 1 consists of general concepts and methodology of DSSs; Volume 2 consists of applications of DSSs in the biomedical domain; Volume 3 consists of hybrid applications of DSSs in multidisciplinary domains. The book is shaped upon decision support strategies in the new infrastructure that assists the readers in full use of the creative technology to manipulate input data and to transform information into useful decisions for decision makers.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Abdelrahman Osman Elfaki, Saravanan Muthaiyah, Chin Kuan Ho and Somnuk Phon-Amnuaisuk (2011). Knowledge Representation and Validation in a Decision Support System: Introducing a Variability Modelling Technique, Efficient Decision Support Systems - Practice and Challenges From Current to Future, Prof. Chiang Jao (Ed.), ISBN: 978-953-307-326-2, InTech, Available from: http://www.intechopen.com/books/efficient-decision-support-systems-practice-and-challenges-from-current-to-future/knowledge-representation-and-validation-in-a-decision-support-system-introducing-a-variability-model

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen