Towards a Conceptual Framework for Expert System Validation

Pedro Meseguer
IIIA
CEAB-CSIC
Cami Sta. Barbara, 17300 Blanes (Girona) SPAIN
pedro@ceab.es

Abstract. In this paper we address a number of fundamental questions around validation of expert systems (ESs), an emerging field that still lacks of a comprehensive treatment in the literature. The final aim is to get a conceptual framework for ES validation, what is a difficult task because of the immaturity of the field. We analyze ES validation from four points of view. First, we consider the meaning of validation in ESs, what raises some terminological questions. We propose precise definitions for the terms verification, validation, evaluation and testing, that are in compliance with their respective meanings in software engineering. Second, we consider a number of validation targets, the elements or processes on which some validation activity can be made. Third, we discuss briefly specific techniques to achieve an effective validation for these targets. And fourth, we locate these validation targets in an ES life-cycle.

In this paper, we address a number of fundamental questions around the validation of expert systems (ESs). The final aim is to get a conceptual framework for ES validation, where the meaning and main components of ES validation are defined and understood in an integrated way. This framework also consider those ES aspects that are suitable for some kind of validation, as well as specific techniques to achieve validation in practice. Developing a conceptual framework is a difficult task because of the immaturity of the field. Different validation aspects are unevenly developed, appearing clearer those parts to which more work has been devoted. Fundamental definitions such as, for instance, the meaning of ES validation, are still not consolidated. Many validation aspects are today elements for debate in the research community. Therefore, the work we present here reflects the lacks and defects of the current state. Nevertheless, drawing the whole validation picture is a useful exercise that, at least, will allow us to identify the most obscure parts.

We analyze ES validation from four points of view. First, we discuss the meaning of validation in ES. This raises some terminological questions about the precise meaning of terms like verification, validation, evaluation or testing, commonly used in the literature. We propose precise definitions for these terms using the concepts of totally and partially formalizable requirements. These definitions are in compliance with the established meanings for these terms in software engineering, and they reflect our experience in validating PNEUMON-IA, an actual ES devoted to pneumonia diagnosis [Verdaguer 89]. Second, we identify a number of validation targets as the elements and processes on which some kind of validation can be performed. We consider the following validation targets: user requirements, knowledge acquisition, expert system architecture, knowledge base structure and contents, inference engine and expert system behavior. Third, we identify specific techniques for each one of these validation targets, aiming for an effective validation. We devote special attention to those aspects that are specific of ESs, although there is a number of points in common with software engineering. And fourth, we locate these validation targets in an ES life-cycle. Every step in the life-cycle has one or several validation activities associated. Then, the whole validation process is the aggregation of all these validation activities.

The structure of this paper reflects these four points of view. Section headings are questions on the meaning, contents, methods and time of validation. Section 1 raises the question *What is Validation?*, analyzing validation in ESs. This section contains a terminological discussion about validation terms in ESs, considering the meaning of these terms in software engineering. Section 2 addresses the question *What Should be Validated?*, considering the previously mentioned validation targets. We analyze the relevance and impact of their validation in the knowledge engineering process. Section 3 investigates the question *How to Validate?*, considering how to perform validation on these validation targets. Section 4 addresses the question *When to Validate?*, looking for the right time in the ES life-cycle for validation activities. Finally, section 5 summarizes the paper.

1 What is Validation?

Validation is not a new topic in computer science. In the context of software engineering, [Adrion et al, 82] define software validation as,

```
Determination of the correctness of a program with respect to
the user needs and requirements.
```

This is an open, non-constructive definition. In practice, no single validation method can assure a complete validation. On the contrary, a large variety of validation methods for conventional software exist with complemented effects.

Differences between ESs and conventional software, specially regarding the kind of problems addressed and the type of programming languages used in both fields, have caused that some authors consider ES validation as somehow different from conventional software validation. We do not share this point of view. An ES is a piece of software that performs some tasks aiming to satisfy the needs of potential users. These needs have to be explicited to allow for an effective definition of the ES purposes. Therefore, the previous definition for validation is perfectly applicable to ESs. From now on, we assume this definition as the right one for ES validation.

The applicability of the conceptual definition does not imply the direct applicability of many validation methods for conventional software to ESs. The existing differences between both fields, mainly regarding the kind of problems considered and the type of development techniques used, justify and demand the existence of specific validation methods for ESs. However, the accumulated experience in conventional software validation cannot be ignored in ES validation. This experience has to be applied considering the essential issues to be solved (shared to a significant extent by both conventional software and ESs) regardless of specific techniques employed that are usually bounded to the style of the used programming languages. A study of conventional validation methods and its potential application to ESs is found in [Rushby 88b].

In the current state of ES validation, terms like *validation*, *verification*, *testing* or *evaluation* are very frequently used. However, their exact definitions and the ways to achieve them in practice are still unclear. In the following we make a tour on these terms, trying to elucidate their meanings and relations. We consider *validation* as a global term, that embodies all others terms as specific aspects.

1.1 Requirements and Specifications

According to the previous definition of validation in software engineering, a program cannot be validated without a set of user requirements. These requirements can be clear or ill-defined, they can be communicated by speech or by written means, or even they can be implicit. But when somebody emits a judgement evaluating a program, these requirements are involved. User requirements are usually expressed in natural language. To enforce their understanding, user requirements are translated into a formal language, generating what is called user requirement specifications, or simply specifications. A specification is a non-ambiguous expression defining some kind of program characteristic or property. A specification must be verifiable and testable with respect to the specified program. Specifications can exist for all the software development steps.

Regarding ESs, user requirements play the same role that in software engineering[1]. However, the complete translation of user requirements into specifications, theoretically assumed in conventional software, does not hold for ESs. There are user requirements that, at the current state of the ES technology, cannot be expressed in formal terms using an specification language[2]. This conclusion, extracted from our own experience [Hoppe & Meseguer, 91], is also shared by [Krause et al, 91] in the context of medical ESs. They used the formal specification language Z to specify the intended behavior of a medical ES using available medical protocols. They conclude that the protocol acts as a specification of the default behavior, but an explicit representation of the circumstances in which this behavior is valid, is missing. So, it seems that the ill-defined nature of ES tasks is the main reason to prevent an accurate and complete specification of the intended ES.

[1] However, a recent survey on the state-of-the-practice reveals a strong reluctance to record written requirements for ESs [Hamilton et al, 91].

[2] We stress that a specification must be verifiable and testable against the ES. Therefore, it should be expressed in terms of actual elements of the ES design and implementation.

1.1.1 Totally and Partially Formalizable Requirements

Although a complete translation of user requirements into specifications seems unfeasible, a partial translation is reachable. This suggests us to divide the user requirements in two classes: *totally formalizable and partially formalizable*[3]. A requirement is totally formalizable if it can be completely translated into specifications. Conversely, a requirement is partially formalizable if it cannot be completely translated into specifications. An example of totally formalizable requirement is:

"The KB should be *structurally correct*"

The concept of structural correctness for KBs is well-defined, so this requirement can be translated into the following specifications (assuming the rule-based paradigm):

1. The KB objects should be syntactically correct.
2. The KB should be contradiction-free.
3. The KB should not contain redundancies.
4. The KB should be cycle-free.
5. All the KB objects should be potentially usable.

These specifications are expressed in natural language for readability reasons, but they can be clearly expressed in a formal language. Considering medical ESs, an example of partially formalizable requirement is:

"The ES has to provide an *acceptable* diagnosis for all the *typical* cases"

In many medical domains there is not a sharp definition of what is an acceptable diagnosis for a case. In addition, the concept of typicality is not well-defined and, in occasions, it is a matter of personal preference. Due to the lack of a precise definition for these concepts, this requirement cannot be *completely* translated into specifications. However, it can be partially translated into specifications, which capture to a certain extent the meaning of the requirement. For instance, in the context of pneumonia diagnosis, the previous requirement about correct diagnosis for typical cases can generate the following specification:

[3] These concepts correspond to formalizable and informal specifications introduced in [Hoppe & Meseguer 91]. However, after reading an earlier version of [Laurent 92], I realized that a specification is always formal, so the concept of informal specification was somehow contradictory. I also realized that the insights underlying these concepts (and the examples provided) were aiming to the user requirements. These are the reasons for this terminological, but important, change.

"If rusty sputum is present, the pneumococal etiology has to be considered"

that is expressed in formal terms saying that the ES cannot stop without having pursued the pneumococal etiology as a goal. The translation is performed by human experts, who interpret the requirement and provide some specifications. In occassions, these specifications are tentative and they have to be confirmed or discarded by experimentation. Nevertheless, specifications obtained from partially formalizable requirements are completely formal. This is pointed out in [Laurent 92], who calls them pseudo-specifications but stressing their formal nature.

1.1.2 Service and Competence Requirements

Rushby points out the necessity of requirements for ESs [Rushby 88a]. He divides ES requirements into *service* and *competence* requirements. Service requirements consider the operation of the ES as a software tool: input-output formats, processing rate, operational conditions, and so on. Competence requirements are concerned with the definition of the intended task for an ES, in the sense that it is a typical human task requiring "knowledge". Competence requirements are further subdivided into *desired* and *minimum* requirements. Desired competence requirements describe *how well* the ES is expected to perform. Minimum competence requirements define *how badly* the ES is allowed to perform: they state the threshold for ES acceptance. Rushby admits that desired competence requirements can be vague or incomplete, because they are usually made in comparison with human performance. However, he states that service requirements and minimum competence requirements should be precisely defined, specially for safety critical applications.

Rushby classifies requirements guided by their role in defining the ES task. Our classification of requirements (totally formalizable vs. partially formalizable) is guided by a more syntactical criterion: their complete translation into specifications. A clear parallelism exists between both classifications. Service and minimum competence requirements are totally formalizable requirements, since they generate "specifications that should be traceable, verifiable and testable just like those of conventional software" [Rushby 88a]. Desired competence requirements can be vague or incomplete, so they fall into the class of partially formalizable requirements, to an extent depending on the specific application.

1.1.3 User Requirements at the Knowledge Level

Why do partially formalizable requirements exist?. Why are not all the requirements totally formalizable?. One may think that is a purely technological matter, that the development of better specification languages would solve. Although this point is important (specially for practical reasons), we think on deeper causes to explain the nature of partially formalizable requirements. These causes are intimately related with the question of knowledge and its representation, addressed by Allen Newell in [Newell 82].

Newell differentiates between the *knowledge level* and the *symbol level* in AI systems. At the knowledge level, a system is viewed as an idealized rational agent, who possesses knowledge, goals and actions. This agent is not subject to computational limitations and it behaves according to the principle of rationality:

```
If an agent has knowledge that one of its actions will lead
to one of its goals, then the agent will select that action.
```

At the symbol level, a system is viewed as a collection of of computational entities (facts, rules, frames, etc) that interact according to some predetermined operations (matching, assignement, etc.). The symbol level is the computational realization of the knowledge level. The relation between knowledge and symbol levels is not univoque. On the contrary, a radical incompleteness characterizes the knowledge level that cannot predict the system behavior in many cases. Mapping a knowledge level description into an adequate symbol representation is an open problem, currently considered by a number of cognitive architectures [Chandrasekaran 87] [McDermott 88] [Steels 90].

The distintion bewteen knowledge and symbol level in ESs is very relevant to us. When users express their requirements, specially the functional ones, they are usually considering the ES at the knowledge level. An expert views the ES as another colleague and expresses the requirements assuming that it possesses the background to understand them. However, specifications are clearly at the symbol level. They deal with well-defined computational objects such as facts, certainty degrees, pursued goals or termination conditions. In this view, the difficulties to express user requirements into specifications are explained by the following causes:

- The knowledge level is an approximation of the actual ES behavior in degree as well as in scope. When user requirements at the knowledge level are translated into specifications at the symbol level, their approximate nature causes difficulties in their complete translation.

- The knowledge level is incomplete since it does not determine completely the ES behavior. A user requirement at the knowledge level can be compatible with many possible and alternative specifications at the symbol level. In addition, Newell affirms that knowledge level descriptions may be completed with precise descriptions at the symbol level (mixed systems). These precise descriptions can be seen as the requirements that are totally formalizable.

- The absence of precise and complete mappings between the knowledge and symbol levels is another cause for the partial translation of user requirements into specifications. The development of cognitive architectures at the knowledge level can represent a significant step towards the accurate expression of user requirements.

1.2 Verification

We define *ES verification* as the process of checking an ES against the specifications generated by its formalizable requirements. The verification process produces accurate responses about the satisfaction of each specification. Given that specifications completely contain the meaning of totally formalizable requirements, the verification process allows for a complete checking of them.

Usually, specifications establish conditions that should (or should not) hold for all the possible ES executions. To test these specifications, verification has to be exhaustive, that is to say, it has to analyze all the possible situations where a specification may be violated. Frequently, specifications contain terms involving the dynamic execution of the ES, such as termination, goal pursuing or fact deduction. In order to check accurately these specifications, the verification process has to reproduce faithfully the execution conditions of the specific ES, that is too say, verification has to consider the operational semantics of KB objects [Evertsz 91]. Otherwise, specifications will not be totally checked and the verification results may be questionable.

Totally formalizable requirements can be domain-dependent or domain-independent. An example of domain-dependent requirement in pneumonia diagnosis is that the concepts of typical bacterial pneumonia and atypical pneumonia cannot be considered both with a high degree of certainty for the same patient [Meseguer 92]. This domain-dependent requirement is modeled as an integrity constraint and it is tested as a potential inconsistency. An example of domain-independent requirement is the common requirement about the structural correctness of a KB, that can be decomposed into (1) correct syntax, (2) contradiction-free, (3) no redundant, (4) cycle-free and (5) without useless objects. Some specifications from domain-independent requirements remain somehow implicit. For instance, the trivial specification of correct syntax is often not explicited. A KB containing objects with syntax errors can cause serious problems in the ES function (an ES crash or an erratic behavior). Therefore, the decomposition of requirements into specifications should be careful and detailed. Some work on ES specifications can be found in [Slage et al, 90] [Batarekh et al, 91].

So far, verification has been mainly focused on checking properties coming from the knowledge representation language used in the KB, corresponding to domain-independent requirements. The reason for this predominance relies on the fact that knowledge representation languages offer a higher level of formalization than the knowledge coded using them. In consequence, extracting formal properties of a representation language is more direct than obtaining formal properties of the involved knowledge. Properties of the representation language can only assure some kind of formal correctness in the knowledge expression, but they cannot deal with the knowledge organization and semantics. To check properties more related to knowledge, some extra knowledge with respect to the KB contents is usually required. The reason for this extra knowledge relies on the fact that the KB usually contains enough knowledge to achieve correct conclusions, but this knowledge is not enough to check the correctness of these conclusions. For instance, if we want to check the following property regarding pneumonias,

When *ecthyma-gangrenosum* is present, the main symptoms and signs supporting atypical pneumonias should not be considered.

we need to describe what are the main symptoms and signs[4] for atypical pneumonias. This information is not explicitely recorded in the KB, since a classification of symptom importance is not contained in it. This information is not explicitely recorded in the KB because it is not needed for deduction. The KB has been constructed only considering deduction, without including support for verification. This necessity of specific knowledge for verification, and in general for validation, has to be considered at early stages in ES development in order to collect it using knowledge acquisition techniques.

[4] A symptom is what the patient feels (headache, sickness, etc) while a sign is what the physician objectively obtains from the patient state (temperature, number of leukocytes, etc).

1.3 Evaluation

We define *ES evaluation* as the process of checking an ES against its partially formalizable requirements. This definition is close to the interpretative validation concept given in [Laurent 92]. Given that no complete translation of partially formalizable requirements into specifications is achieved, the evaluation process always contains a subjective element of interpretation.

In ES evaluation we can differentiate three steps: (i) *requirement analysis*, (ii) *specification checking* and (iii) *result interpretation*. The requirement analysis step consists in expressing the partially formalizable requirements into specifications. This step can be tentative or experimental in many cases, given the vagueness and imprecision of the requirements. The specification checking step considers the satisfaction of the specifications for the ES, in the same way they are considered in verification. The result interpretation step consists in, from the specification checking results, assessing to which extent the original requirements are satisfied. The subjective element of interpretation is present in steps (i) and (iii).

An example may help to understand these steps. Let us consider the following partially formalizable requirement for ESs in medical diagnosis:

"The ES has to provide an *acceptable* diagnosis for all the *typical* cases"

The first step, requirements analysis, has to specify what is understood by *acceptable* diagnosis and by *typical* cases. A possible interpretation of acceptable diagnosis is the ability to explain all the important symptoms and signs encountered in the patient. To define typical case, a subset of all possible combinations of the considered symptoms and signs has to be specified. Assuming these definitions as specifications, the second step checks them in the ES. This can be made either (i) analyzing the KB, inferring the ES function for the typical cases and showing for which typical cases all the important symptoms and signs are explained, or (ii) selecting a sample of typical cases and executing the ES on them. The third step interprets the ES results in terms of the original requirement. Those typical cases for which no acceptable diagnosis has been obtained, are studied to assess (a) the importance of the non explained symptoms, and (b) their degree of typicality. If specification checking has been made by test sample, the degree of sample representativity has also to be considered.

So far, evaluation has been focused almost exclusively on assessing ES performance using testing methods. It is known that high performance is not synonymous of user acceptability for ESs [Buchanan & Shortliffe, 84]. Therefore, other ES characteristics have to be evaluated to achieve a global ES validation. In this sense, the development of metrics assessing ES characteristics numerically can be of significant help. ES characteristics can be quantitative or qualitative, while metrics are always quantitative, so metric results have to be interpreted in the context of the specific ES in order to obtain their real meaning. No one-to-one relation exists between characteristics and metrics, for each characteristic several metrics can be considered.

In the three step process for ES evaluation described above, the use of metrics to assess user requirements is limited to step (ii). Step (i) considers the expression of requirements in terms of metrics, while step (iii) interprets metric results. A variety of ES metrics is needed to assess ES characteristics, since according to [Gasching et al, 83], complex objects like ESs cannot be evaluated by a single number. The final goal of ES metrics is to summarize information to allow for an effective checking of the different types of user requirements, ranging from well-defined structural properties to other aspects of imprecise definition like utility or understandability.

1.4 Testing

We define *ES testing* as the process of examinating ES behavior by its execution on sample cases. According to this definition, ES testing has the same meaning that conventional program testing [Adrion et al, 82]. In practice, ES testing is a very important technique to evaluate ES correctness. Testing has been used as a fundamental criterion for formal acceptance of ESs [Buchanan & Shortliffe 84] [McDermott 81] [Bachant & McDermott 84].

A central issue in testing, shared for both conventional software and ESs, is the selection of the test set. Random, structural and functional approaches have been developed for conventional software. ES random testing can be used in ESs to check robustness, although it seems to be of little value to check correctness, since a random input will probably be meaningless in the ES domain. ES structural testing is performed using test-case generators. ES functional testing relies completely on the selection of test cases by human experts. Historical files on the domain of the ES task are usually taken as source of potential test cases. The quality of the test set depends largely on the extension of these files and on the accuracy of the recorded data.

Specific issues of ES testing are: (i) comparing the ES performance against human expert competence and (ii) judging the adequacy of the deduction path followed, as well as the correctness of the ES output. To assess the ES performance, the presence of human experts as evaluators is needed. This raises new issues, since human experts often disagree and can exhibit problems like prejudice, parochialism and inconsistence. To prevent prejudice in human evaluators, Turing tests have been successfully used [Buchanan & Shortliffe 84]. To measure the consistency degree among several experts some statistical approaches have been proposed. Considering the adequacy of the deduction path followed by the ES, this aspect has been traditionally evaluated in relation to its explanation capability. This aspect has great importance for final ES acceptance, so it has to be considered in isolation..

1.5 The Global Picture

The validation definition given at the beginning of this section establishes that validation has to be determined with respect to the user requirements. We have classified user requirements into totally formalizable and partially formalizable, depending on their total or partial translation into specifications. Verification is the process of checking an ES against its totally formalizable requirements, while evaluation considers ES compliance with respect to its partially formalizable requirements. Combining these definitions, validation appears composed of two parts: verification and evaluation (also called objective and interpretative validation in [Laurent 92]). This separation is induced by the existence of two types of requirements, differentiated by a syntactical criterion. In this way, properties that can be objectively checked (by verification) are differentiated from the other aspects that require subjective assessment (by evaluation)[5]. With respect to testing, it is currently the most important technique for assessing ES performance.

In practice, the difference between totally and partially formalizable requirements, or what is the same, between verification and evaluation processes is not always totally definite. With the exception of domain-independent requirements, that are always perfectly defined in formal terms since they are originated by properties of the knowledge representation language, a few user requirements are totally formalizable. Some requirements are of a clear formalizable nature, but some experimentation is required to obtain the right specifications. For instance, the domain-dependent, totally formalizable requirement explained in section 1.2,

[5] Other authors, like [O'Keefe et al, 87] and [Grogono et al, 92], assign different meanings to these terms. They consider evaluation as the most global term, that includes validation as checking the correctness of ES behavior. These terminological differences are due to the early development stage of the field.

"Typical bacterial pneumonia and atypical pneumonia cannot be considered with high evidence for the same patient"

generates four alternative specifications for a MILORD-based ES,

1.	**Val** (`bact`) ≤ *quite-possible*	and	**Val** (`atip`) ≤ *quite-possible*
2.	**Val** (`bact`) ≤ *very-possible*	and	**Val** (`atip`) ≤ *quite-possible*
3.	**Val** (`bact`) ≤ *quite-possible*	and	**Val** (`atip`) ≤ *very-possible*
4.	**Val** (`bact`) ≤ *very-possible*	and	**Val** (`atip`) ≤ *very-possible*

where the facts `bact` and `atip` represent the concepts of typical bacterial pneumonia and atypical pneumonia respectively, and the **Val** function obtains the certainty value associated to the fact passed as parameter. These specifications are not independent, since some are included in others. Then, what is the right specification set for the previous requirement?. This problem occurred in the verification of PNEUMON-IA. We tested the four specifications, computing the situations in which these specifications were violated. From these situations, we realized that the specification (1) was too restrictive while (4) was too loose. Specifications (2) and (3) reflected properly the desired behavior, so we selected them as the right specification set for the mentioned requirement. This example shows that boundaries between requirements in practice are not as crisp as they can be conceived in theory. Some interpretation is almost always required when checking requirements.

In the mid-term future it seems reasonable that partially formalizable requirements will evolve into a more formalizable ones. This is based on the short history of ES verification. Early verifiers saw ESs as classical logical systems, where only logical issues were tested. Currently, verifiers consider ESs as complex software programs that have to be analyzed taking into account the operational semantics of the knowledge representation language, which is only partially based on logic. Verification issues do not only consider properties of the knowledge representation, but also include properties of knowledge. This evolution will continue in the future, allowing for a more accurate and exhaustive checking of requirements.

Finally, we can view ES validation into the wider framework of software quality assurance (SQA). SQA is concerned not only with assuring the compliance of a program with the user requirements at the present, but also with those aspects that influence how well the software will continue to satisfy the user needs in the future [Rushby 88b]. Software quality is analyzed in [Boehm et al, 78]. A hierarchy of software attributes is proposed in [Adrion et al, 82], including the concepts of reliability, adequacy, testability, understandability, measurability, usability, efficiency, transportability and maintainability.

Very little work has been made on these topics considering ESs [Barrett 90]. It seems reasonable to think that these topics will be addressed in the future for ESs, as a result of the consolidation of ES technology.

1.6 Relation with Software Engineering

Previous definitions for validation, verification, testing and evaluation are in compliance with the essential meaning for these terms in software engineering. Definitions of validation and testing are the same for ESs and for conventional software. The definition of verification for conventional software as "the demonstration of the consistency, completeness and correctness of the software at each stage and between each stage of the development life cycle" [Adrion et al, 82], is based on the notion of specification, that determines the correct pattern at each stage. This definition fits pretty well the one we give for ES verification as the checking the compliance of an ES against the specifications coming from its formalizable requirements. The definition of evaluation in software engineering has always an aspect of subjective assessment. This aspect fits the subjective interpretation of the satisfaction of partially formalizable requirements, involved in the definition of evaluation for ESs.

Regarding validation methods, a major difference exists between software engineering and ESs. In software engineering it is assumed, at least theoretically, that validation can be obtained by verification of successive software development stages [Adrion et al, 82]. This approach does not hold for ESs, because of the impossibility of a complete translation of user requirements into specifications [Hoppe & Meseguer 91].

In summary, previous definitions are in compliance with corresponding terms in software engineering, capturing their essential meanings. The stated difference about the complete achievement of validation by verification, is caused by the type of user requirements for ESs, but this difference is not fundamental. In this sense, the previous definitions fit pretty well in a general framework of software validation, where different kinds of software coexist with a common understanding of the kernel issues.

2 What Should be Validated?

The practical application of ES validation can be seen from two points of view:

1. An ES is composed of several parts (KB, IE, etc.). ES validation consists in the validation of ES components plus the validation the relationships among these components.

2. An ES is the result of a knowledge engineering process with several phases. A set of intermediate results are produced along these phases. The validation of an ES consists in the validation of each phase, by validating the intermediate results produced.

Both approaches are complementary and non-exclusive. Approach (1) is effective in the sense that decomposes the validation of a complex object into the validation of its parts. Approach (2) includes the idea of validation by construction. Validating intermediate results demands the existence of requirements for the knowledge engineering phases. These requirements would be originated from the decomposition of high-level user requirements. Given the early development stage of user requirements in ESs, this decomposition is only partial. In addition, the kind of products generated at each phase is not yet consolidated, as a result of the evolution of knowledge engineering techniques. Nevertheless, validation cannot be delayed until the final ES stage. The cost of correcting an error escalates as the development advances[6], so correcting an error at the very end of the development can be very expensive. Therefore, checking a number of properties enforcing validation on these intermediate results is almost mandatory, in order to assure the effective validation of the final system.

With these ideas in mind, we identifiy the following set of processes and ES parts on which validation has to be present: user requirements, knowledge acquisition, ES architecture, KB structure and contents, inference engine and ES behavior. This list does not aim to be an exhaustive enumeration of all the potential validation aspects, but a set of validation targets common to any ES. Regarding the different procedural elements composing the ES, all of them can be validated using software engineering techniques and they are not analyzed here. As an exception, we consider the inference engine because of its key role in the ES function. In the following, we discuss each one of these validation targets.

2.1 User Requirements

[6] Just as it happens in software engineering.

Given that ES validation is performed against user requirements, it seems quite reasonable to validate them prior to any use, assessing their internal consistency and completeness. Service and competence requirements have to be clearly differentiated, as well as minimum and desirable competence requirements. Totally formalizable requirements have to be completely decomposed into specifications, while for partially formalizable requirements this decomposition is only partial and in some cases tentative. The subjective aspects of partially formalizable requirements have to be clearly identified.

2.2 Knowledge Acquisition

Knowledge acquisition (KA) is defined as the elicitation and analysis of data on expertise with the aim of building an ES [Breuker & Wielinga 87]. KA activities are developed during the initial stages of ES development, aiming to obtain a conceptual picture of the task to be performed by the ES. This picture has to be detailed and complete, since it will be the basis for ES implementation. The elusive nature of human expertise causes many difficulties to capture it, to the extent that KA has been considered as a bottleneck for knowledge engineering (for an analysis of human expertise see [Gaines 87]). Many different KA techniques exists, all of them sharing an intense interaction with a human expert in order to extract his/her expertise. Among these techniques, we can mention focussed interviews, structured interviews, instrospection, self-report, expert-user dialoges and reviews. See [Hart 86] for an overview of these techniques. A number of tools supporting automated KA exist, see [Buchanan et al, 83] [Anjewierden 87] for further details.

Some authors consider knowledge engineering as a modelling activity. In this view, an ES is not a container filled with extracted knowledge, but a model that behaves in an specific way analogous to a system in the real world. To build this model we have to set a mapping from the data extracted from the expert into a given representation formalism. But in addition to this *transformation*, the assessment of data relevance and data structuration has to be made by *abstraction* [Breuker & Wielinga 87]. The knowledge acquisition process is very related to abstraction, since it is a fundamental method to conceptualize and structure complex domains. In this view, an important result of knowledge acquisition is a conceptual model of the intended task for the ES, expressed in some language.

KA is at the basis of knowledge engineering. KA results have a determinant impact in the further stages of ES development. In addition, the conceptual model of the ES task will be used for validation purposes on the implemented ES. Therefore, validation of KA appears as a very important aspect in the global set of validation activities. Validation of KA has two main parts: assuring the quality of KA techniques used in the knowledge engineering process, and checking the adequacy and completeness of the conceptual model conceived. Adequacy is concerned with the correction, grain-size description, homogeneity and integrity of the conceptual model. Completeness considers to which extent the conceptual model captures the whole intended task.

A related point is the acquisition of knowledge for validation, mentioned in section 1.2. It is increasingly apparent that verification of knowledge properties demands more knowledge than the minimum required to perform correct deductions. This extra knowledge has to be obtained in the KA phase, requesting the expert for justifications or causal explanation of his/her behavior. The knowledge for validation has to be included in the conceptual model of the ES, completing and enhancing the conceptual model required for pure deduction.

2.3 Expert System Architecture

An ES is a complex piece of software that contains different parts such as the knowledge base, the inference engine, the user-interface, explanation capabilities, input-output facilities, and others that are application-dependent. ES validation requires the validation of each part in isolation plus the validation of the interrelations among these parts. This problem exists in software engineering when a complex program composed of different modules is validated. Except the KB, all the mentioned ES parts are conventional procedures and their relations can be validated using software engineering techniques. In the following, we will concentrate on the validation of the KB architecture.

A previous step to the validation of KB architecture is the evaluation of the representation capabilities offered by the knowledge representation language with respect to the problem to be solved. If the language does not provide facilities to represent the different aspects of the domain knowledge, the resulting KB is unlikely to be adequate. As an example of this kind of inadequacy, consider a problem dealing with uncertain or imprecise information and a representation language without uncertainty management.

A KB is no longer a bunch of rules without structure. On the contrary, a KB has an internal structure induced by the characteristics of the problem domain. Different types of knowledge are expressed by different elements of the representation. The variety of KB objects (facts, goals, rules, rule sets, metarules and others) and its internal organization are the actual expression of the KB architecture. The KB has to be designed before rule coding, to assure that its organization is adequate for the problem to be solved and not the result of an unordered addition of KB objects. Assuring the adequacy of the KB architecture is a main concern in the validation of the overall ES architecture. The conceptual model of the ES task obtained during the knowledge acquisition phase acts as the basic support for the validation of the KB architecture.

2.4 Knowledge Base Structure and Contents

The KB is a central part of the ES It contains the knowledge of the system coded in some representation language (frequently production rules). The KB is formed by a set of declarations that are interpreted by a fixed procedure, called inference engine, that carries out the operational semantics of the representation language. The KB plays a fundamental role in the ES function since all the actions performed by the system have their origin in the interpretation of the KB contents. Therefore, the validation of KB is a essential step in the global ES validation.

Regarding KB validation, two main aspects exist: validation of KB structure and validation of KB contents. Validation of KB structure is concerned with checking a set of properties of the knowledge representation language on the KB objects. These properties have a double aim. First, to guarantee that no ES malfunctions will exist during the interpretation process performed by the inference engine. And second, to detect those KB objects that can appear as anomalous, although they do not cause any malfunction. So far, validation of KB structure is the most developed subfield of ES validation, with a significant a number of techniques available.

Validation of KB contents considers the adequacy of the KB objects with respect to the knowledge they are supposed to represent. Adequacy includes correctness, consistency and completeness, and it requires checking knowledge properties. Validation of KB contents can be two-fold. First, validation of each single KB object, in the sense that it is a proper representation of a piece of knowledge. This validation is relatively easy to perform, since it implies the analysis of simple elements (typically condition-action pairs). Second, validation of those sets of KB objects that can interact in a ES execution, producing a global effect. The number of these sets is very high (consider for instance the potential number of rule chains in a KB of some hundreds of rules). For this reason, this second aspect of validation is quite difficult in practice.

Validation of KB contents cannot be performed in a complete form without a conceptual model of the knowledge involved in the KB. This conceptual model acts as a framework that supports, integrates and gives sense to the different validation activities that can be performed regarding the KB contents. Without this model, different aspects can be validated in isolation but a global validation of KB contents will not be reached. The role of models in ES validation is analyzed in [Bellman 90]. She mentions that ES task models act as the framework supporting ES design and implementation, so they should be used to support ES validation. Several models can be used for different aspects of the ES task (they are called minimodels). In this sense [Rushby 88b] points out that models bring internal coherence to collections of rules. He considers the explicit construction and scrutiny of models as an essential aspect for trustworthy ESs.

2.5 Inference Engine

The inference engine (IE) is the procedure that interprets the KB objects and executes the actions stated in them according to their operational semantics. Most of the effort in ES validation has been put on the declarative part, i.e. the KB, while the IE was usually assumed correct by default. However, the role of the IE is so important in the ES function that an ES cannot be considered validated without a prior validation of its IE.

Validation of the IE aims at checking that it implements correctly the operational semantics of its knowledge representation language. To perform IE validation, a precise and formal definition of the operational semantics of the considered knowledge representation language is required. This formal definition acts as the specification for the desired behavior of the IE. However, most of the knowledge representation languages do not have a formal definition for their operational semantics. This represents a serious drawback to validate an IE since, without a formal definition, IE validation can only be made by informal means. The absence of formal validation increases the probability of errors in the IE code. In addition, many IEs allows the user to program some IE aspects such as the conflict-set resolution strategy. This means that the user can modify the operational semantics of KB objects. If this occurs, the IE has to be revalidated as well as those KB objects that have been created assuming a different sematics.

The absence of formal definitions for the operational semantics of knowledge representation languages has other consequences for ES validation. Automatic verifiers analyze KB objects considering their operational semantics, in order to predict the ES behavior in a number of situations. If this semantics is not well-defined, the simulation intended by the verifier will be only approximate. This may cause that erroneous situations will be missed, at expenses of the verifier accuracy.

2.6 Expert System Behavior

Validation of the ES behavior consists in assessing to which extent the ES fulfils the user requirements for the intended task. While in previous sections, validation was focused on specific ES parts, now validation considers the ES as a whole. User requirements are again the key point for a true validation. User requirements are clearly application-dependent but they commonly refer to operation conditions, performance level, user interaction, explanation capabilities and acceptance criteria.

Operation conditions establish the requirements for a fully operational ES. Basic hardware and software for the ES, loading time, maximum requirements of time and memory for an execution, etc. are examples of these requirements. For safety-critical applications the operation conditions have to be carefully detailed and exhaustively checked, since a failure in the ES operation may lead to situations very expensive to recover (and this may be caused by a simple syntax error).

The performance level of an ES plays an important role in user requirements, since the ES is expected to perform at a human expert competence level. An accurate assessment of human expert competence is a difficult task that requires statistical measures and consensus functions among experts' opinions. For this reason, performance evaluation is a complex task always requiring human assistance. Performance evaluation can also include the assessment of a lower performance bound, corresponding to the minimal competence requirements described in section 1.1.2.

User interaction involves all the ways in which the user can interact with and use the ES. The communication with the ES is performed through the user interface, that has to be easy-to-use and adaptable to the operational environment. An important point is the quality of ES questions and answers, that should be precise and prevent possible misunderstandings. The ES execution can admit several options: providing data manually or from a file, storing the dialog for later examination, recording the fired rules for archival purposes, etc. A flexible set of execution options can enhance the ES usability. The explanation capability is an aspect of user interaction of particular importance in ESs. Users will not gain confidence in the system without a clear explanation of its reasoning line. This fact conveys a great relevance to the explanation quality.

Acceptance criteria are a number of conditions that the ES must satisfy. In the selection of acceptance criteria, we have to consider that no technique provides a complete validation. On the contrary, partial validation evidence is obtained from a variety of different techniques with complemented effects. Therefore, the acceptance criteria have to involve a representative combination of validation techniques focusing on different ES parts. The critical conditions for the ES function have also to be present in this selection.

3 How to Validate?

A set of techniques for ES validation have been developed in the last years. This set is far from being complete, since there are many validation aspects without an appropriate answer. However, these techniques can validate significant portions of current ESs and they can be of great help for knowledge engineers. Validation methods developed for software engineering can be imported into knowledge engineering, after their adaptation to ES peculiarities. AI techniques, and specially the ones coming from machine learning like KB refinement, can also be of great help for ES validation. In the following, the validation aspects considered in section 2 are revisited, identifying some of the available techniques that are suitable for them.

3.1 User Requirements

No specific technique has been developed to validate user requirements for ESs. This problem exists in software engineering, so it seems reasonable to adapt software engineering techniques to the ES case. Reviews involving end-users, human experts and knowledge engineers can be the basic element for validation of these requirements.

3.2 Knowledge Acquisition

Little work has been devoted to develop validation techniques for KA. The work of [Benbasat & Dhaliwal, 89] provides an initial framework for KA validation. They define validation of KA as the degree of homomorphism between the representation system, i.e. the ES, and the system that it is supposed to represent, i.e. the expert. They differentiate three stages in the development of a KB: conceptual KB, elicited KB and implemented KB. They consider four types of validation: conceptual, elicitation, implementation and representational. Conceptual validation considers the quality of the modelling process that has generated the conceptual KB from the human expert. Elicitation validation addresses the completeness and correctness of the process of translating the conceptual KB into the elicited KB. Implementation validation considers the quality of the process that transforms the elicited KB into the implemented KB. Representational validation aims at matching the characteristics of the implemented ES with the human expert. For each validation type, they give a number of tests to assess different aspects with respect to the intended homomorphism. Regarding conceptual and elicitation validation, they suggest as specific validation techniques different forms of KB inspections and structured walkthroughs, involving source experts, independent experts, knowledge engineers and final users.

The ES development methodology determines, to a great extent, the style of the KA process and the kind of validation we can perform on it. Broadly speaking, we can differentiate two types of ES development methodologies: bottom-up and top-down. A bottom-up methodology develops an implemented system as soon as some domain data are structured and understood. Rapid prototyping is the paradigm of bottom-up methodologies. In this approach, the conceptual structuration and abstraction of the data extracted from the expert is relatively low and a global conceptual model is missing. KA validation is limited to assuring the quality of KA techniques, such as expert interviews, and an adequate implementation of the recorded knowledge. This second aspect is not easy to fulfil, given that available techniques such as KB inspections are of difficult applicability when a significant amount of knowledge is involved.

Top-down methodologies, like KADS methodology [Breuker & Wielinga 87], include the development of a conceptual model of the intended task prior to any implementation. This conceptual model is expressed in a modelling language, that provides a vocabulary in which the expertise can be described in a coherent way. In this approach, validation can be performed not only assuring the quality of expert interviews, but also and what is more important, on the conceptual model. Getting a true validation of the conceptual model would mean an step of immense value in knowledge engineering. In this case, the most ellusive validation issues, those involving deep knowledge aspects that are not properly captured by its representation, would be solved. Some preliminary results on validation of acquired knowledge using specific tools, as well as on validation of conceptual models can be found in [Shadbolt 91].

3.3 Expert System Architecture

As stated in section 2.3, validation of ES architecture can be done for all the ES components (except the KB) by using standard techniques of software engineering. Validating the internal KB architecture involves experts and knowledge engineers. Experts have to evaluate how well the KB design fits the overall structure of the problem domain. Knowledge engineers act as interfaces between experts and specific constructs of the KB.

The conceptual model of the ES task is an important element to support this kind of validation. This model acts as a reference point for both experts and knowledge engineers. Without a conceptual model, differences of vocabulary and points of view between experts and knowledge engineers may cause a superficial validation of the KB architecture. In this case, undetected errors will appear later in the ES development, with a higher correction cost.

The expressive capacity of KB objects has a significant impact in this kind of validation. Advanced knowledge architectures, like generic tasks [Chandrasekaran 87] or component of expertise [Steels 90], provide high level constructs that are closer to the conceptual model expression and make easier its scrutiny and validation. On the other hand, modularity features in the knowledge representation language facilitate a correct KB design and allow to define some hierarchical relationships among KB objects that can be exploited for validation purposes [Sierra et al, 91].

3.4 Knowledge Base Structure and Contents

Validation of KB structure is performed by checking those requirements related with the knowledge representation language used. These requirements are usually formalizable and domain-independent, so validation of the KB structure is totally achieved by verification. Validating the KB structure demands exhaustive checking, only reachable by automatic verifiers. A significant number of verifiers is currently available for different ES models, to the extent that this is the most developed subfield of ES validation. A verifier of the KB structure has to analyze KB objects considering their operational semantics [Evertsz 91].

Validation of KB contents implies checking knowledge properties, to assess the knowledge correctness, consistency and completeness. The set of knowledge properties to be tested has to convey a significant evidence about the adequacy of the KB contents. Selection of the knowledge properties has to be supported by the conceptual model of the intended ES task. Otherwise, only isolated knowledge properties will be tested without achieving an integrated revision of the encoded knowledge. The existence of a conceptual model is specially relevant when checking the KB completeness, in the sense that all the knowledge required for the intended task is contained in the KB. In the elicitation of the knowledge properties to be tested, new aspects of the knowledge contained in the KB are usually required. These aspects have not been explicited because they are not required for deduction, but they are needed for validation purposes. These new aspects have to be encoded in auxiliary representations (like integrity constraints). Techniques used in automatic verifiers can be adapted to check those knowledge properties that assess the validity degree of KB contents.

Validation of individual KB objects can be made by inspection of the KB using experts that are independent from ES developers. It provides evidence on the correctness of single KB objects, but does not give any proof of the correctness of the KB as a whole. Validation of sets of interacting KB objects always requires computer-supported tools, since manual checking is unfeasible due to the large number of sets involved. Inconsistency is a good example of this kind of validation, that has been extensibly considered in the literature. Inconsistency is a serious error that demands important computational efforts for effective checking. On the other hand, procedures for checking inconsistency are relatively simple. Integrity constraints declaring those facts that are incompatible are the only extra knowledge required for inconsistency checking. Facilities for inconsistency checking are currently available in most of the existing automatic verifiers.

Validation of KB contents can be obtained by means of ES testing. Validation through testing is indirect, partial and incomplete. Validation is indirect because it can only be induced from the results of ES executions. Validation is partial because there is no guarantee that all the KB components have been used in the testing executions. Validation is incomplete because no specific knowledge properties are checked in ES executions, except for the functionality of some KB objects in some test cases. In spite of these drawbacks, testing is the most frequently used method to assess the validity of KB contents. This is due to the following causes: (i) testing is much easier and direct than using complex verifiers for knowledge properties (verifiers that have to be built specifically for a shell), (ii) conceptual models supporting the validation of KB contents are infrequently built, and (iii) developers have an strong inclination to validate KB contents by ES execution. This inclination is reinforced by the usual lack of written requirements for the ES, so the only way to check the correctness of its constituting parts is by comparing ES outputs with the opinion of human experts.

Validation of KB contents can be made using KB refinement techniques. KB refinement aims at improving the KB contents from a set of cases with known solutions. When some cases are treated incorrectly by the ES, the KB is modified to achieve a correct treatment in all cases. It is assumed that only minor KB changes are required. KB refinement is founded on ES testing, so it shares the testing drawbacks. However, automatic refinement tools able to detect errors and to suggest substantiated changes in the KB, represent a significant help for the practical assessment of KB validity.

3.5 Inference Engine

An IE is a conventional interpreter of a knowledge representation language. It can be validated using standard techniques of software engineering. Knowledge representation languages do not often provide a formal definition of their operational semantics (see section 2.5). This is an important difficulty to achieve a true IE validation. Facilities to link user programs to the IE also difficult its validation.

As any other interpreter, the IE has to be defined in terms of a set of elementary operations that actually implement its functionality. Interpreters of standard programming languages are defined using memory cells, stacks, assignments, increments, conditionals and branchings. Elementary operations for an IE can be condition satisfaction, logical operators, truth value assignment and others. No consolidated set of elementary operations to deal with knowledge representation languages exists. This lack is an extra difficulty since, prior to any IE validation, the set of elementary operations has to be defined. Methods and techniques used to validate interpreters of standard programming languages could be adapted for IE validation.

3.6 Expert System Behavior

Validation of ES behavior is usually made by testing, although in case of critical requirements other techniques such as exhaustive verification can be used. The testing process has to allow for an effective checking of user requirements. Simulated test cases can be used to check specific conditions, while real test cases are needed to assess ES performance. Test set composition and comparison with human behavior are two of the key issues in ES testing.

Two testing techniques have an special relevance in ESs, Turing tests and field tests. A Turing test consists in the evaluation of the ES output mixed with recommendations of human experts for a set of given cases. A set of independent experts acts as evaluators without knowing which is the system and who are the humans. This test is very suitable for performance evaluation. A field test consists in the regular use of the ES in its target working environment. ES errors and users complains are recorded and solved. The ES is considered tested when user complains have ceased. This test is suitable for assessing aspects of the user interaction, overall utility, and regular performance. Both types of testing require important amounts of human effort.

4 When to Validate?

Paraphrasing [Adrion et al, 82], knowledge engineering is an exercise in problem solving. As with any problem-solving activity, determination of the validity of the solution is part of the process. Therefore, validation has to be explicitely included in knowledge engineering, that will not be considered finished until the produced ES has been validated.

Validation cannot be delayed until the final phases of ES development. At this stage, ES errors could be very expensive to correct. As we have stated in section 2, the cost of correction of a hidden error escalates as the development advances. Therefore, early validation is always recommended. A number of validation activities can be performed during the ES development. The position of the validation activities in the ES life-cycle is discussed in the following.

4.1 Validation in the Life-Cycle

We propose an ES life-cycle composed of the following steps: requirements, knowledge acquisition, design, implementation and maintenance. The first four steps fit exactly with the first four steps of the development cycles proposed by [Buchanan et al, 83] or [Miller 89]. We add a fifth step, maintenance, that includes all the operations made on the ES after its release. Maintenance operations can be frequent in some applications (it has been said

Life-cycle step	**Step Products**	**Validation Activity**
Requirements	Service/Competence Totally/Partially Formalizable Specifications	Validation of User Requirements
Knowledge Acquisition Acquisition	Conceptual Model	Validation of Knowledge
Design	ES architecture	Validation of ES architecture
Implementation	Operational ES	Validation of KB Struct & Contents Validation of Procedural Parts Validation of ES Behavior
Maintenance	New operational ES	Any of the previous

Figure 1. Proposed ES life-cycle and corresponding validation activities.

that a KB is never complete because new knowledge is always ready to be added[7]). Therefore, this step is needed for a comprehensive treatment of the different stages in the ES life-cycle. No specific step devoted to testing or evaluation exists; validation activities are included in each development step. The whole process is represented in figure 1.

We want to stress the importance of validation activities in the ES life-cycle in a two-fold way. First, the products originated in each development step are validated by specific activities. This enhances the quality of intermediate products and increases the validity of the final ES. Second, the presence of validation in each step reinforces the design for validation. If knowledge engineers are aware that validation is a part of their job, they will acquire knowledge for validation and they will include facilities for validation in the design and implementation phases. The final product should be *valid* and this has to be kept in mind throughout the ES development.

We notice that the validation of an ES is more that the validation of its KB. If in the implementation step procedural parts are developed, they should be verified following standard techniques of software engineering. In particular, this holds for the inference engine that should be validated, at least informally, if it is not certified. Changes in the maintenance phase can affect to any previous development step. To validate these changes, the validation activities corresponding to the affected steps should be repeated.

The proposed life-cycle does not imply necessarily a waterfall or rapid prototyping methodology. It can be used to develop a prototype, that can be later refined and expanded to achieve a final system, like [Miller 89]. In this case, all the life-cycle steps have to be repeated (requirements, knowledge acquisition, etc.), although the implementation step will not start from scratch but from the previously developed prototype.

5 Concluding Remarks

This paper contains several ideas about the meaning of validation for ESs and on the forms to achieve it in practice. Some of these ideas are supported by the experience, others are

[7] We share this opinion. After the release of PNEUMON-IA, a new etiological diagnosis called *Chlamidia Pneumoniae* was identified. To include this new knowledge, we have reorganized an existing module and we have created a new one.

proposals to improve the current state of validation. In the following, the main ideas exposed in this chapter are summarized:

- *Validation Terminology*: we define validation in terms of user requirements, just like in software engineering. User requirements for ESs are totally or partially formalizable, depending on their complete or partial decomposition in specifications. Validation is composed of two main parts, verification and evaluation, depending on the type of user requirements respectively checked. Verification aims at objectively checking ES properties, while evaluation considers those aspects that require subjective assessment. Testing is viewed as the main technique to assess ES performance. These definitions are in compliance with the corresponding ones in software engineering, forming a general framework of software validation.

- *Complete Validation*: KB validation is not synonymous for ES validation. To achieve a complete validation, all the ES parts have to be validated. Procedural parts can be validated using standard software engineering techniques. Special emphasis has to be devoted to the inference engine, because of its central role in the ES function. To validate declarative parts specific techniques for ESs are required.

- *Validation throughout the ES life-cycle*: an ES life-cycle is proposed, including maintenance as a separate step. Validation, as an element of knowledge engineering, has to be present in all the steps of the ES development. . Different validation activities can be performed at each step. This presence in the whole life cycle enforces validation by construction, improving the quality of the intermediate products. In addition, it stresses the design and implementation for validation.

- *Knowledge Validation*: all the knowledge extracted from the expert has to be validated. To effectively perform this validation, the knowledge has to be organized forming a conceptual model. This model has to contain all the dependencies and relations among the intended task elements, as well as the heuristics and processes used by the expert to perform this task. The accuracy and completeness of this model is a crucial point, since it acts as the supporting framework to validate the ES implementation. In order to validate knowledge properties, some extra knowledge is required. This extra knowledge is not used for deduction, it is just for validation. It is acquired and represented in the usual ways, and it is also subject to validation. Early validation activities in the ES life-cycle enforce the acquisition of this kind of knowledge.

Acknowledgements

I thank Ramón López de Mántaras, Enric Plaza and Carlos Sierra, who have read a previous version of this paper providing many useful comments. I also thank Jean Pierre Laurent for early access of his work on validation terminology.

References

Adrion W.R., Branstad M.A., Cherniavsky J.C. (1982). Validation, Verification and Testing of Computer Software. *Computing Surveys*, **14**(2), 159-192.

Anjewierden A. (1987). Knowledge Acquisition Tools. *AI Communications*, 0 (1) 29-38.

Bachant J., McDermott J. (1984). R1 Revisited: Four Years in the Trenches. *AI Magazine* **5**(3) 21-32.

Barrett B. W. (1990). A Software Quality Specification Methodology for Knowledge-Based Systems. *Workshop on Knowledge-Based Systems: Verification, Validation and Testing*, *AAAI'90*.

Batarekh A. Preece A., Bennett A., Grogono P. (1991) Specifying an expert system. *Expert Systems with Applications*, **2**(3).

Bellman C. L. (1990). The Modelling Issues Inherent in Testing and Evaluating Knowledge-based Systems. *Expert Systems with Applications*, **1**(3), 199-216.

Benbasat I., Dhaliwal J.S. (1989). A Framework for the Validation of Knowledge Acquisition. *Knowledge Acquisition*, **1**(2), 215-233.

Boehm B. W., Brown J.R., Kaspar H., Lipow M., MacLeod G.J., Merrit M.J. (1978). *Characteristics of Software Quality*. North-Holland.

Breuker J. Wielinga B. (1988). Models of Expertise in Knowledge Acquisition. In *Topics In Expert Systems Design: methodologies and tools*, Guida & Tasso eds., North Holland.

Buchanan B. G., Barstow D., Bechtel R., Bennett J., Clancey W., Kulikowski C., Mitchell T. and Waterman D. (1983). Constructing an expert system. In *Building Expert Systems*, Hayes-Roth F. D., Waterman A. and Lenat D. B. eds, 127-168. Addison-Wesley.

Buchanan B.G., Shortliffe E.H. (1984). *Rule-Based Expert Systems, The MYCIN Experiments of the Stanford Heuristic Programming Project.* Buchanan and Shortliffe eds. Addison-Wesley.

Chandrasekaran B. (1987). Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1183-1192.

Evertsz R. (1991). The Automatic Analysis of Rule-based System Based on their Procedural Semantics. *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (*IJCAI'91*), 22-27.

Gaines B.R. (1987). An overview of knowledge-acquisition and transfer. *International Journal of Man-Machine Studies* **26**, 453-472.

Gaschnig J., Klahr P., Pople H., Shortliffe E., Terry A. (1983). Evaluation of Expert Systems: Issues and case Studies. In *Building Expert Systems*, Hayes-Roth, Waterman, Lenat eds., Addison Wesley.

Grogono P., Batarekh A., Preece A., Shinghal R., Suen C. (1992). Expert System Evaluation Techniques: A Selected Bibliography. *Expert Systems*, to appear.

Hamilton D., Kelley K., Culbert C. (1991). State-of-the-Practice in Knowledge-based System Verification and Validation. *Expert Systems with Applications*, **3**, 403-410.

Hart A. (1986). *Knowledge Acquisition for Expert Systems*. Kogan Page Ltd.

Hoppe T., Meseguer P. (1991). On the Terminology of VVT. *Proceedings of the European Workshop on Verification, Validation and Testing of KBS* (*EUROVAV'91*), 3-14.

Krause P., O'Neil M., Glowinski A. (1991). Can we Formally Specify a Medical Decision Support System?. *Proceedings of the European Workshop on Verification, Validation and Testing of KBS* (*EUROVAV'91*), 247-258.

Laurent J. P. (1992) Proposals for a valid terminology in KBS validation. To appear in ECAI'92 conference.

McDermott J. (1981). R1's formative years. *AI Magazine* 2(2), 21-29.

McDermott J. (1988). A Taxonomy of Problem Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*, ed. S. Marcus, 225-256, Kluvier.

Meseguer P. (1992). *Validation of Multi-Level Rule-Based Expert Systems*. PhD dissertation, Universitat Politecnica de Catalunya.

Miller L. A. (1989). A Comprehensive Approach to the Verification and Validation of Knowledge-Based Systems. *Workshop on Verification, Validation and Testing of KBS*, *IJCAI'89*.

Newell A. (1982). The Knowledge Level. *Artificial Intelligence*, 18, 87-127.

O'Keefe R.M., Balci O., Smith E.P. (1987). Validating Expert System Performance. *IEEE Expert*, Winter 87, 81-89.

Rushby J. (1988a). Validation and Testing of Knowledge-Based Systems. How Bad Can It Get?. *Workshop on Knowledge-Based Systems: Verification, Validation and Testing*, *AAAI'88*.

Rushby J. (1988b). *Quality Measures and Assurance for AI Software*. SRI-CSL-88-7R, SRI Project 4616.

Shadbolt N. (1991). Building Valid Knowledge Bases: An ACKnowledge Perspective. *Proceedings of the European Workshop on Verification, Validation and Testing of KBS* (*EUROVAV'91*), 195-210.

Sierra C., Agustí-Cullell J., Plaza E. (1991). Verification by Construction in MILORD. Proceedings of the *European Workshop on Verification and Validation of Knowledge-Based Systems* (*EUROVAV'91*), 211-226.

Slage J.R., Gardiner D.A., Han K. (1990). Knowledge Specification of an Expert System. *IEEE Expert*, August 90, 29-38.

Soloway E., Bachant J., Jensen K. (1987). Assessing the Maintainability of XCON-in-RIME: Coping with the Problem of a Very Large Rule Base. *Proceedings of the Sixth National Conference on Artificial Intelligence* (*AAAI'87*), 824-829.

Steels L. (1990). Components of Expertise. *AI Magazine*.11(2) 28-49.

Verdaguer A. (1989). *PNEUMON-IA: Desenvolupament i validacio d'un sistema expert d'ajuda al diagnostic medic*. PhD dissertation. Universitat Autonoma de Barcelona.