Introducing a Method for Modeling Knowledge Bases in Expert Systems Using the Example of Large Software Development Projects

Franz Felix Füssl, Detlef Streitferdt
Institute for Computer and Systems Engineering
Technische Universität Ilmenau
Ilmenau, Germany

Weijia Shang
Computer Engineering Department
Santa Clara University
Santa Clara (California), United States of America

Anne Triebel
Institute for Business Information Systems Engineering
Technische Universität Ilmenau
Ilmenau, Germany

***Abstract*—Goal of this paper is to develop a meta-model, which provides the basis for developing highly scalable artificial intelligence systems that should be able to make autonomously decisions based on different dynamic and specific influences. An artificial neural network builds the entry point for developing a multi-layered human readable model that serves as knowledge base and can be used for further investigations in deductive and inductive reasoning. A graph-theoretical consideration gives a detailed view into the model structure. In addition to it the model is introduced using the example of large software development projects. The integration of Constraints and Deductive Reasoning Element Pruning are illustrated, which are required for executing deductive reasoning efficiently.**

Keywords*—Knowledge Engineering; Ontology Engineering; Knowledge Modelling; Knowledge Base; Expert System; Artificial Intelligence; Deductive Reasoning Element Pruning

I. Introduction

IT technologies are subjects to a fast changeable field of application. Software development teams have to adapt continuously for fitting newest stakeholder needs and finding success in the market. Especially success of large software development projects for example product line developments depends on many different influencing factors, introduced in [1]. These influencing factors determine for example the composition of teams, the choice of software tools or the selection of a suitable software development process.

There are a couple of previous and current projects with the goal of developing an open source expert system (ES) including the ability of machine learning in a specific field. Examples are [2], the "scikit-learn" library [3], the "Mlpy" library [4] or "Orange" library [5]. As opposed to these projects and publications the project behind this paper focuses on large software developments that typically have many various influences and a large set of required or requested software tools and business artifacts.

The most important part of an ES is the basis of decisions. There are a couple of systems, based on a simple decision tree or relational data models [6]. But currently in the domain of software developments there is no appropriated model for illustrating knowledge bases (KB) [6], which are necessary for automated handling machine learning and deductive reasoning. For this reason an abstract human readable meta-model (defined in [7]) should be developed that deals as architectural basis for further investigations in autonomous decision making having regard to different specific influences as project-specific, personal, economic-driven, product-related or technology-based.

A. Knowledge bases in Expert Systems

According to basic literature as [8], [9] or [10] an ES is a knowledge-based system that is used for intelligent assistance, decision making or problem solving. Main goal of the research project behind this paper is to develop a system that is able to detect any objects that are helpful to bring a software project to success. Thus the system needs to make decisions for solving a specific problem. To do so, it is necessary, to have a profound KB what can be used for inference, in particular deductive and inductive reasoning.

So what exactly is a KB and why it is important to consider? According to [11] or [12] KB are specialized bodies or nets of knowledge and skills. So it is a construct of information, data and associations, where knowledge can be generated and derived by "heuristics or informal 'rules of thumb' experts" and "reasoning methods" [13]. In the field of this research project the KB, which is to develop, contains knowledge of software developers, project managers, software architects, software producer or experts and consultant in the field of software development processes.

B. The origins of the model: Ontology, Topic Map and Artificial Neural Network

According to [14] an artificial neural network (ANN) is organized into layers with processing elements, called units. Every unit has its own specific setup, but they are similar in activation events and output functions. Associations can be done among units of the same layer and between elements on different layers. The units are associated by weighted connection paths. As described in [15] "successful training [of ANN] can result […] in performing tasks such as predicting an output value, classifying an object […] and completing a known pattern".

An ANN seems to be the right KB to pursue the goal of this paper. But there are differences to the underlying KB of this paper. ANN work with an unspecified number of layers. They have a known input pattern and a known output pattern. In case of a multi-layer feedforward ANN there is at least one hidden layer in-between these patterns [16], which leads to a worse human readability. Also it is not provided to have different views to the knowledge, for instance a descriptive and a deciding view. The model should be able to perform inferring processes, making decisions and edit knowledge with a focus to human-readability.

With searching a possible solution for solving the human-readability problem, Ontologies have to be mentioned. As described in [6] they serve as method for representing knowledge and it is possible to integrate machine learning by 'Ontology learning'. The problem of Ontologies is the degree of formalization, which is too low for scalable reasoning. Thus generic usable and efficient deductive reasoning algorithms are difficult to integrate and not a goal of Ontologies. Nevertheless the main idea of Ontologies, the descriptive functionality, has been used for developing the model behind this work. [6]

Another possibility to represent knowledge is creating Topic Maps. But here are no approaches for machine learning integration. As with Ontologies the focus of Topic Maps is the presentational view of knowledge. [6]

Taken as a whole, the KB, which builds a foundation of the model in this work, is lightly adopted to multi-layered feedforward ANN with advantages of Ontologies and Topic Maps.

II. Modeling the Knowledge Base

The architecture of the knowledge base can be described as five-layered meta-model. The layers represent different abstraction levels, where information can be stored and processed. Fig. 1 shows these five layers of the meta-model.

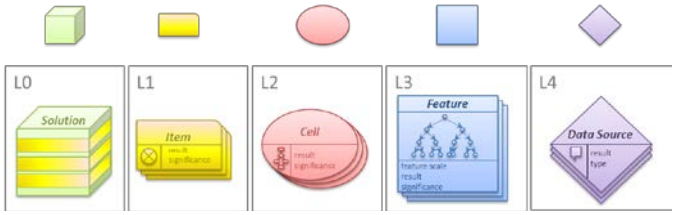

Fig. 1. Multi-layerd Meta-Model: Layer Overview

One layer contains at least one *element*, exemplified by repetitive geometric figures of Fig. 1. Elements are associated by different connection types. Thus association rule learning is possible, as it is used for example in artificial intelligence (AI) or Data Mining [17]. With these associations, elements have relations to each other within the same layer or through different layers. The model provides best possibilities for using the principle of loose coupling, which leads to high interchangeability and extensibility of all containing elements [18]. Each element contains implicitly a problem, a specific way for finding the solutions (logical part) and a result for using the element explicitly. The models hierarchical and logical structure helps to break these problems down into manageable pieces, which is one of the major and well known paradigms in AI [19], [20], [21].

Beginning with the general description of terminology the models characterization is followed below:

A. Layer description and terminology

The Layer ***L4*** contains the *Data Source* elements of the KB. It serves as application layer and forms the main communication between the AI system and project participants (users), hardware or software interfaces. Elements are for instance questions, measuring tools, sensor technologies or API definitions. This layer collects primary circumstances by a simple request-response method. Thus it is possible to gather actual issues and specific factors of influence.

Each factor of influence is stored as *Feature* in layer ***L3***. Examples of these Features are "project budget", "operating system", "personal motivation" or "personal experience". Impacts of each feature correlate closely with research and technological development, which is why layer *L*3 is subject to high degree of variability. For simplifying the analysis of results, features are classified in nominal scaled, ordinal scaled and metric scaled. This classification depends on the connected data source element and bases for example on different types of questions as "multiple-choice" and "single-choice" or on different types of input data, as Integer and String. Input data mean typically data generated by measurements. Questions could have this data type too, especially when user input is necessary.

Each Feature is associated with at least one *Cell* that is collected by another layer of the model: ***L2***. According to Landauer each Cell could be described as context block [22], which collects data by their associations and interprets it as information by deposited algorithms. Thus they represent the basis for generating knowledge. In addition to analyze specifically adjacent Features, Cells are able to access other Cells by connecting among each other and involving the associated result into the own analyses. They are weighted differently according to their number of associations. Cells can be dependent on each other, for instance "Scrumwise" as software solution and "Scrum" as development process.

The information that is stored in Cells is used by specific *Items*, which build layer ***L1***. Each Item contains an abstract component that could be necessary for realizing the project. Items serve as partial solution for reacting to a determined problem. For that reason all Items have to examine carefully, how they conduce to project's success. They can be used in different parts and steps of the project. For instance, one Item symbolizes "requirements engineering", which is necessary for product management and very important for project success. Further examples of Items are "software development process", which has to be ascertained or "software architecture pattern".

For determination Cells and Items in a more understandable manner: *L*1 (layer of Items) serves as 'descriptive' view on the information of the KB, while *L*2 (layer of Cells) builds a 'deciding' view within the system using the KB. Cells contain something concrete while Items are more a general view into the KB.

The solution layer **$L0$** represents a complete build package for solving a predefined problem, for example finding the 'projectalized' development process or a customized developer environment. The decision of combining project relevant elements bases on a simple suitability test. Each association of every single Item verifies its project suitability. Then the Items determine their linked Cell with the highest suitability value. The corresponding results are abstracted to a project-specific solution.

B. Mathematical consideration as graph

When describing the five-layered meta-model with basic definitions of graph and set theory, for example by [23] or [24], the model is a weighted directed graph $G = (V, E, i)$ without loops and a non-regular property. The layers ($L0\ to\ L4$) are sets of ordered pairs (V, E) with

- $V(G)$ as finite set of all nodes (elements of the layers)
- $E(G)$ as set of all edges (associations) and
- $i = i_G$ as mapping that assigns to each edge $e \in E$ a pair $i(e) = \{x, y\}$ with elements $x, y \in V$.

Set V can be divided into five subsets, as shown in Fig. 2:

$$L_0, L_1, L_2, L_3, L_4 \subset V(G)$$

They contain each element of the different layers of the model. L_2 (Cells layer) and L_1 (Items layer) are exceptional as opposed to the other layers. They are induced subgraphs $C_1, C_2 \subset G$. Within L_1 and L_2 it is possible to build edges between the nodes (on the same layer). According to [25] C_1 and C_2 have maximum associations of

$$\binom{n}{2} = \frac{n(n-1)}{2}\ with\ n = |L_2| \text{ or}$$

$$\binom{m}{2} = \frac{m(m-1)}{2}\ with\ m = |L_1|$$

This maximum achievable number of nodes of L_2 and L_1 implies the completeness of the respective subgraphs. It can only be achieved by associating each element of one layer with every other element of the same layer.

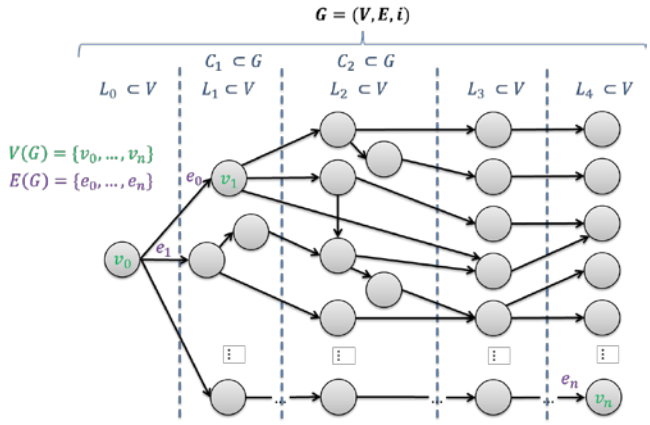

Fig. 2. Model illustration of an example as graph

By integration of weights deductive reasoning can be accelerated. Goal is processing the graph on prioritized paths, for faster reaching important elements (nodes, information). Here the importance of an element can be concluded from the degree centrality, similar to the PageRank-Algorithm [26]. According to this the significance of a node depends mainly on the number of its edges. [26]

The set $E_G(v)$ is the set of all adjacent edges of a node $v \in V$. Further the set $E_G^+(v)$ is defined as a set containing all edges to successors and $E_G^-(v)$ as a set with all edges to predecessors of one node v.

The weighting is made by assigning an edge $e \in E_G(v)$ a weight $\omega(e)$, which is a real number $\omega : E \to \mathbb{R}$ [27]. It can be simplified by $\omega : E \to \mathbb{Z}$. Each node is able to attach its degree $d_G(v)$ as weight to all of its adjacent edges. If an edge has already had a higher weight, this edge keeps the original weight value:

$$\forall\ v \in V: \omega(e_v) < d_G(v) \to \omega(e_v) = d_G(v)$$
$$with\ e_v \in E_G(v)$$

Each node has a specific result that can influence a calculation of a neighbors result. The result of a node $v \in V$ is defined as $r(v)$. Calculation of results can be done in different manner. For example it might be calculated by a linear function, where the variable represents a dependency to an adjacent result:

$$r(v_x) = r(v_y) - 4 \quad with\ v_y \in N_G(v_x)$$

Here $N_G(v_x)$ is the set of all neighbors of v_x. Also it might be possible that results are sets containing other nodes, for instance:

$$r(v_x) = \{v_a, v_b, v_c\}\ with\ v_a, v_b, v_c \in N_G(v_x)$$

C. Edge types and their usage

The system should be able to distinguish between optional and obligate connections between elements. Reason for this is to handle optional paths while deductive reasoning, for example by asking the user for his needs. An optional path is an edge with a specific successor that might be interesting for decision making, but the actual need of this successor is not sure until the decision making process is executed. The set with optional-edges is defined as followed:

$$E_{opt}(G) = \{e_x, \ \ldots, \ e_n\}$$

As opposed to optional paths, required paths are those that are included in a decision making process in any case. They represent the usual edge type and are contained in the set of required-edges:

$$E_{req}(G) = \{e_x, \ \ldots, \ e_n\}$$

The visualization in Fig. 3 illustrates the usage of optional (Fig. 3, a) and required (Fig. 3, b) paths as well as the usage of four other path categories. Required and optional paths can be visually distinguished by the end of each edge. An optional path ends with a non-filled connection. A required (usual) path is represented by a filled arrow head.

In addition to optional and required paths it is necessary to distinguish between more types of paths:

An ‘is-path’ is used for building inheritance relations, for example ‘MS Visio is Software’. It is visualized with a cross-filled circle (Fig. 3, c) on the predecessor element: ‘v_5 is v_6’. This type is always a required path, otherwise the system would not be able to decide, if an element is another or not.

Also paths that represent ‘used-for’-relations between elements are always required edges (Fig. 3, e). An element ‘is used for’ an activity or not; it is not consistent to say ‘perhaps an element is used for an activity’.

For modeling characteristics or attributes the model provides a further type of an edge, the ‘has’-path (Fig. 3, d). These paths serve as instruments to specify elements. For example every software ‘has’ a price and an installation type. Modeling this knowledge means three Items: software, price and installation type. The edges between these elements would be ‘software has price’ and ‘software has installation type’. Now every element, which ‘is’ software, has a price and an installation type, too.

The sixths path type is the ‘part-of’-edge (Fig. 3, f), which is used to build part-whole relations between elements. In opposite to a ‘has’-relation the ‘part-of’ element is not able to exist for its own, which means the whole-unit has to exist.

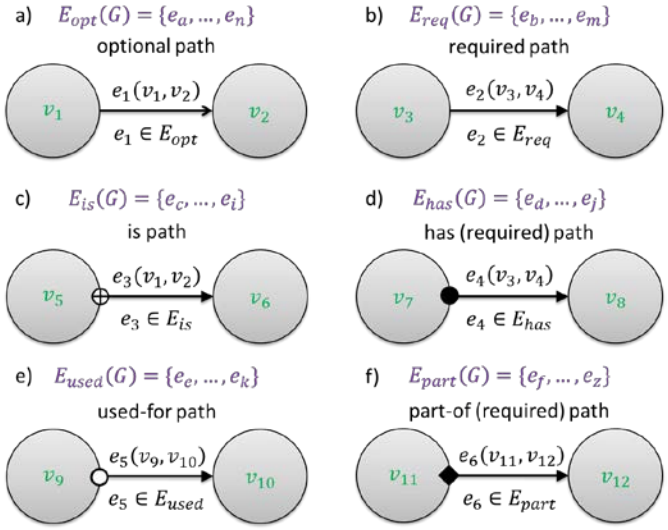

Fig. 3. Visualization of path types

Fig. 4 demonstrates an example with three Items. ‘Requirements documentation’ builds the successor, ‘Documentation of functional requirements’ and ‘Creating-

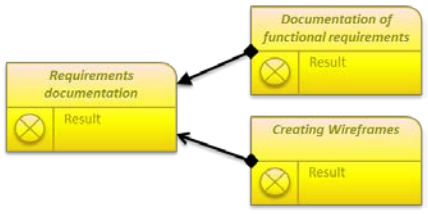

Fig. 4. Example of using optional and required paths

Wireframes’ are predecessors. Both predecessors might be interesting for a deductive reasoning process, for example ‘Find suitable software for requirements documentation!’. In addition to this both are connected with ‘part-of’ relations, which means the predecessors are part of ‘Requirements documentation’.

The difference between the relations is creating wireframes, which is not a mandatory functionality of requirements documentation, whereas the documentation of functional requirements represents one of the most important topics. Thus in case of decision making ‘Which software would be the most suitable for a specific software development project?’, users has to determine first, if they need the functionality of creating wireframes.

D. Constraints and Deductive Reasoning Element Pruning

Constraints are barriers, which help excluding paths from the set of all paths within the graph. Thus any association between elements can define preconditions or requirements. So the processing of an element is solely necessary if all of its constraints are complied. Goal is:

- more efficient processing through the graph by
- more intelligent operating on elements and thereby
- shorter times of results in deductive reasoning

If at least one constraint of an element is false, the element can be excluded from the entirety of all possible elements of deduction. Using the example of this publication, the integration of constraints leads faster to a list of matching project artifacts. Visualization of constraints is done by a dotted line as illustrated in Fig. 5.

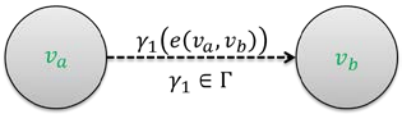

Fig. 5. Illustration and definition of constraints

The set of constraints is defined as:

$$\Gamma(G) = \{\gamma_1, \dots, \gamma_n\}$$

Considering an example of a simple association between Scrum and the number of regarded team members, Fig. 6 shows the functionality of constraints within the graph. A constraint γ is assigned to the edge $e(E1, E2)$ by $\gamma(e(E1, E2)): r(E2) > 5 \wedge r(E2) < 9$ that means the result of $E2$ has to be greater than five, but less than 9.

So γ can be interpreted as follows: Only if the result of node $E2$ (the number of team members) is between five and nine, element $E1$ (Scrum) is relevant for the deductive reasoning. If the result of $E2$ is less than five or higher than nine, $E1$ is irrelevant for the deductive reasoning, because at least one constraint is false.

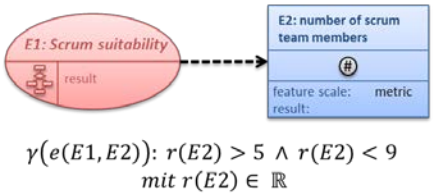

Fig. 6. Example of using constraints

By integration of constraints the entirety of nodes within a deductive reasoning process can be decreased. This process of reducing nodes can be named as Deductive Reasoning Element Pruning (DREP). Element Pruning is a well-known term in different Selection Algorithms as [28], [29] and [30] or in Clustering Methods for example [31] and [32]. In the domain of this model and publication DREP is an optimization measure during deduction, which is able to reduce the number of elements at the time T to be proceeded until $T + 1$. With DREP parallelism can result in jumping to nodes, which are possibly not reachable anymore, because they are on a pruned path. This case occurs with pruning of bridges (graph theory). Therefore it should be completely dispensed with parallelism or an event-driven system should be used, in case of integrating pruning.

The usage of DREP will be introduced in future work, where an algorithm for deductive reasoning will be shown that is suitable for the model of this paper.

E. Inference

As described above, the structure of the model should serve as KB with the ability of self-learning functionality and deductive reasoning. This is why the architecture of the model is a composition of ANN, which are used in artificial intelligence systems, and ontologies that have their usage in knowledge engineering. Furthermore it has already been expounded that each element within the model can output a specific result, which can be used by any other element. Considering it all together – artificial intelligence components, ontology-based representation of knowledge and the specific result of each element – lead to an architecture that is able to handle inductive reasoning and deductive reasoning in different ways.

By the derived structure of multi-layer feedforward ANN, the model can be used with well-known approaches of machine learning paradigms, as supervised learning, unsupervised learning or reinforcement learning. In addition to these inferring methods, it is possible to make simple decisions by searching the result of a specific element. Example of such a simple decision is 'Is Scrum suitable for a specific project team?'. Assumed that a Cell 'Scrum suitability' exists and that the result of this Cell is the actual value of how suitable scrum is, there would be no need to perform a complicated machine learning algorithm for answering the predefined question. The only need is to output the result. This can be done on two different ways. On the one hand the connected knowledge of the 'Scrum suitability' can be interpreted manually, by human reading and reasoning, or on the other hand automated, by a decision-making system. For doing it automatically, one of the next steps of this project is defining an algorithm that handles this behavior under consideration of constraints and DREP.

III. Example of Application

As outlined above, the model can be used for illustrating information and knowledge. One of the major goals of the research project behind this paper is to develop an automated decision system for identification suitable project tools and required artifacts, especially for large software development projects. Fig. 7 shows an extract of this use case and demonstrates the complexity of modeling a KB. The figure illustrates four out of five layers of the model: $L1$ Items (rectangular, rounded corner), $L2$ Cells (elliptical), $L3$ Features (rectangular) and $L4$ Questions (rhombic). The lines between the elements serve as connectors and represent the associations with their corresponding types.

There are two major levels of abstraction in the model: a descriptive and a deductive level. The first mentioned descriptive view is represented by Items as 'Software' and Features with a connection to Items as 'Price (amount)'. Descriptive elements are essential for learning and generating information. Items can also represent problems or goals, for example 'Classification of requirements', which is part of 'Requirements Engineering'. For solving this problem 'Jira' can be used. Jira is concrete 'Software' and so on.

In the example, 'Software' and their connected elements can be read as followed:

- *'Software' has 'Price' and 'Installation Type'*
- *'Operating System' is Software*
- *'Gliffy', 'Jira' or 'Astah' is concrete Software*
- *'Jira' can be used for 'Documentation of non-functional requirements',*
- *'Documentation of non-functional requirements' is a part of 'Requirements Engineering'.*

Features can be connected to Cells and Items. In cases of connection with Items, they will be executed with inductive reasoning, which means during learning phases. As opposed to Features that are connected to Items, Cell-Features need to be executed within a concrete deductive reasoning process, which is for example a decision process. Reason for this behavior is distinction between general and specific. The following two examples describe this approach:

Example 1: Sentence to learn: 'Jira is software.' The system already learned 'Software has a price'. Connected question to price: How much is the price? So the system has to ask: 'How much is the price [for Jira]?'

Example 2: Constraint to learn: 'Jira requires Windows 8.' The KB knows: 'Windows 8' is an 'Operating System'. So whenever the system has to make a decision through the Cell Jira, it has to ask: 'What is the Operating System?'. Obviously the system might ask directly 'Do you use Windows 8?', but in this case the operating system of the user would remain unknown and element pruning could not be made in the same efficiency as with the question above.

IV. Conclusion

A new method for modeling knowledge, information and data is introduced in this paper. It serves as architectural basis for developing expert systems by building knowledge bases in the field of knowledge engineering.

The abstract meta-model is constructed by five layers. They consist of descriptive and deductive elements and are derived by an artificial neural network. The model is illustrated both in a descriptive way and in a mathematical view by considering it

as graph. In addition to the general description the authors give an insight into a case of application using the example of large software development projects.

By integration of constraints and DREP the proceeding time of the graph can be decreased. Thus a deductive reasoning process is more efficient and the interaction with users can be reduced, when using the model as knowledge foundation. A further advantage of using the model as basis for additional research projects is the ability of extensibility. For instance it is very easy to assimilate different approaches for realizing deductive and inductive reasoning.

When considering deductive reasoning in one of the next steps, it is important to give solutions for the following problems: (1) Detecting the end of the deductive reasoning under regard to have an arbitrary entry-point and (2) handling conflicts in case of mutually exclusive Constraints.

In addition to use this model for knowledge engineering in the domain of large software development projects, it can also be used in different other domains. With the solution approach it could be possible to model knowledge of study advisers or career counseling, to build a basis for deciding what kind of occupation is the most suitable in dependency of personal characteristics. Further use cases are settled in 'Health and Medical' systems for building a foundation to detected symptoms and give suitable solutions. In case of building end-user systems the model can be used to develop a knowledge base for example of travel agents, fashion advisers or as product adviser.

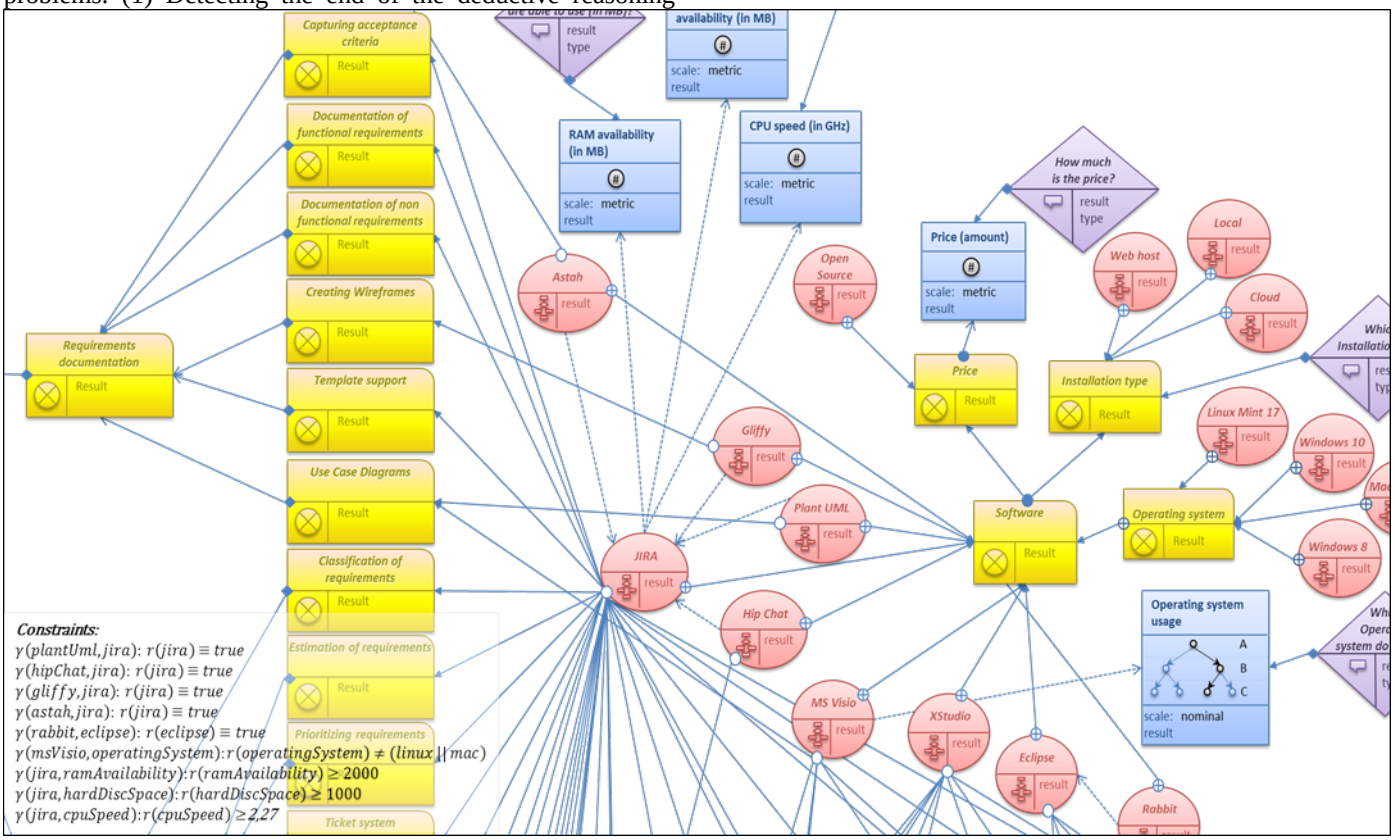

Fig. 7. Extract of using the model for knowledge engineering in large software development projects

REFERENCES

[1] F.F. Fuessl, J. Ciemala, "Variable Factors of Influence in Product Line Development", Computer Software and Applications Conference Workshops (COMPSACW), 21-25 Jul. 2014, Vasteras, pp. 390-395

[2] M.S. Mohktar, K. Lin, S.J. Redmond, J. Basilakis, N.H. Lovell, "Design of a decision support system using open source software for a home telehealth application." Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), 8-9 Nov. 2011, Bandung, pp. 390-395

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel; P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, "Scikit-learn: Machine learning in Python" The Journal of Machine Learning Research 12, 1 Feb. 2011, pp. 2825-2830

[4] M. H. Nguyen, F. De la Torre, "Optimal feature selection for support vector machines." Pattern recognition 43.3, Mar. 2010, pp. 584-591

[5] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Stajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, B. Zupan, "Orange: data mining toolbox in Python." The Journal of Machine Learning Research 14.1, Jan. 2013, pp. 2349-2353

[6] F. F. Fuessl, A. Triebel, D. Streitferdt, " Modeling Knowledge Bases for Automated Decision Making Systems – A Literature Review ", International Journal of Advanced Computer Science and Applications(IJACSA), 6(9), 2015

[7] K. He, C. Wang, Y. He, Y. Ma, P. Liang, "Theory of Ontology and Meta-Modeling and the Standard" Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization, Aug. 2009, pp. 85.

[8] J.D. Ullman, "Principles of database and knowledge-base systems" Computer Science Press, New York, 1988, p. 24

[9] R. Akerkar, P. Sajja, "Knowledge-based systems", Jones & Bartlett Publishers, 2010, p. 21

[10] G.S. Tuthill, S.T. Levym, "Knowledge-based systems: a manager's perspective", TAB Professional and Reference Books, 1991

[11] R. Donmoyer, M. Imber, J.J. Scheurich, "The knowledge base in educational administration: Multiple perspectives", State University of New York Press, Albany, 1995, pp. 17-18

[12] L.J. Heinrich, R. Riedl, D. Stelzer, "Informationsmanagement: Grundlagen, Aufgaben, Methoden", Walter de Gruyter GmbH & Co KG, 2014, p. 319

[13] B. Bouchon, R.R. Yager, "Uncertainty in Knowledge-Based Systems: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems", Springer Science & Business Media, 1987, p. 3

[14] Bayya Yegnanarayana, “Artificial neural networks” PHI Learning Pvt. Ltd., 2009., p. 29

[15] J.E. Dayhoff, J.M. DeLeo, "Artificial neural networks", Cancer 91.8, Apr. 2001, pp. 1615-1635

[16] V.S. Desai, J.N. Crook, G.A. Overstreet, "A comparison of neural networks and linear scoring models in the credit union environment", European Journal of Operational Research 95.1, Nov. 1996, pp. 24-37

[17] P.D. McNicholas, Y. Zhao. "Association rules: an overview", Information Science Reference (imprint of IGI Global), 2009, pp. 1-10

[18] H. Zhu, "Software Design Methodology: From Principles to Architectural Styles", Butterworth-Heinemann (imprint of Elsevier), 2005, pp. 156-157

[19] P. Norvig, "Paradigms of artificial intelligence programming: case studies in Common LISP", Morgan Kaufmann Publishers Inc., 1992

[20] E. Bonabeau, M. Dorigo, G. Theraulaz. "Swarm intelligence, From Natural to Artificial Systems", Oxford University Press, 1999

[21] R.A. Brooks, "Achieving Artificial Intelligence through Building Robots", Massachusetts Institute Of Technology, No. AI-M-899, May 1986.

[22] C. Landauer, "Data, information, knowledge, understanding: computing up the meaning hierarchy", Systems, Man, and Cybernetics, San Diego (CA), 11-14 Oct.1998, pp. 225-2260 vol.3

[23] M.C. Golumbic, "Algorithmic graph theory and perfect graphs", Vol. 57. Elsevier B.V., second edition 2004

[24] M. Foreman, A. Kanamori, "Handbook of set theory", Springer Science & Business Media B.V, 2010.

[25] P.C. Biswal, "Discrete mathematics and graph theory", Fourth Edition, PHI Learning Pvt. Ltd., Delhi, 2015, P. 101

[26] C. Rousseau, Y. Saint-Aubin, "Mathematics and technology", Springer Science & Business Media, LLC, 2008, p. 269

[27] R. Garnier, J. Taylor, "Discrete mathematics for new technology", second edition, IOP Publishing Ltd., 2002, p. 592

[28] D. Dor, U. Zwick, "Selecting the median", SIAM Journal on Computing 28.5, 1999, pp. 1722-1758.

[29] D. Dor, "Selection algorithms", Diss., Tel-Aviv University, Sep. 1995

[30] T. Anand, P. Gupta. "A selection algorithm for X+ Y on mesh", Parallel processing letters 08.03, Sep. 1998, pp. 363-370.

[31] G. Bisson, C. Nédellec, D. Canamero, "Designing Clustering Methods for Ontology Building-The Mo'K Workbench", ECAI workshop on ontology learning. Vol. 31, 2000

[32] L.A.F. Fernandes, A.C.B. García, "Association rule visualization and pruning through response-style data organization and clustering", Advances in Artificial Intelligence–IBERAMIA 2012, Lecture Notes in Computer Science Volume 7637, Springer Berlin Heidelberg, 2012, pp. 71-80