VALIDATION OF KNOWLEDGE-BASED SYSTEMS THROUGH COMMONKADS

Feras Batarseh

Intelligent Systems Lab
School of Electrical Engineering and Computer Science
University of Central Florida (UCF)
4000 Central Florida Blvd.
Orlando, FL 32816

Avelino J. Gonzalez

Intelligent Systems Lab
School of Electrical Engineering and Computer Science
University of Central Florida (UCF)
4000 Central Florida Blvd.
Orlando, FL 32816

Rainer Knauf

Ilmenau University of Technology
Department of Computer Science and Automation
P.O.Box 100565
Ilmenau, Germany 98684

ABSTRACT

This paper defines a method that can be used for validating knowledge-based systems (KBS) throughout their entire lifecycle. Method's name is MAVERICK. It stands for Method for Automated Validation Embedded into the Reusable and Incremental CommonKADS. The lack of suitable, rigorous and general validation methods has become a serious obstacle to user acceptance of KBS for critical applications. In spite of recent significant advances in validation of KBS, it still remains an open problem. The ideas presented in this paper are based on the concept that validation should be performed in a structured and guided manner, integrated within a knowledge-based systems' lifecycle development method.. We define an incremental validation method for KBS based on extracting test cases from CommonKADS. Furthermore, we introduce our method for reducing the number of test cases and thus reducing validation's effort and cost.

Index Terms **-** Validation, CommonKADS, Knowledge-based systems, Test case.

1. INTRODUCTION

This paper describes a method that integrates validation within a life-cycle development method. The most comprehensive definition of validation was recently introduced by Gonzalez et al. [1] in the context of knowledge-based systems: "Validation is the process of ensuring that the output of the intelligent system is equivalent to those of human experts when given the same input." We adopted this definition because it is general and because it states that validation is comparing the system to the real world. Different methods for the validation of knowledge bases have been developed such as BKB [2], VKB [3], KVAT [4], SEEK and SEEK2 [5]. Furthermore, methods for system validation were developed, such as Bi-directional many-sided explanation typed multi-step validation, VESA [3], CORUS [6], CASE VALIDATOR [7], KJ3 [8], VVR [4] and quasi-exhaustive set validation [9]. Additionally, other multi-purpose validation tools were developed such as SHIVA, DIVER, EITHER and EMBODY [10]. None of these methods is fully incorporated into a life-cycle model.

2. BACKGROUND

Validation can and should be performed at any and all levels of the system development stages [1] [11]. O'Keefe et al. [11] and Lee et al. [12] have looked into incorporating validation into a conceptual software development model. However, their success was limited. After working with different general validation approaches, O'Keefe et al. [11] concluded that "We should build validation into the development cycle". However, none of the existing methods perform formal validation across all development phases. Furthermore, none of the mainstream methods presented here is completely based on a life-cycle model for system development. In this paper, we introduce a formal method towards achieving the goal of having a guided and incremental validation. This will be done through CommonKADS. Anderson et al. [13] conducted a study to measure the benefits of incremental validation using many systems in many domains. They came out with the following conclusions:

1. Rates of uncovering errors early in development were better.

2. Validation and verification found 2.3 to 5.5 errors per thousand lines of code.
3. Over 85% of the found errors affect reliability and maintainability.
4. Early error detection saved 20-28% of validation costs if validation begins at coding phase.
5. Incremental validation saved 92-100% of validation costs if validation begins at requirement phase.

Gilb et al. [14] did a similar study and illustrated their results. They concluded that when validation is postponed, costs will grow exponentially. Incremental validation can prevent this increase in costs. Incremental validation helps the user in getting frequent information about the development process of the system, helps the knowledge engineer in finding early comprehensive solutions instead of rushing fixes to meet deadlines and helps the manager in decision-making and instant feedback.

3. COMMONKADS SET OF MODELS

CommonKADS (Knowledge Acquisition and Design Support) is based on KADS. It concentrates on the conceptual structure of the knowledge and the system. The most accepted KBS development method is CommonKADS. It doesn't currently include guidelines for validation, verification or testing in any of its models. The six CommonKADS models are categorized in three groups [15]:

1. Context Models:

Organization model: Supports the description and the analysis of the organization.

Task model: Describes the tasks that might be performed by the system within the organization.

Agent model: Supports the capabilities, constraints and roles of the agents performing the tasks.

2. Concept Models:

Knowledge/Expertise model: Supports the description of the knowledge invoked in the tasks.

Communication model: Describes the relation between the agents, their interaction and their communication.

3. Artifact Models:

Design Model: Supports the design and the structure of the system.

Figure 1 illustrates the CommonKADS set of models. These models are presented in worksheets, UML diagrams, pseudo code and text. All the models are mapped to implementation to form the system. Tools were developed to help in implementing CommonKADS such as Model-K and OMOS [15]. The development of these and other tools reflects the general acceptance of CommonKADS by the KBS development community. Conceptual model languages had been introduced to support CommonKADS representation formally such as ML^2, VITAL and FORKADS [15]. CommonKADS supports reusability, and offers guidelines for the developer to achieve high quality systems. CommonKADS is a knowledge representation dependent model and was not created independently from other software models. Rather, other software models (e.g. object-oriented paradigm) influenced the definition of CommonKADS. CommonKADS has a powerful organizational sub-model that can represent many domains. CommonKADS offers a *de facto* standard for building systems and ensures a modular approach. CommonKADS is the most used knowledge-based systems lifecycle model and is the most accepted [15] [16]. Considering all the advantages of CommonKADS mentioned above, it should be no surprise that we chose it as our knowledge-based system development model for our validation method

In the next three sections, the validation lifecycle, test cases extraction and reduction are introduced.

4. MAVERICK

Incremental validation is based on the idea that "prevention is better than cure". Incremental validation locates the problem in its early stages. For example, if there is an error that is created during knowledge elicitation as a result of miscommunication between the expert and the knowledge engineer, incremental validation helps in identifying the error before it's absorbed into the design and then the implementation. The deeper this error is absorbed the harder it will be to identify it. Therefore, based on the CommonKADS structure, MAVERICK is performed at five levels in the following order:

1. Context Test Cases Extraction: This step defines the test cases that need to be executed after defining the first three models (the Context models: Organization, Task and Agent).
2. Analysis Test Cases Extraction: In this step, the test cases are extracted from the communication and knowledge models. In CommonKADS, the analysis phase is done after building five models: organization, task, agent, communication and knowledge. These five models represent all the requirements of the system. After those five models are defined and before moving into the design model, analysis validation is performed. Inspection validation starts here, first step of inspection validation is analysis validation. This validation checks for conflicting requirements, missing aspects in the analysis and any ambiguities. This validation is performed by the experts and the knowledge engineer manually on all the documents and diagrams defined so far.

3. Design Test Cases Extraction: This is the last step for test case extraction where test cases are extracted from the design model. Inspection validation stops here, second step of inspection validation is design validation. It is performed after this step and before implementation of the knowledge-based system starts. Validation inspects the Class diagrams for DM1 to check the initial design. DM1 represents the whole system.
4. Spiral System Implementation: Implementation of the system is performed iteratively. While iterating, system development proceeds and validation is performed by executing test cases. Test cases are selected in every iteration by the CBV tool described later in this dissertation.
5. Spiral System Validation: Validation is performed spirally, test case selection occurs iteratively and test cases are executed on the system. The validation approach is discussed and introduced in greater detail in section 6. Steps 4 and 5 are indicated to as CBV.
Figure 1 illustrates our general approach towards performing incremental validation within the CommonKADS steps. Different validation steps are performed during the building of the CommonKADS models and the system.

5. COMMONKADS TEST CASE EXTRACTION

The test case extraction starts early, while defining the Organization model. The first worksheet from which to extract cases from is OM3. OM1, OM2 and OM4 are used to introduce the knowledge engineer to the process that needs to be developed into the knowledge-based system and the assets of the organization. Nothing from OM1, OM2 and OM4 is used as a part of the target system. OM3 is the process break down sheet. All the processes in OM3 breakdown into the Task model for more details. In this sheet, each task is defined with who is performing it and what part of knowledge is needed for it. This worksheet doesn't involve the essence of the task. That's the goal of the Task model. Example: Task1 is performed by Paul Hewson and for this task documents 1 and 2 are needed. When the system is built, a test case would be necessary to check the availability of the needed documents when this task is performed by the mentioned employee. The test case would have the following format:
1. Test case ID: *1*.
2. CommonKADS model: *Organizational model (worksheet: OM3 (organization tasks))*.
3. Input variables: *Paul Hewson's user name and password.*
4. Test setup values: *Logout from all accounts and close all documents.*
5. Test execution steps: *Run task 1 by clicking on the "start task" button, log in as Paul Hewson* and *click on "get documents 1 and 2"*
6. Expected solution: *Two PDF files opening on your computer with documents 1 and 2.*
7. System's solution: *Document 1 opened but document 2 didn't.*
8. Local Importance: *2.5*.
9. Number of execution times: *1*.
10. Informal description: *Paul Hewson needs access to documents 1 and 2 with task 1.*

OM2 has a "culture and power" part in the worksheet that deals with social issues, political constraints and rules of thumbs at the organization. This part doesn't apply to many organizations, but in case it's necessary, then for every point in this part of the worksheet there should be test cases to cover it.

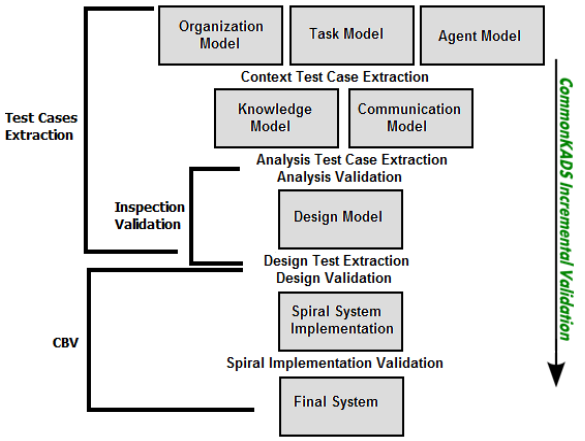

Figure 1 MAVERICK

An important part where test cases are to be extracted is the worksheet TM1. TM2 deals with making the knowledge engineer familiar with assigning tasks to knowledge. It won't be used for test case extraction. In worksheet TM1 however, each task is likely to need a number of test cases, where the inputs of the test case come from the dependency and flow section. In this section, the input objects and the output objects are defined, which are then transformed to the input variables and the test setup values of the test case. In the expected output part of the test case format, the quality and performance part are used. The quality and performance part in the worksheet deals with expected outcome of the task; this would be the criteria for the test case failure or success. Furthermore, in TM1, one part discusses the preconditions and the post conditions of the task. For each condition a set of test cases should be defined. Worksheet AM1 defines the agents' access to the system. Test cases extracted from this worksheet are related to security, roles and accesses. As previously introduced in test case 1 example, Paul Hewson needed access to task 1. Similar test cases are extracted from AM1. The Knowledge model is a critical model in CommonKADS as it is transformed to represent the knowledge base. In CommonKADS, the inference structure and the domain schemas

provide the set of test cases to validate the knowledge. The inferences and the transfer functions are parts of the inference structure, each instance of them is presented in a test case. KM1 is a central worksheet for test case extraction as it defines important parts of the knowledge. The knowledge engineer might need to present some domain requirements in the domain schemas of the Knowledge model, as every object in the domain schema is presented by a test case (refer to test case 2 for an example). In KM1, an important part is the "scenarios" section where any scenario related to a certain part of the knowledge is introduced. Other parts in this worksheet include a glossary of terms, the elicitation material and other sections that will not be transformed into a knowledge-based system. An example of a scenario and a test case: scenario (The employee Dave Evans needs knowledge about credit cards overdraft fees to answer a bank's client). A test case for this scenario would be:

1. Test case ID: *2*.
2. CommonKADS model: *Knowledge model (worksheet: KM1)*.
3. Input variables: *Dave Evans user name and password.*
4. Test setup values: *Run the credit card sub-system.*
5. Test execution steps: *log in as Dave Evans*, *enter a clients name and account number*, *click on "Display credit cards fees rules"*
6. Expected solution: *Correct overdraft fees list of rules should display to employee Dave Evans.*
7. System's solution: *Correct overdraft fees list of rules displayed to employee Dave Evans.*
8. Local Importance: *1.75.*
9. Number of execution times: *1*.
10. Informal description: *Overdraft fees rules display when required by the employee.*

The Communication model defines the interaction between the tasks, the agents and the system. CM1 and CM2 are used for test case extraction as both of these worksheets components are built into the targeted knowledge-based system. In CM1 the constraints section is used to extract test cases and the agents involved in this test case.

CM2 defines the contents of the communication messages and the control over the messages, each transaction needs to be tested using at least one test case. In the Communication model, all the information exchange, message sending and processes between agents are represented in a pseudo code defined specifically for CommonKADS.

For each pseudo construct, a set of test cases should be defined. For example, a message for a new loan is to be sent from the teller Adam Clayton to the management department employee Larry Mullen indicating that a new loan is granted to a client ahs the following construct: *SEND tramsaction1(loan granted) from teller to RECEIVE management*.

The dialogue diagram in the Communication model is used to test the sequence of the tasks performed by the system and the agents. The Design model in CommonKADS represents the initial design of the targeted system. DM2, DM3 and DM4 are worksheets that help the knowledge engineer to select the hardware platform, software platform and all technical issues related with building the system, but the real system design is found in DM1. DM1 defines all the subsystems. Test case extraction from this worksheet targets the issue of the integration of those subsystems. Relation between the subsystems is reflected by communication between the subsystems and the tasks sequencing among subsystems. In all the subsystems, the domain specifications are introduced in the Organizational, Task and Agent models. The functional specifications are presented in the Knowledge and Communication models. Using the test case extraction step defined in this section, all the aspects of the knowledge-base are covered and test cases are generated from all the entities included in the targeted system.

6. TEST CASE REDUCTION (CONTEXT BASED VALIDATION)

In our method, Knowledge-based system development and validation are performed using the spiral model. At any iteration of development, variables' values need to be modified and the system undergoes refinement. This work reduces the number of test cases based on the user's needs and the context of validation. This is where the term context-based validation (CBV) came from. In problem solving, the context would inherently contain much knowledge about the situation's context in which the problem is to be solved or the problem's environment [17]. In the case of test case reduction, testing is intensified for the model that failed the most in the previous testing cycle. To reduce the number of test cases, the knowledge engineer chooses what test cases to remove. This is not performed manually; it is performed spirally by the knowledge engineer and based on the CommonKADS models.

Before the knowledge engineer starts with system implementation, it is necessary to define a number of control variables that are used to select what test cases to be used in every cycle. These variables are:

1. Local Importance (LI): Each test case is assigned a local importance variable that falls between 1 and 5. *Local importance = Average of (dependency + domain importance + criticality + occurrence)*. Local importance is a factor of dependency (Value assigned from 1-5), domain importance (Value assigned from 1-5), criticality (Value assigned from 1-5) and occurrence (Value assigned from 1-5). All the values are defined by the knowledge engineer and the expert. Additionally, the frequency of the task is indicated in TM2, this is the basis for defining the occurrence

factor. Dependency is in the nature of CommonKADS, the Design model depends on the Knowledge and Communication models, which depend on defining the task and the Agent models which are both based on the Organization model which is defined based on the knowledge elicitation. The Organization model has the lowest dependency rate (1) and the Design model has the highest dependency rate (5).

2. Model Weight (MW): Every CommonKADS model is assigned a weight after any iteration of development. Initially all the models have the same importance (MW is set to 5), but when the development starts, model weights will constantly change based on the outcomes of the test cases. The model weight values fall between one and ten. Model weight reflects the assurance level in testing for the CommonKADS models. When the assurance of all models reaches 10 and implementation is done, validation stops.

3. N: Represents the number of test cases to be selected in any iteration.

4. Global Importance (GI): This variable is used to decide what test cases to select in any iteration. *Global Importance = Local Importance * Model Weight.*

Approaches to test case reduction have varied between random, formal and informal. Using a well established model like CommonKADS provides a solid ground and an assurance that all the aspects of the system are covered, and that the test cases extracted using this method make sure that the system is well covered for tests. The steps of CBV presented in figure 2 are:

1. Extract test cases from the worksheets and diagrams. Set all the parameters defined previously. Assign each test case to a CommonKADS model
2. Assign *local importance* for each test case.
3. Set the size of test case subset: *N*, initially all the test cases that have global importance more than 20 (LI*MW = 4*5 = 20). All test cases with local importance of 4 or 5 needs to be selected, cases with 1, 2 and 3 importance are less important.
4. Set all models' *weights/assurance* to 5
5. Calculate *global importance* = local importance * model weight. Sort test cases according to global importance
6. Start implementation using the spiral model
7. At the end of the first iteration, select N number of test cases. From the ordered list pick test cases 1 to n.
8. Execute the test cases on the system, and record the results
9. Based on results for each CommonKADS model test cases, re calculate assurance for each model. Example: if 30% of test cases of a certain model went wrong, that model's assurance will be 7 using the following formula: *100 - (percentage of successful test case)/10*
10. Recalculate global importance of test cases and reorder
11. Refine system; go to next iteration (Manual)
12. *Flag* test cases with a positive outcome (not to be picked again unless a change to their status was made), flag test cases with unexpected outcomes (this is used to make sure that the test case is reselected before end of validation), select different test cases every next iteration
13. Stop when assurance of all models is equal to 10. Assurance of all models = average of all models assurances.

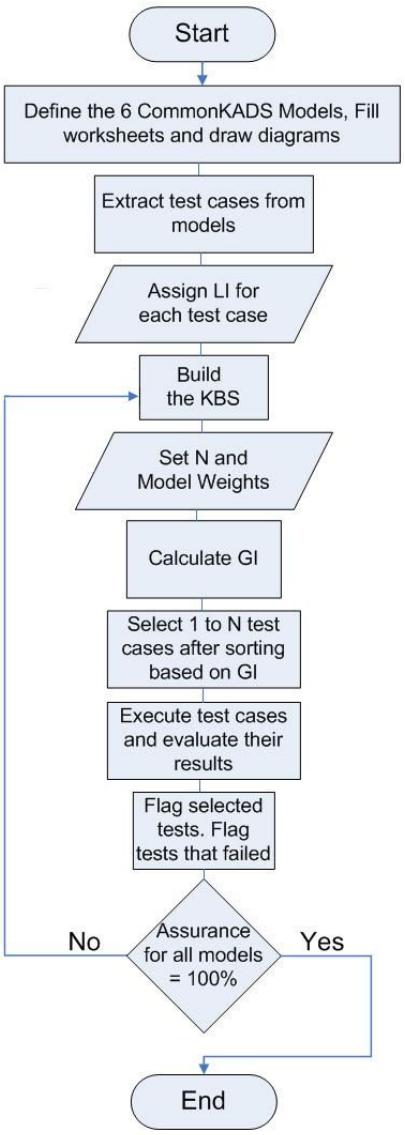

Figure 2 CBV

Test case reduction steps are illustrated in Figure 2.

A Java tool was developed to select, sort and recommend test cases for the knowledge engineer from the universal set of test cases using the method presented in this paper. Figure 3 is a screen shot that represents one panel from the seven panels in the tool. This tool updates the test cases instantly and sorts all the test cases in real time for selection of N test cases.

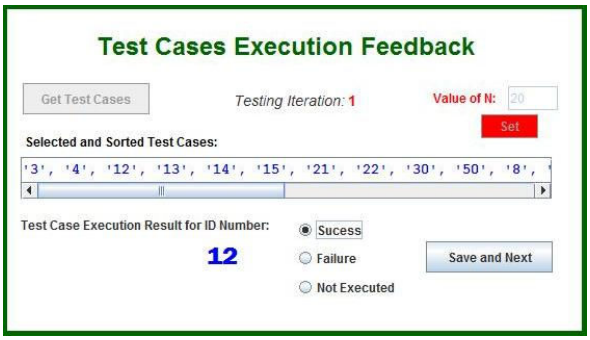

Figure 3 Test execution Java panel

7. CONCLUSIONS

The approach presented in this paper requires some manual work from the knowledge engineer or any other person performing validation but it has many advantages. Advantages of this approach are:

1. Flexibility: the weights and the models could be changed to any other values. This gives the knowledge engineer full control.
2. Usage-oriented: this approach is based on the user needs and a real time testing feedback. It is not a static function, rather a resilient one.
3. It's based on a comprehensive, well defined and well structured model: This function is based on CommonKADS, which as discussed previously, has many advantages.
4. Effort and time reduction: reducing the number of test cases reduces effort and time.

In this paper, we introduced a validation method based on a lifecycle model called CommonKADS; we introduced the validation lifecycle, extracting test cases from the six CommonKADS models and reducing the number of executed test cases and thus reduce time, manpower and expenses.

8. REFERENCES

[1] A.J. Gonzalez, and V. Barr, "Validation and Verification of Intelligent Systems – what are they and how are they different" Proceedings of the Journal of Experimental &Theoretical Artificial Intelligence, pp.407-420, 2000

[2] E. Santos Jr., and H. Dinh, "Consistency of Test Case in Validation of Bayesian Knowledge Bases", Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence – ICTAI 2004.

[3] R. Knauf, S. Tsuruta, and A.J Gonzalez, "Towards Reducing Human Involvement in Validation of Knowledge- Based Systems", Proceedings of the IEEE transaction on Systems, Man and Cybernetics, Volume 37, pp.120-131, January 2007

[4] N. Zlatareva and A. Preece, "State of the Art in Automated Validation of Knowledge-Based Systems", Proceedings of the journal of Expert Systems with Applications, pp.151-168, 1994

[5] A. Ginsberg. S. Weiss, and P. Politakis, "SEEK2: A Generalized Approach to Automatic Knowledge-base Refinement" Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 367-374, 1985

[6] K. Abdallah, T. Mohammad, and F. Louis, "Validation of Intelligent Systems: a Critical Study and a tool, Corus", Proceedings of the International Journal of Soft Computing, pp.191-198, 2007.

[7] S. Smith and A. Kandel, "Validation of Expert Systems" Proceedings of the Third Florida Artificial Intelligence Research Symposium (FLAIRS), pp.197-201, 1990

[8] Wu, C. and Lee, S. "KJ3- a tool assisting formal validation of knowledge-based systems", Proceedings of the Int. J. Human-Computer Studies, pp. 495-525, 2002.

[9] J. Herrmann, K. Jantke, and R. Knauf, "Using Structural Knowledge for System Validation" Proceedings of the 10th FALIRS Conference, pp. 82-86, 1997.

[10] S. Lockwood, and Z. Chen, "Knowledge Validation of Engineering Expert Systems" Proceedings of the Journal of Advances in Software Engineering, pp. 97-104, 1995.

[11] R. O'Keefe, R. Balci, and E. Smith, "Validating Expert System Performance" IEEE, Proceedings of the IEEE Expert, Volume 2, pp.81-90, 1987

[12] S. Lee, and R. O'Keefe, "Developing a Strategy for Expert System Validation and Verification" , IEEE, Proceedings of the IEEE Transaction on systems, Man and Cybernetics, Volume 24, pp.643-655, 1994.

[13] C. Anderson, T. Thelin, P. Runeson,N. Dzamashvili, "An Experimental Evaluation of Inspection and testing for Detection of Design Faults", Proceedings of the International Symposium on Empirical Software Engineering – ISESE 2003

[14] T. Gilb, and D. Graham "Software Inspection" Published by Addison Wesley 1993

[15] G. Shreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. Van De Velde, and B. Wielinga,. "Knowledge Engineering and Management-The CommonKADS Methodology" published by The MIT Press 2000

[16] A. Al Korany, K. Shaalan, H. Baraka, and A. Rafea, "An Approach for Automating the Verification of KADS-Based Expert Systems" Proceedings of the 7th International Conference on Applied Informatics and Communications- (WSEAS), pp. 1-22, 2007

[17] A.J Gonzalez, B. Stensrud, and G. Barret, "Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior", Proceedings of the International Journal of Intelligent Systems, pp. 822-847, 2008