VODRE: Visualisation of Drools Rules Execution

Maxim Lapaev, Maxim Kolchin
Saint Petersburg National Research University of Information Technologies,
Mechanics and Optics
Saint-Petersburg, Russia
m.lapaev@telemetria.ru, kolchinmax@niuitmo.ru

***Abstract*—Knowledge-based Systems and Expert Systems, in particular, are expensive to build and difficult to validate and debug because of their complexity and dynamism. Therefore, it is not easy for knowledge engineer and domain expert to identify the gaps and mistakes in knowledge base. Unit testing is unable to cover validation process at all stages, in many cases manual thorough review of decision process is needed. In this paper we spot main approaches to validation and verification issue and describe a component that helps to debug a knowledge base by visualising execution of rules that derive a particular result. This component is developed for Knowledge-based Systems built on Drools Platform[1] and we demonstrate application of this component in a knowledge-based engineering system for structural optical design.**

I. Introduction

The standard of engineering when a decision could be made only with a help of reference book, pencil and calculations is far behind. Today's industry requires efficient on-the-fly solutions and decisions. Sometimes not only effective decision is important but the time to produce the solution, as well. Nowadays the majority of knowledge-based systems are complex artificial products which are able to take the place of a real expert in a particular knowledge domain. Expert systems (ES) became an indispensable part of industry and reached almost all fields of modern life: finances, cartography, military, industry, medicine, science and so forth [1], [2], [3], [4]. Not only do expert systems simplify the process, they make it more reliable and qualitative. However, ESs are aimed not at replacing an expert but at becoming an assistant. Need for urgent solutions requiring time-consuming calculations and analysis and lack of highly-qualified domain experts make ES development to become a perspective trend.

Although quality and efficiency of ES has improved dramatically over last decades, knowledge-based system engineers still face difficulties. On the one hand, ES must be reliable and produce right and optimal solutions quickly, on the other hand, test and debug toolkit still leaves much to be desired. A wide range of testing frameworks is available for developers of conventional system, but only a few of them are suitable for ES testing, validation and debugging. For example, JUnit[2] is one of the most distributed and recognised testing frameworks among Java programmers, but as to consider knowledge-based engineering, its functionality narrows to test cases related with code base only and leaves knowledge base unattended. Any expert system is at least a combination of user input-output interface, knowledge base, rule base and decision system (code base). It is obvious that regular test frameworks are suitable for code base testing and debugging only. The same disadvantage is peculiar to all testing frameworks aimed at testing code-base and algorithms. Therefore, researches on evaluation in knowledge-based engineering are made over last years.

Correctness of an expert system assumes that both the right system was built and the system was built right [5]. These stages are referred to as validation and verification, respectively. Using a system not thoroughly tested (unreliable system) may cause a disastrous result, especially when a failure is concerned not with material costs, but with security or human life. Proper validation of an expert system must ensure that both code base and knowledge base produce correct solution. Domain rules and reasoning processes are to be tested, as well. Prototyping is not suitable for testing large-scaled systems as it is impossible to anticipate all possible findings and decisions. Functionality of prototype is limited to key aspects. Fine testing and debugging strategy includes visualisation of reasoning. A coloured progress bar that is popular in unit testing is not enough for inspection of test cases and reasoning process as it is intended to signalise whether the test failed or succeeded without deep investigation of intermediate stages. But not-easy-to-find bugs, which are hard to analyse without depicted work-flow, are often encountered in all subsystems of ES as well as the rule base.

II. Problem Statement

In the context of this paper we focus on the validation and visualisation process of knowledge-based engineering system for structural optical design[6]. We use the Drools platform developed by Red Hat company as a rule engine. The idea is to produce a sufficient way to depict the reasoning process in a manner which is suitable for testing by a person slightly related with development process and may become a part of explanation subsystem in future.

Many of debug techniques do not support a possibility of sufficient manual review. However, thorough reasoning and decision testing process is usually related with manual review by an expert in subject area. This often includes a careful analysis of large blocks of hard-to-display-on-the-screen data. Scrolling distracts the attention of the expert from investigation. Furthermore, a domain expert is not necessarily supposed to be familiar with the system and specialised software, the same situation is with our OSYST optical design system. Thus, visualisation charts must contain as little information and blocks as possible but enough for inspection. Demonstrative and readable charts and diagrams make it easier to analyse work-flow and solution derivation and speed up the process

[1]Drools Platform, URL: http://drools.org
[2]JUnit, URL:http://junit.org

ISSN 2305-7254

of testing. Decision-support system visualisation must be a flexible, powerful tool. Moreover, a well-designed visualisation system for validation and debugging may become a part of explanation subsystem as well. Thus, a thorough overview of related works must be conducted and a not-time-consuming-to-cope-with visualisation mechanism must be produced.

The rest of the paper is organised as follows: Section 3 briefly overviews widely-used approaches in expert system development, validation and verification and other works related to expert system debug and test tools. Section 4 focuses on description of OSYST system and its components, its implementation and the improvements made for the convenience of validation process and system's feasibility in general and validation and visualisation techniques, in particular, as well as experimental evaluation of introduced component.

III.Related work

A. Validation and Verification Overview

The topic is not new to knowledge-based engineering. ES testing has been an object of attention as long as decision-support systems exist. As soon as first expert systems were developed, validation, testing and debugging became a discussed research subject. Studies on unit testing can be found in [7]. But all of them take into consideration only validness of final solution and sidestep deep analysis of work-flow and rule implementation by a domain expert.

Development of a knowledge-based system is a continuous process, including many stages. Thus, validation and verification is to be applied not when the system is completed, but throughout the whole expert system development cycle. Vanthiene, Mues and Aerts[8] focus on validation from the very beginning of development cycle, particularly at modelling stage. They sum up investigation, stating that a lot of anomaly types can easily be prevented or detected, and, consequently also corrected in an early stage of development. But the approach is not acceptable or not of current importance, when validation was omitted at modelling stage and applied later on further stages of development.

Preece[9] provides a critical assessment of current state of the practice in knowledge-based system validation and verification, including an overview of available evidence of the efficiency of various validation and verification methods in real-world knowledge base development projects. The author offers recommendations for the use of validation and verification methods and presents overview of other validation technique studies (the Minnesota study, the SRI study, the Concordia study, the SAIC study, the Savoie study) and concludes that the collective knowledge on efficient validation and visualisation techniques is very limited and is not suitable for every knowledge-based system.

Further studies concentrate on methods and tools for decision-process visualisation at the stage of empirical testing [5], [10].

Baumeister[5] suggests DDTree (derivation/dialog tree) - a powerful tool for ES validation and visualisation of reasoning. DDTrees are able to present both final and intermediate solutions in an easy-to-read intuitive way. Successful test cases are coloured in green, erroneous cases are red. New test cases are shadowed until test is launched. Tree-like diagram may be printed and inspected by an expert in subject area. Branched structure of chart allows analysing the reason of failure based on incoming findings. DDTree is a part of KnowWe[11] platform which is used for building knowledge based systems. But derivation trees are used only for static analysis. They do not have a timeline to track order of execution.

Another visualisation method is proposed by Tavana[10]. He offers PNs (Petri Nets) for dynamic system representation and rule derivation. It overcomes the drawback of derivation trees and allows analysing system in dynamics.

B. Main Approaches to Validation and Verification

Nowadays knowledge-base engineers mark out a number of approaches to knowledge-base system validation and verification. B. J. Wielinga, A. Th. Schreiber and J. A. Breuker[12] introduce a so-called KADS approach. In KADS (Knowledge Analysis and Documentation System), the development cycle of a knowledge-based system (KBS) is treated as a modelling process. A KBS is not considered to be a box filled with expert knowledge, but an operational model which displays a desirable behaviour in terms of domain area. Authors give an overview of activities undertaken by engineers to produce an expert system and illustrate it with examples in the domain of troubleshooting audio equipment.

KADS was further developed into CommonKADS: a comprehensive methodology for KBS development. CommonKADS postulates are:

1) knowledge engineering is not some kind of 'mining from the expert's head', but consists of constructing different aspect models of human knowledge;
2) the knowledge-level principle: in knowledge modelling, first concentrate on the conceptual structure of knowledge, and leave the programming details for later;
3) knowledge has a stable internal structure analysable by distinguishing specific knowledge types and roles.

A paper by Prat, Akoka and Comyn-Wattiau[13] is concerned with MDA (Model-driven architecture) approach to knowledge engineering centred on the CommonKADS knowledge model. They start by grouping elements of the CommonKADS knowledge models into a so-called "inference groups", propose and demonstrate an algorithm that defines these inference groups automatically and propose a comprehensive CommonKADS knowledge metamodel.

Another approach to knowledge-based systems' design and development is described by Cairo and Guardati[14]. They introduce a KAMET (Knowledge-acquisition methodology) II approach, which states that knowledge acquisition (KA) process is not "mining from the expert's head" and producing rules for building KBS as it used to be 20 years ago when modern engineering toolkit did not exist and knowledge acquisition was confused with knowledge elicitation activity. Nowadays knowledge acquisition process must involve both dynamic modelling and knowledge generation activities. The paper presents attempts to build a new KA approach, that includes all of these ideas. KAMET II brings a number of new to KBS development processes and tends to transform tacit knowledge into explicit knowledge.

However, all stated above approaches are largely academic and theoretical and may cause challenges, especially for those who have not developed an expert system before. A step-by-step approach described by Freiling, Alexande, Messick, Rehfuss and Shulman[15] helps to overcome these challenges at the beginning of design stage and through the development process. The authors make two fundamental assumptions:

1) knowledge is more valuable than inference strategies;
2) a knowledge engineering project must provide adequate documentation of its progress; at any stage in the process knowledge engineers must be able to show the results of their work.

To reduce the risk of inconsistency of a developed system authors recommend to follow 6 steps to an expert system prototype (Fig. 1).

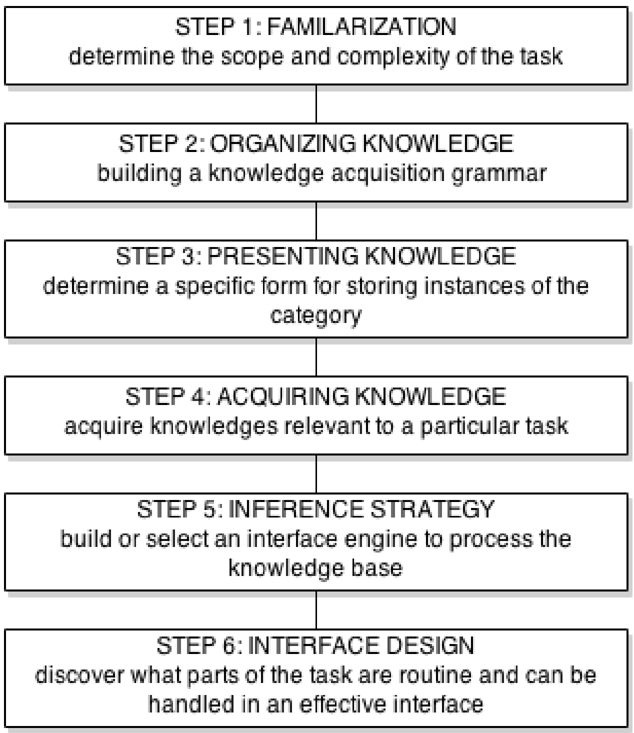

STEP 1: FAMILARIZATION
determine the scope and complexity of the task
STEP 2: ORGANIZING KNOWLEDGE
building a knowledge acquisition grammar
STEP 3: PRESENTING KNOWLEDGE
determine a specific form for storing instances of the category
STEP 4: ACQUIRING KNOWLEDGE
acquire knowledges relevant to a particular task
STEP 5: INFERENCE STRATEGY
build or select an interface engine to process the knowledge base
STEP 6: INTERFACE DESIGN
discover what parts of the task are routine and can be handled in an effective interface

Fig. 1. Six steps to an expert system prototype

One of the keystones stated by authors is choosing the right tool. A tool is concerned not only with design process, but with all stages of development lifecycle, including validation and verification, thus, they mark out four requirements for a knowledge engineering methodology:

1) the methodology must be simple;
2) the methodology must be gradual;
3) the methodology must aim at getting the knowledge;
4) the methodology must provide measurable milestones.

However, all steps implemented do not guarantee that the system is consistent. Proper testing is to be done to ensure that the system is built right and the right system is built. They suggest that engineers use testing tools as well as visualisation tools (CHECKA and PIKA, respectively, in their case). Authors conclude that the collection of tools and components supports a step-by-step approach to knowledge engineering, providing a way to keep making progress on the problem at hand and using all possible tools is a good practice to fulfil a proper development and testing cycle.

IV. The OSYST system implementation and improvement

A. *Subject Area Overview*

The investigations implemented in this work are concentrated on knowledge-based optical design system OSYST[3]. Main issues of optical systems' (OS) composition are classification of elements in OS and analysis of their applicability in cases concerned. All elements, by their function, are divided into four groups: basic (B), corrective (C), high-aperture (T) and wide-angular (Y) elements. The derivation of element sequence to result the specific optical characteristic is referred to as structural synthesis. Structural synthesis is the first stage of automated optical system design.

Since a large number of similar and differing structural schemas often meets the same technical requirements, a process of structural synthesis does not have any determined solution algorithm. An expert in optics relies on her/his experience and knowledge in subject area. The more experienced domain expert is, the more optimal schema is produced.

A great experience and knowledge in domain area, described by Rusinov and Livshits allows formalising the process and, consequently, building up a basis for automated structural synthesis and OSYST system development. The optical system calculation brings to ability of engineer to arrange elements of optical scheme so that an optimal sequence is produced in order to be able to drive the system to better image quality without exceeding established geometrical constraints.

Optical system synthesis starts with analysis of technical requirements which allow to define applicability criteria. According to requirements an optical schematic diagram is produced (Fig. 2).

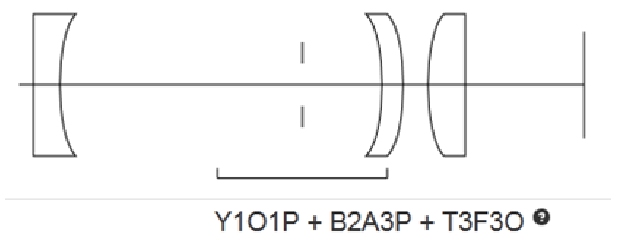


Fig. 2. An example of optical schematic diagram produced by OSYST with its structural formula

Starting point selection is the determinative stage of design as final outcome of synthesis becomes less time-consuming as a result of better convergence of optical system optimisation process. Starting point misselection brings to running over a great number of possible combinations of free parameters of

[3]OSYST repository, URL: https://github.com/ailabitmo/OSYST

the system causing a tiny possibility of optimum solution to be ever found. In practice, it results in using trial method, which is time-consuming for both expert and computer. In most cases such unproductive calculations are unsuccessful. Thus, a failure in optical system synthesis by an expert system causes waste of time and overcosts. A rapt attention must be paid to expert system reliability.

B. Rule Engine Choice

Drools production rule system, using an enhanced implementation of the Rete algorithm, was chosen as a rule engine. Drools supports the JSR-94 standard for its business rule engine and enterprise framework for the construction, maintenance, and enforcement of business policies in an organisation, application, or service. Drools is considered to be one of the most developed and supported platforms. It has a number of knowledge representation ways. The main and the most wide-spread of them is a decision representation language (DRL), which is used in production system development. DRL is tightly integrated with Java language. Furthermore, the platform enables to write DSL constructions to make rules more vivid and easy-to-understand for domain experts. A detailed description of Drools is redundant as the platform is well-documented, thus, only features significant for the OSYST system are stated below.

Drools has an advanced decision engine that supports both forward and backward chaining that allows wider range of abilities at design stage (a forward chaining is used in our expert system as it is acceptable when an object is to be synthesised based on the facts). The last but not the least, Drools platform has database design toolkit referred to as Guvnor, which is not included in most of other platforms. Guvnor combines all essential tools for knowledge-based engineering, such as rule editor, rule storage, test tools, decision tables and so forth. Furthermore, the described platform has a user-friendly web-interface. Drools is cross-platform, its installation process is not time-consuming, knowledge bases can be easily exported and imported, which results in better portability. All facts considered, Drools platform has many advantages over the other platforms and meets our needs.

C. System architecture

The OSYST expert system consists of four fundamental components: a knowledge base, implemented on Drools platform, which is described before in this paper; a server component developed using Play! Framework[4]; a client component, represented by a browser user interface and a database to store user accounts and saved work (Fig. 3). Logging for visualisation process takes place on server side, logging process is described further in details. Knowledge- and rule bases are developed, using built-in tools of Guvnor. The platform offers a fine versioning mechanism which is very desirable, especially during development and testing stages. Rules set the correspondence between technical and general characteristics. The example of such rule in DRL language is represented in Alg. 1. It sets the correspondence between ”technical” focal length and ”generalised” or ”formalised” F.

[4]Play! Framework, URL:http://www.playframework.com

Fig. 3. The OSYST system architecture. Logging for VODRE tool is indicated by curvy line.

Algorithm 1 An example of rule in DRL language

```
rule F1
when
    $class : Classification()$
    $Requirements(focalLength > 50 < 100)$
then
    $\$class.setF(1);$
end
```

Drools Guvnor publishes a web-service that is used as a RESTful interface for connection to knowledge base. As it was mentioned before, Drools platform is tightly integrated with Java language (in fact it is developed using Java language), therefore, Java platform for server application was chosen. Server component is intended to accept data inserted by user, process it according to knowledge base stored in Guvnor and give the result back to user by client’s request using Ajax, which results in quicker response (reaction) of the system.

The choice of server framework is concerned with time consumption. Play framework allows developing software quickly, following the Convention over Configuration paradigm, that tends to minimise the number of decisions made by developer by simplifying configuration process, granting finished modules and so forth. Furthermore, it supports MVC (Model-View-Control) pattern that allows developing a server and a client component in the same development cycle.

Client pages are developed, using standard and widely-spread tools and languages (HTML, CSS, JavaScript). All structural synthesis drawing functions, including visualisation, are implemented with a help of KineticJS Canvas JavaScript[5] framework that includes all basic primitives and canvas functions. User accounts are stored in MySQL database.

D. Validation and Verification

A variety of tools were used for expert system validation and verification: unit testing, integrational testing, systems testing and database testing. Unit testing was applied to all components of the system in client application as well as in server (Jasmine[6] and JUnit. A manual alpha-testing was conducted by potential users of the system to check application consistence. However, till now all manual tests were conducted by comparing schema synthesised based on inserted parameters with expected schema visually. A component for rule execution visualisation was developed to simplify the process of manual testing. Data about facts inserted and rules executed is collected by AgendaEventListener and Working MemoryEventListener from Knowledge API[7]. Collected logs are received by client, then analysed and represented in debug (before this investigation) and in visualisation (now) windows, which lets an alpha-tester have both schema and rule execution

[5]KineticJS, URL:http://kineticjs.com

[6]Jasmine Framework, URL:http://jasmine.github.io

[7]JBoss Knowledge API, URL:http://docs.jboss.org/jbpm/v5.1/javadocs

order for analysis in front of his/her eyes (Fig. 4). Visualisation process is described in details in next subsection.

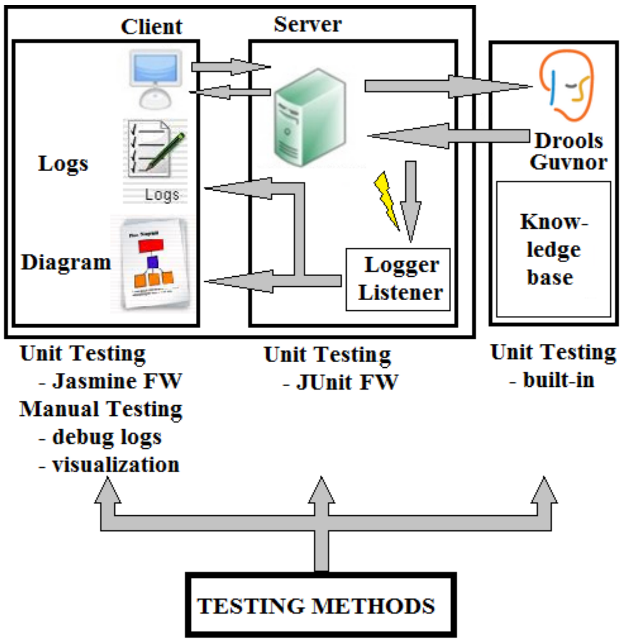


Fig. 4. Testing tools after visualisation component introduction

E. *Visualisation Mechanism and Diagram Elements*

Visualisation process requires data about facts provided and rules being triggered in working memory as well as the order of rule invocation. For that purpose a logging mechanism for connection between rule engine and server was developed based on standard JBoss knowledge tools. After data insertion at client side a request is sent to server and synthesis process listened by visualisation (rule invocation) logger starts. A diagram of rule invocation logger is displayed in Fig. 5. RuleRunner class handles a stateless session with knowledge base, which is listened by WorkingMemoryEventListener. Thus, listening process is parallel to optical schema generation, and logs received include all steps of synthesis. Listener class implements methods for entity insertion, update and retraction (*objectInserted()*, *objectUpdated()* and *objectRetracted()* respectively) which handle information on events in working memory. As soon as knowledge session is finished, synthesised schema formulae are returned to client together with synthesis logs.

All logs received are parsed on client side and visualised. Log parsing is accomplished by *parseRules()* function written in JavaScript synthesis module. Generated schemas are presented to user and two windows (debug and visualisation) are displayed. Windows are used for debug log output and visualisation chart drawn on canvas, respectively.

Two types of identities are used in reasoning process of the synthesis: these are rules and facts. Both rules and facts can be added to working memory, removed from it or changed (updated). A fact (or a set of facts) make rules to be fired. Based on the above, a set of visualisation diagram elements was worked out. All diagram elements are represented in

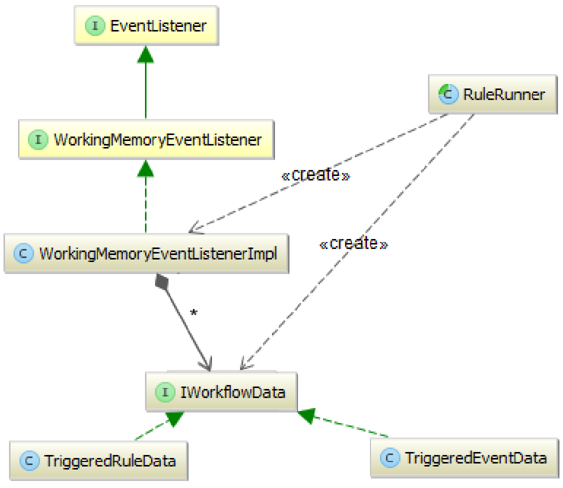


Fig. 5. A class diagram of rule invocation logger (only logger package classes are displayed, others are omitted for the reason of diagram complexity)

Fig. 6. Rules are represented by parallelograms (a) with a name of the rule inside; facts are represented by rectangles (b), containing facts' names (values); operations on facts are displayed by horizontal arrows directed to left side (c), right side (d) and both directions (e) for adding, removing or changing a rule in working memory, respectively; vertical bent arrows (g) are used between rules and facts, operated by the rule. Elements are aligned along a timeline (f), represented by a vertical line, from top to bottom.

Thus, a validation process after introduction of visualisation component should look as follows:

1) an alpha-tester provides system with input data;
2) as soon as input is finished, a request is sent to server;
3) schemas are synthesised on the server, logs of rules' invocation are collected in parallel;
4) optical formulae of synthesised schemas and logs are returned from the server;
5) schemas are drawn in web-browser client of OSYST;
6) synthesis process visualisation diagram is built based on rule invocation logs;
7) rules and rule execution order is analysed with a help of visualisation chart (a more time-consuming manual comparation of synthesised schema with expected schema, which had to be calculated in advance, was required before).

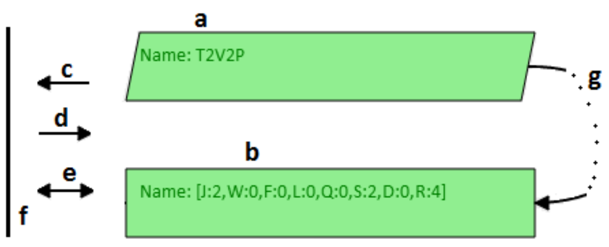


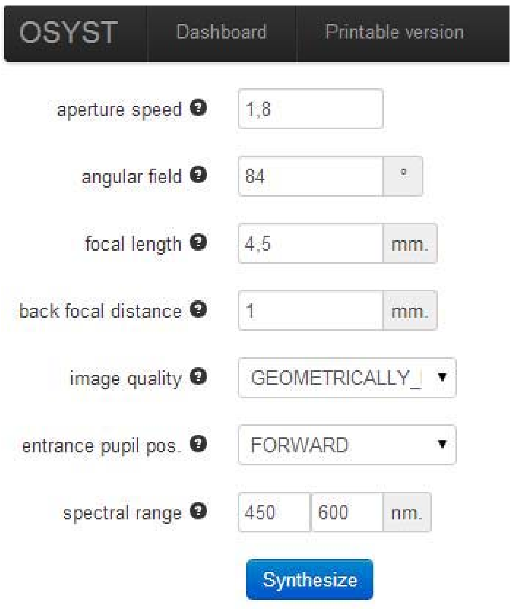
Fig. 6. Elements of workflow diagram

F. Experimental Evaluation

For the sake of reliability and robustness of visualisation component manual analyses of schemas generated and rules triggered was conducted. Manual review showed that the rule implementation order displayed on visualisation chart corresponds the order of rules to build schemas, and the schemas, in turn, appear to be valid for the technical requirements and data provided. But after a thorough testing of visualisation mechanism charts are to replace manual inspection of schemas with a simple check of rule implementation order based on the visualisation.

During experimental evaluation process the system was provided with sets of input optical parameters, for which possible synthesis results were already known, to make sure that expected schemas are generated and visualisation component displays charts with correct rule invocation order to get these structural schemas. The complete process of evaluation for one of the input sets is described next.

One of the synthesis processes is presented here as an example. An input form of photographic lens subsystem (Fig. 7) was filled with an initial set of valid test data (Table I) and a request to server was sent by ”Synthesis” button. Based on facts inserted, a total of 21 rules were triggered (12 from *generation* package, 6 from *corrective* package and 1 from *basic*, *fast* and *wideangular* packages). A set of 12 structural schemas was produced (Fig. 8).



aperture speed	1,8
angular field	84 °
focal length	4,5 mm.
back focal distance	1 mm.
image quality	GEOMETRICALLY_
entrance pupil pos.	FORWARD
spectral range	450 600 nm.

Fig. 7. Experimental evaluation. Test initial data set.

TABLE I. INITIAL DATA SET FOR EVALUATION

Aperture speed:	**1.8**
Angular field:	84°
Focal length:	**4.5mm**
Image quality:	**Geometrically limited**
Back focal distance:	**1mm**
Entrance pupil position:	**Forward**
Spectral range:	**450..600nm**

The schemas produced appear to be valid for the technical requirements provided and rule invocation order, seen in visualisation chart, in turn, is correct (Fig. 9). Other experiments with structural schema synthesis and visualisation showed satisfactory results and feasibility of the component, as well.

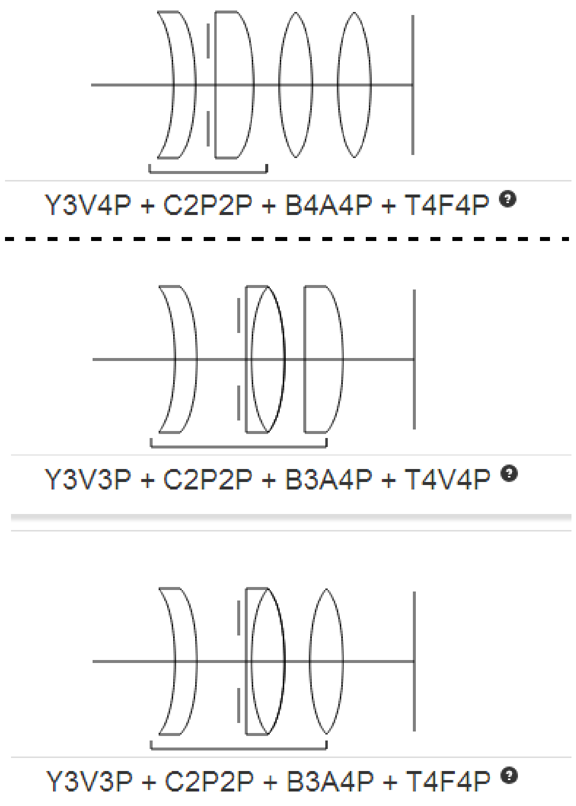

Y3V4P + C2P2P + B4A4P + T4F4P
Y3V3P + C2P2P + B3A4P + T4V4P
Y3V3P + C2P2P + B3A4P + T4F4P

Fig. 8. Produces schemas. Not all of 12 schemas are presented.

G. Examples of Visualisation

For testing and clearness purposes a synthesis result for another sets of input values was visualised. The following example displays a synthesis process for fake random input not related with real-world application in order to simplify produced diagram so that the visualisation chart may be presented full-sized and uncut. An initial set of input data presented in Table II was chosen as it satisfies simplicity requirements. Based on test initial data a diagram was built (Fig. 10). As one may notice, first rules are displayed ordered by invocation time. Then, facts are displayed in ascending order by insertion time. Finally, a relationship between rules and facts is drawn (arcs with arrows).

TABLE II. INITIAL DATA SET FOR SIMPLIFIED DIAGRAM EXAMPLE

Aperture speed:	**1**
Angular field:	2°
Focal length:	**1mm**
Image quality:	**Geometrically limited**
Back focal distance:	**1mm**
Entrance pupil position:	**Forward**
Spectral range:	**1..2nm**

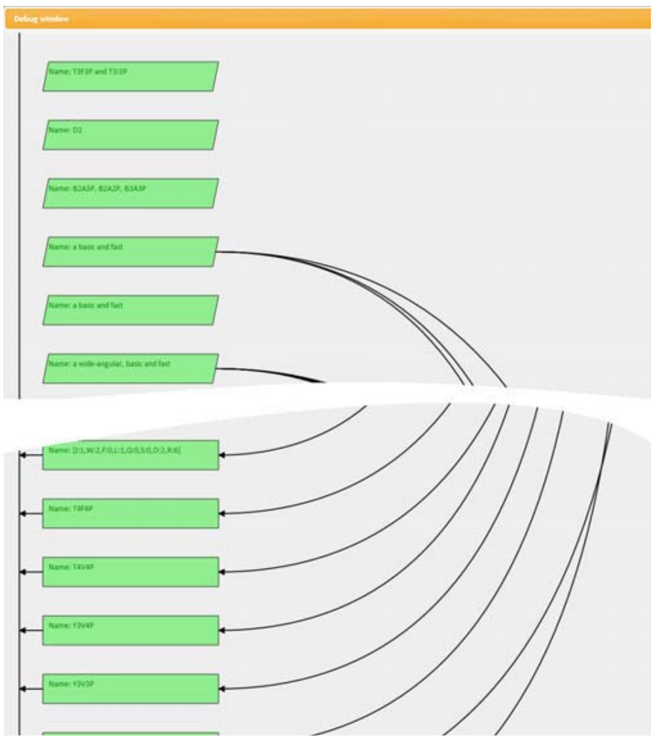


Fig. 9. Experimental evaluation. Visualisation of synthesis based on real-world parameters (cut for size reasons)

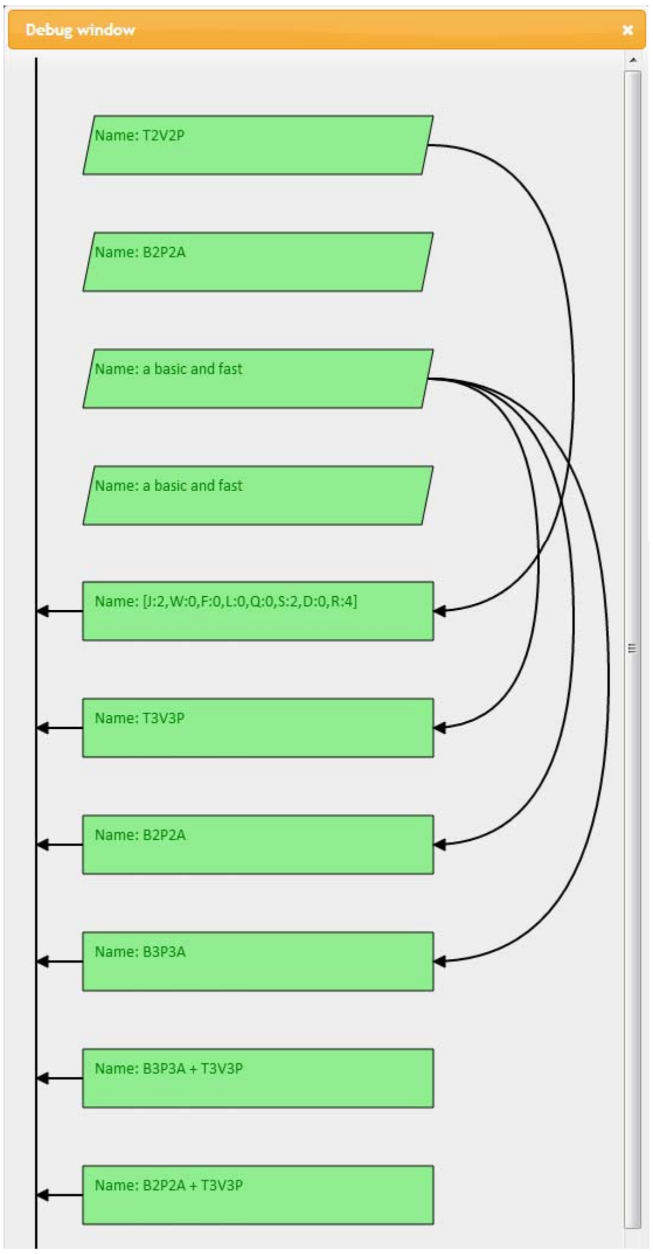


Fig. 10. Example 1: visualisation of simplified synthesis based on not real-world data

V. Conclusion & Future work

Expert systems are complex artifacts that are difficult to develop and test. This paper presented technical aspects of OSYST, an environment for automated structural synthesis of optical systems (photo-objectives), and the process of its validation and verification, and reasoning visualisation in particular. The analysis of the system has shown a drawback of validation process, which was overcome by introducing a visualisation component for tracking working memory processes, mainly rule invocation and operations on facts.

In this study, we showed that a new visualisation component gives an expert another opportunity of inspection of knowledge engine workflow. The more ways to assess rule executions alpha-tester has, the more thorough and efficient validation process becomes. Concentrating on rule firing order rather than comparing synthesised schema with expected schema simplifies validation and verification in general.

Despite the fact that the visualisation component designed is applicable for validation purposes, further revision is desirable. A scaling mechanism is needed to make it easier to review large diagrams, and more information on rules fired would be an advantage, too. More research and development is necessary to expand functionality of the component to make it a powerful debugging tool.

Future work should be focused on improvement of expert system, in general, and improvement and introduction of new components and tools, in particular.

References

[1] S.H. Liao, “Expert system methodologies and applicationsa decade review from 1995 to 2004.“, *Expert systems with applications*, vol.28(1), 2005, pp. 93-103

[2] M. Huettig, G. Busher, T. Menzel, W. Scheppach, F. Puppe, H.P. Buscher, “A Diagnostic Expert System for Structured Reports, Quality Assessment, and Training of Residents in Sonograph“, *Medizinische Klinik*, vol.99, 2004, pp. 117-122

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

[4] R.K. Lindsay, B.G. Buchanan, E.A. Feigenbaum, J. Lederbeg, “DENDRAL: a case study of the first expert system for scientific hypothesis formation“, *Artificial Intelligence*, vol.61, 1993, pp. 209-261

[5] J. Baumeister, “Advanced Empirical Testing“, *Knowledge-Based Systems*,

vol.24(1), 2011, pp. 83-94

[6] D. Mouromtsev, I. Livshits, M. Kolchin, “Knowledge based engineering system for structural optical design“, *Frontiers in Artificial Intelligence and Applications*, vol.246, 2012, pp. 254-272

[7] R. Knauf, A. J. Gonzalez, K. P. Jantke, “Validating rule-based systems: a complete methodology“, *in Proc. IEEE SMC’99 Conf.*, 1999, pp. 744-749

[8] J. Vanthienen, C. Mues, A. Aerts, “An illustration of verificat on and validation in the modelling phase of KBS development“, *Data and Knowledge Engineering*, vol.27, 1998, pp. 337-352

[9] A. Preece, Evaluating Verification and Validation Methods in Knowledge Engineering, in R Roy (ed), *Micro-Level Knowledge Management*, Morgan-Kaufman, 2001, pp. 123-145

[10] M. Tavana, “Knowledge-Based Expert System Development and Validation with Petri Nets“, *Journal of Information & Knowledge Management*, vol.7(1), 2008, pp. 3746

[11] J. Baumeister, J. Reutelshoefer, F. Puppe, “KnowWe: a Semantic Wiki for knowledge engineering“, *Applied Intelligence*, vol.35(3), 2011, pp. 323-344

[12] B. J. Wielinga, A. Th. Schreiber, J. A. Breuker, “KADS: a modelling approach to knowledge engineering“, *Knowledge Acquisition - Special issue on the KADS approach to knowledge engineering*, vol.4, 1992, pp. 5-53

[13] N. Prat, J. Akoka, I. Comyn-Wattiau, “An MDA approach to knowledge engineering“, *Expert Systems with Applications: An International Journal*, vol.39(12), 2012, pp. 10420-10437

[14] O. Cair, S. Guardati, “The KAMET II methodology: Knowledge acquisition, knowledge modeling and knowledge generation“, *Expert Systems with Applications: An International Journal*, vol.39(9), 2012, pp. 8108-8114

[15] M. Freiling, J. Alexander, S. Messick, S. Rehfuss, S. Shulman, “Starting a Knowledge Engineering Project: A Step-by-Step Approach“, *AI Magazine*, vol.6(3), 1985