The VIVA Method: A Life-Cycle Independent Approach to KBS Validation

Steven A. Wells
AIE Department, Lloyd's Register
29, Wellesley Road, Croydon CRO 2AJ, England
steve@aie.lreg.co.uk

Abstract

This paper describes the VIVA method; a life-cycle independent method for the validation of Knowledge-Based Systems (KBS). The method is based upon the VIVA validation framework, a set of products by which a KBS development can be described. By assessing properties of these products, and properties of the links between the products, a framework for validation throughout the KBS life-cycle is defined.

Introduction

The VIVA[1] Technical Annex (VIVA 92) identifies the needs for a Knowledge-Based System validation method which covers the whole of the development life-cycle. These needs arise from identified problems with software-based validation, which can be summarised as follows:

- It is not possible to determine the validity of a system from the software alone.
- It is not possible to ensure that an implemented system will be valid unless validation is carried out throughout the life-cycle.

A validation method, designed to be integratable with existing KBS development approaches and which addresses these and other issues, is being developed in the VIVA project. This paper presents a brief overview of the method as it currently stands, and describes future work plans. Lloyd's Register will build on the VIVA method to allow the assessment of both KBS, and systems containing KBS components. This will significantly increase Lloyd's Register's scope for software assessment in the future.

[1]The Research reported here is being carried out in the course of the VIVA Project. The project is partially funded by the ESPRIT Programme of the Commission of European Communities as project 6125. The partners in this project are University of Aberdeen (UK), Computer Resources International (DK), CISI Inginerie (FR), European Space Agency (NL), Lloyd's Register (UK), Logica (UK) and the University of Savoie (FR)

The Derivation of Requirements for a Validation Method

In order to determine requirements for a validation method covering the whole of the KBS life-cycle, an analysis of current KBS development methods was carried out early in the VIVA project. The development methods studied included VITAL (Kontio and Rouge 91), CommonKADS (Aamodt *et al* 92, de Hoog *et al* 92, Taylor *et al* 92), KOD (Vogel 90) and the European Space Agency (ESA) KBS development guidelines (Allard 92). A number of differences were found, which are presented below.

Ordering.

There is no unique ordering of development phases between life-cycles. It was not always the case that life-cycle *phase X* came before life-cycle *phase Y*. For instance, the *Knowledge Model development* phase is defined in VITAL to occur after the *Requirements* phase, but in CommonKADS, it may come before or after.

Transformation.

There is no strict transformation between products in development phases. The Knowledge Model, for instance, would not necessarily be transformed *en masse* into another product in any life-cycle. The inferencing may be transformed into part of the Functional Specification, but the Domain Knowledge may be directly coded. This would be the case if the domain knowledge consisted of an *is_a* hierarchy, and the implementation environment provided support for direct coding of hierarchies, as in, for instance, KEE.

Non-Standardisation.

The above diffferences are compounded by the lack of standardisation in KBS life-cycles. The "same" development methodology differs between organisations, and even between projects. There are a great many "KADS" approaches, for instance.

In order to define a suitable framework for the VIVA method, it thus became necessary to take a product-based view of the KBS life-cycle. By defining a minimum set of products which could be used to describe

existing and future, KBS development approaches, the applicability of the VIVA method to any arbitrary development method should be ensured.

The VIVA Validation Framework

The VIVA method consists of two parts, shown below.

- A set of products by which to describe a KBS application.
- A set of steps describing how the products can be used to validate a KBS application.

This paper concentrates on aspects of the products known as the VIVA Validation Framework (Wells 92). The products are describes in Table 1. This framework defines the contents and features of the products and their constituent sub-products. The products in the set possess two types of properties, which form the basis of the VIVA method:

- Properties of Products.
- Properties of Inter- and Intra-Product links.

KBS will be validated by assessing such properties using methods and tools developed on the VIVA project.

Products in the KBS Life-Cycle
Situation. A description of the environment into which the KBS will fit. An example would be an organisation model.
Requirements. Functional and non-functional requirements of the system.
Knowledge Model. A Knowledge Level model. Usually a description of an expert's, or system's, reasoning capabilities.
Functional Specification. A specification of the functional capabilities of a system.
Design. A specification of the architecture, interfaces, data structures and algorithms to be used in the code.
Code. The implemented software, including documentation.

Table 1: The VIVA Validation Framework

As the VIVA method will be applicable to any development life-cycle, it is not possible to fully describe all possible properties and techniques. What the method will provide will be a framework which can be adapted to the specific attributes of a life-cycle or development. This framework will allow the user of the method to describe the products, sub-products and links of an application to a level of detail which will allow the complete description to be user-completed. Figure 1 illustrates the level of support the method will provide.

The products and sub-products in the Validation Framework are related to each other in one of two ways, *structural* or *transformational.*

Structural links are usually, but not solely, internal to a particular product. Examples could be a cross-reference in a requirements document, or a calling relation between functions.

Transformational links are those links where a "source" product at one end of a link is used to generate the "target" product at the other. These are usually, but not exclusively, external, or between product links. Examples are the transformation of inference function designs into their implementation in code, or the transformational of a particular individual requirement into its corresponding functional specification.

In practice, there is not a clear-cut distinction between structural and transformational links. Once transformed, a link often becomes structural. An example is the link between a design and its implementation in code. During maintenance, analysis is carried out at the design level, and then mapped on to the code. Thus, the design-code link has become structural.

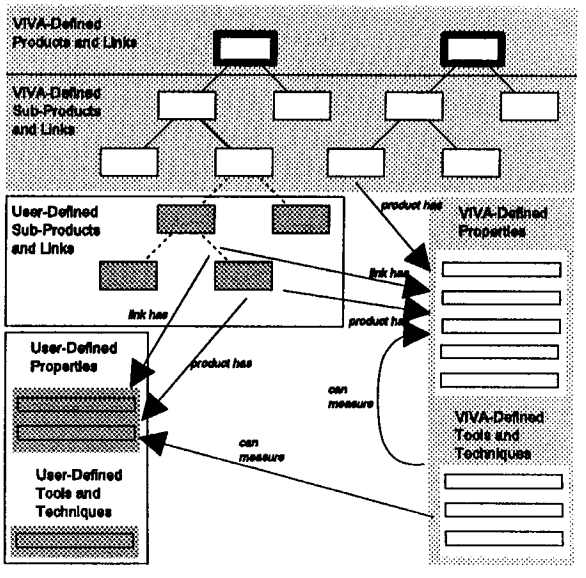

Figure 1: Level of Support of the VIVA Method

Figure 2 shows a subset of the links between three of the products, namely, the Knowledge Model, the Design and the Code. The links are further defined in (Wells *et al* 92), and some are further explained below.

The structural links within the Knowledge Model show the links between domain models and knowledge roles, and between basic tasks and primitive inferences.

The transformational link between the Knowledge Model, in this case a CommonKADS Knowledge model, and the Logical design, is the link between the top-level task in the task knowledge of the Knowledge Model, and the top-level control function in the logical design.

The transformational link between the domain knowledge of the Knowledge Model and the Knowledge Base illustrates the direct coding of domain knowledge.

The transformational link between the inferencing design and the inference in the code represents the implementation of an inference function design.

The VIVA method views a development as consisting of a (*sub*-)set of the products and sub-products in the Validation Framework and the structural and transformational links between them.

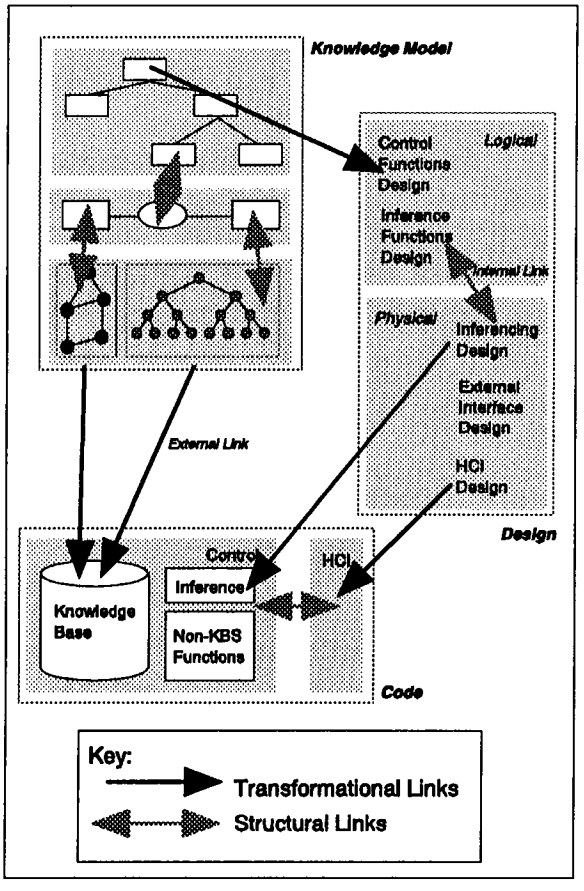

Figure 2: Examples of Link Types

In validating a KBS application using the VIVA method, the products and sub-products of the Validation Framework which will be constructed in the application are identified, along with the structural and transformational links between them. The properties of these products and links define the set of Validation Opportunities for the application. By defining the structure of the VIVA method in this manner, it allows the method to be used in both post-hoc (Validity Assessment) and developmental (Validity Assurance) validation.

The Derivation of Validation Specifications

The validation opportunities for a development will be a (*potentially large*) set of properties. These must be transformed into a validation specification, against which the development can be assessed. There are many different types of validation opportunities.

The properties of products can be general properties, applicable to any product or sub-product. Examples of general properties of a product are correctness and completeness. An example of a specific property is *inference re-use.* (*The property of an inference being in a re-usable library or not*).

Links also have certain general properties, which are defined merely by identifying such a link. These include:

- *existence*; the existence of the link in the current development,
- *connection*; the existence of the entities at each end of a link and
- *typing*; the correctness of the type of link.

Other properties are specific to one type of link, for instance, for structural links:

- *referencing*; whether entities at each end of the link correctly reference themselves, and each other.

For transformational links:

- *correctness*; correctness of the transformation, for instance, provably correct.
- *completeness and relevance*; whether or not all, and only, the information required in the transformed entity has been preserved.

Other properties are specific to the particular link in question, for instance:

- *(in a CommonKADS Knowledge Model) task-inference link correctness*; whether or not primitive tasks and the inferences they refer to have the same name, or
- *(in an implemented Knowledge Base) rulebase-domain model reference existence*; whether or not all rules reference defined concepts.

Other properties of both products and links may be defined from the needs of a particular application. As an example, an application which had a critical need for maintainability may specify a property of speed of determining correctness on the links between design and code.

The properties of products and links are of two types, formalisable and non-formalisable (Laurent 92). Formalisable properties are those which can be stated in an unambiguous way and which can be said to present or absent. An example is

No rule shall be self-contradictory.

This can be expressed formally, and the presence or absence determined. There is no debate about the meaning of the result of the test. However, other properties are not formalisable in this manner. An example would be:

The interface shall be acceptable to users.

It is possible to pseudo-formalise (Laurent 92) non-formalisable properties. The previous example could be pseudo-formalised as:

If 100 users are asked, over 80% shall report that the interface is acceptable.

The truth, or otherwise, of this specification could be tested, but this does not guarantee the truth, or otherwise, of the original property. It is believed, in this case, that 80% is a good threshold, but there is no direct, unambiguous, mapping from the property to the specification.

In practice, some formalisable specifications are untestable, due, for instance, to problems of size or combinatorial explosion. Such specifications are thus also expressed pseudo-formally. As an example, it is not usually feasible to carry out 100% testing, so a range of relevant tests would be generated. A tool such as SYCOJET (Vignollet 92) could be used for this purpose. Another example of a formal property which perhaps could not be formally assessed is the property that a system will correctly react to all invalid inputs.

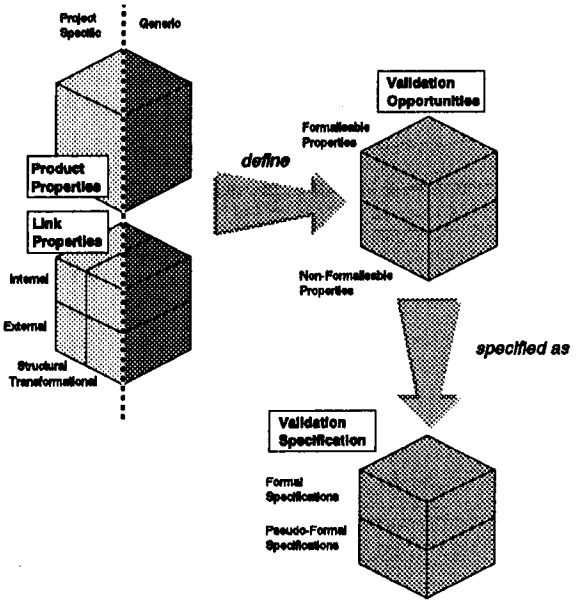

Figure 3: Derivation of Validation Specifications

Thus, in order to define a validation specification, validation opportunities are transformed into a set of formal and pseudo-formal specifications as depicted in Figure 3.

The VIVA method starts by identifying the development products, and through the application of a number of steps, yields a validation specification. This specification consists of formal and pseudo-formal validation specifications. Future development of the method will identify the tools and techniques to carry out the assessment of the specification against the development.

Future Work

The definition of the VIVA method is on-going, and will continue throughout the remainder of the project, which finishes in late 1995. Future work will focus on the following topics:

Products, Links and Properties.

The properties of the products and links will be further defined. Work is currently underway to determine what tool support will be required to assess the range of validation properties identified so far. This work will focus on both existing tools and new tool requirements. The work on properties will be extended to provide guidance on the particular properties which need to be assessed for a particular application. For instance, which properties should be assessed in a safety-critical domain? In an application with critical maintenance needs?, *etc.*

Alignment with Standards and Software Engineering Practice.

The method will align itself alongside existing Software Engineering standards and will be integrated with existing assessment methods (e.g. (Denvir and Neil 92)) to allow KBS and embedded KBS assessments to be performed. If this is successful, it will also provide justification for the approach taken. Software Engineering practices which increase confidence in applications, such as Hazard Analysis, will also be included.

Method Guidelines.

At some level of detail, as shown in Figure 1, the user of the method will need to define sub-products and properties particular to the application being assessed. The guidelines will provide support for this activity, and some examples are shown below:

- *Validation Opportunity Identification and Specification.* e.g. Is the set of products in the development sufficient?, Which properties can be formalised?
- *The Relation between project features and specifications.* i.e. Given that a project has high criticality and limited resources, which of the possible validation opportunities should be in the validation specification? Which properties should be formalised?

- *Available techniques and tools.* The guidelines will show which techniques and tools are available, and the cost of their use. Thus the guidelines will answer such questions as; Which tools and techniques should be used, and in which order?

Conclusions

This paper presented the current status and some of the future work plans of the VIVA method, a life-cycle independent approach to KBS validation. Although the method is in its infancy, it appears that the goal of the method, to provide life-cycle independent KBS validation throughout the complete development cycle, is acheivable. As the method progresses, it will be integrated into conventional software assessment strategies, allowing the assessment of systems containing KBS components.

Acknowledgement

The author would like to thank the Committee of Lloyd's Register for permission to publish this paper. The views expressed in this paper are those of the author, and not necessarily the views of Lloyd's Register.

References

Aamodt A., Bredeweg B., Breuker J., Duursma C., Lockenhoff C., Orsvarn K., Top J., Valente A., van de Velde W., 1991. *The CommonKADS Library*, KADS-II/T1.3/VuB/TR/005/1.0

Allard, F. 1992. *Software Quality and Artificial Intelligence, Knowledge-Based Systems: Recommendations for the Development Life-Cycle*, WGS/FA/92.55

Vignollet, L., 1992. *SYCOJET: Une Approche Pour la Construction Automatique des Jeux de Tests Pour les Systemes a Base de Connaissances*, These de l'Universite de Savoie - LIA, Chambery, Mars 1993

Bright, C., Martil, R., Williams, D. and Rajan, T., 1992. The KADS-II Framework for Project Management, In Proceedings of SGES 1st KBS Methodologies Workshop

Denvir, T., and Neil M., 1992. *Lloyd's Register Model for Software Dependability Assessment, V1.1*, Lloyd's Register of Shipping, August 1992

de Hoog R., Martil R., Wielinga B., Taylor R., Bright C., 1992. *The Common KADS Model Set*, KADS-II/WPI-II/RR/UvA/018/3.2

Kontio, J., and Rouge, A., 1991. *VITAL Life-Cycle Guide*, VITAL Deliverable DD111

Laurent, J-P., 1992. *First Entries for the VIVA Glossary*, VIVA Deliverable TN-610

Taylor R., Bright C., Menezes W., Groth J., 1992. *Principles of the LCM*, KADS-II/T2.1/TR/TRMC /010/1.0, Volume 1

VIVA, 1992. *(The VIVA Consortium), VIVA Technical Annex*, V4.0

Vogel, C., 1992. *Manual KOD*, CISI Inginerie

Wells, S. A., 1992. *Internal Report 3.1: The VIVA Validation Framework*, VIVA Deliverable TN-302.1.0-WP3-LR

Wells, S. A., Allard, F., Ayel, M., Millington, P., Nielsen, L. B., and Thierry, A., 1992. *Internal Report 3.2: Guidelines for a KBS Validation Method*, VIVA Deliverable TN-308.1.0-WP3-LR