Formal Methods in the development of safety critical knowledge-based components

Giovanna Dondossola
ENEL-SRI, Department of Electrical and Automation Research, Electronic Technologies for Automation,
Via Volta 1, Cologno Monzese 20093 Milan, Italy
E-mail: dondossola@pea.enel.it

Abstract The work reported in this paper is part of the ongoing Esprit project Safe-KBS n. 22360[*]. A main objective of the project is the definition of a engineering methodology for certifiable knowledge-based software components to be embedded into safety critical systems. Since about a decade the use of formal methods in the development of traditional software for safety critical systems has been greatly encouraged. On the other hand, research works in the Knowledge Engineering field are proposing new formal methods as a means to increase the quality of KB software products and processes. Therefore it seems quite natural to propose a pervasive use of formal methods from the early stages of the development as a vehicle to promote the acceptance of KB software in safety critical application domains. The subject of this paper concerns both the role of formal methods in the Safe-KBS engineering methodology and the experimentation of their application based on a general-purpose formal method called TRIO. The specification and V&V features of TRIO will be analysed and judged with respect to the requirements coming from the safety critical KB software.

Keywords: knowledge based components, safety critical software, formal methods, temporal logic, object-oriented concepts, specification, verification, validation, certification, life cycle, methodology.

Introduction

Nowadays the advantages of formal methods[1] for the development and certification of safety-critical software are widely recognised by both Software Engineering community and standard organisations. Some safety critical domains specific standards explicitly either highly recommend **[RTCA 92], [IEC 1508 95]** or mandatory require **[MOD 91]** the use of formal methods. The specific benefit recognised to formal methods is that they allow "complex behaviour" to be analysed (by means of proofs or state exploration), reviewed, and analysed in their totality, rather than merely sampled as by testing or simulation. Thus, the major benefit derives from a double application of formal methods that is by formal requirements specification coupled with formal verification. From the certification point of view, formal methods are recognised to increase the degree of confidence in achieving software of high integrity levels **[Rushby 95]**.

A number of industrial level applications of the most consolidated formal methods (such as VDM and Z) already exists **[Bowen 97]**. However, the state of the art about formal methods & safety critical systems essentially refers to software developed in a traditional way[2].

According to the terminology introduced in **[SafeKBS R3.2-a]** a *knowledge-based component (KBC)* is a technologically homogeneous part of a software system based on knowledge based technology, whilst a *knowledge-based system (KBS)* is a software system that includes one or more KBC and, eventually, other traditional software components. A KBC includes a *knowledge base*, containing an explicit representation of the knowledge relevant to some specific competence domain, and a *reasoning mechanism* that can exploit such knowledge in order to provide high-level problem-solving performance. An *empty system* is a KBC with an empty knowledge base.

KBC are a relatively recent type of software (sub-) systems

- whose development processes refer to specific life cycles and methodologies,
- whose specification, verification, validation and implementation activities sometimes require different, specifically knowledge-based methods, techniques and technologies.

Looking at the specification activity, it has been observed in **[Vermesan 95]** that requirements of KBC are typically vague, and that it is generally hard to specify in advance a KBS completely because of the incremental nature of the knowledge elicitation process. Actually specification is a typical incremental activity both in conventional and KB software. A complete and correct specification, both formal and informal, can never be given in advance but it is surely the result of an iterative process of adjusting, completing and improving of some intermediate specification whose number of iterations depends on the complexity of the system. On the other hands "vagueness" in the requirements specification is something risky in a safety critical context, especially if it is a source of artificial non-deterministic behaviours, and it should be avoided whenever possible. In other words, vagueness should not be intended as a synonymous of imprecision, but on the contrary vague requirements should result into abstract and precise specifications. Summarising, it could be affirmed that

- the possibility (really the need) to describe the system at different abstraction levels, independent from low level design/implementation details and efficiency issues, without loosing the clarity and precision essential to the safety

[*] The Safe-KBS project is partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 22360. The partners in the Safe-KBS project are Sextant-Avionique, Det Norske Veritas, Enel-Sri, Tecnatom, Computas Expert Systems, Uninfo, Qualience. This paper reflects the opinions of the author and not necessarily those of the consortium.

[1] The phrase "formal method" is used to mean a language whose semantics is described by using a mathematical theory. Therefore a formal method is always equipped with a formal reasoning calculus.

[2] See the World Wide Web Virtual Library on Safety-Critical Systems at URL http://www.comlab.ox.ac.uk/archive/safety.html.

aspects of the KBC
- the incremental style of the specification activity
- the flexibility and exhaustivity of the analysis methods of the specification

are typical features of formal specification methods facing well the nature of KB requirements.
The advantages of formal methods (FM) in the development of KB software are beginning to be recognised by the Knowledge Engineering (KE) research community **[Aben 95], [van Harmelen 95], [Meseguer 96]**. Recent works on FM in KE are moving towards two directions, namely the development of new KB-specific FM and the application of existing FM coming from the traditional Software Engineering (SE) field. Comparative analyses of KB vs. traditional FM can be found in **[Fensel 94], [Fensel 95], [Aben 95].**
Given the maturity level of FM and KB technologies, a need arises for a KE methodology establishing the role and the scope of the FM application in safety critical and KB contexts, that is the primary aim of the present work. The Safe-KBS engineering methodology shall represent a support to the work of both knowledge engineers (specifiers, designers, programmers) and certification authorities in charge of approving safety critical systems embedding KBC.
The second contribution of the present work concerns the application of a highly declarative SE formal method termed TRIO (Tempo Reale ImplicitO) to develop safety critical KB software. TRIO is a language equipped with a method for the specification, verification and validation of industrial systems **[Ciapessoni 97].** Historically, it was thought first for the specification of real time systems whose time dimension is relevant. Since about ten years, the TRIO language has been applied to the development of several industrial projects in the energy field within the long-term joint research carried on by CISE, ENEL-Research and Politecnico di Milano. A global evaluation of the TRIO method shall be provided in terms of a classification table by merging the FM classification factors proposed in **[Fensel 95]** and **[Aben 95]**.
The proposed Safe-KBS engineering methodology and the specification features of the TRIO language have been exercised on test cases taken from a KBS of the Safe-KBS project in the safety critical avionics domain. The application is termed FINDER **[Safe-KBS R1.1]** and consists of a support system for aircraft pilots aiding them in detecting possible problems that may occur during the flight and in formulating alternative routes to circumvent the problem.

The contents of the paper are organised as follows. Section 1 presents the role of FM in the Safe-KBS engineering methodology, including the definitions of the main concepts used. Section 2 describes the TRIO-specific languages, techniques and tools relating them to the FM-generic work pieces of the Safe-KBS methodology. Section 3 classifies the TRIO method in terms of FM-related factors and compares it with other emerging FM. Section 4 describes the some experiences from the application of the methodology and TRIO. The conclusions try to evaluate the initial results of the experimentation activity.

1. Role of formal methods in the Safe-KBS engineering methodology

A first high-level description of the Safe-KBS methodology, together with the definitions of the ontological concepts on which it is based, has been presented in **[Safe-KBS R3.2-a]**. It describes the life cycle in a way which abstracts from the use of FM. The idea is that the impact of FM on the life cycle can be expressed as a sort of specialisation and refinement of that description of the methodology. Therefore the first two sub-sections shall report the main ontological concepts and a summary of the life cycle structure defined in **[Safe-KBS R3.2-a]**, whist the third sub-section shall describe the impact of FM on that structure.

1.1 Safe-KBS life cycle ontology

- A *KBC life cycle* is a co-ordinated set of processes necessary to produce and exploit a KBC. A KBC life cycle has a temporal evolution that covers the entire life of a KBC. The processes of a KBC life cycle, although not independent, are assumed to be executed in parallel. All span, at least in principle, over the entire life of a KBC.
- A *KBC life cycle process* is a unique logical organisation of activities necessary to achieve a specific issue relevant to the production and exploitation of a KBC. The concept of issue is used here in a generic meaning; it denotes an objective to be achieved, a facet to be taken into account, a high level requirement to be satisfied. The issue is what characterises a process; in a sense, what keeps together a set of activities to form a unique, coherent process. The concept of process is intuitively bound to that of a specific, dedicated agent in charge of executing it, called the *process executor*. Different processes correspond, at least in principle, to different executors. A process executor includes all the resources needed to execute a process, namely a working group supplied by the necessary tools and infrastructure.
- A KBC life cycle specified up to the level of processes is called a *KBC life cycle model*. Thus, in a KBC life cycle model, processes are defined in detail, while activities are only given distinctive names and specified at the level of inputs and outputs. The KBC life cycle model is given a precise semantics by means of a unique *finite state automaton*, describing the execution mechanism of a generic activity in terms a set of input/output relations. Thus the logical organisation of the whole life cycle is obtained by applying the automaton to instances of the input/output relations associated to the activities.
- An *activity* is a logically and temporally confined component of a KBC process, constituted by a logical

organisation of tasks. The definition of an activity includes: the specification of activity inputs/outputs, the declaration of a set of tasks, the specification of possible constraints to meet in the execution of tasks (optional). According to the definition given, an activity is not characterised by a defined execution schema, but allows for a variety of different execution schema to be adopted. This is a crucial point for allowing effective tailoring of the methodology to specific application contexts and industrial practices.

- A KBC process specified up to the level of activities is called a *KBC process plan*. Thus, in a KBC process plan, activities are defined in detail, while tasks are only given distinctive names.
- A KBC life cycle in which each process is specified up to the level of activities is called a *KBC life cycle plan*. A KBS life cycle plan can therefore be viewed as the collection of all the KBC process plans relevant to the KBC life cycle processes.
- A *task* is a coarse-grained work unit to be carried out in order to achieve a significant advancement in the execution of an activity. A task is constituted by a logical organisation of actions. The definition of a task includes: the declaration of a set of actions, the specification of possible constraints to meet in the execution of actions.
- An *action* is a fine-grained, elementary work unit. The concept of action is considered as atomic and is specified only in intuitive terms. The possible internal organisation of an action is not relevant to the life cycle concept.
- A *support package* is an integrated collection of methods, techniques and tools devoted to guide and help the execution of an activity, a task, or an action. The concepts of methods, techniques and tools are understood here in intuitive terms. However, the following distinction can be made. A *method* is a general and abstract specification of a conceptual procedure to follow to meet stated goals. A *technique* is a specific and detailed specification of a practical procedure to follow to meet stated goals; techniques are used to implement methods. A *tool* is a computer-based system that can help in the application of a method or a technique. Clearly, support packages may be defined at a variable level of detail: from a mere literature reference to a fine-grained description of an industrial practice.
- A KBC life cycle specified up to the level of activities or tasks, together with a non-empty collection of support packages relevant to its activities, tasks or actions is called a *KBC methodology*. Note that concepts of KBC life cycle model and of KBC process plan are limited to the specification of "what" should be done in order to produce and exploit a KBC, while the concept of methodology explicitly deals with "how" the specified activities, tasks, and actions should be actually executed.
- A KBC methodology that includes a collection of support packages specifically oriented towards a specified issue, is called a *dedicated KBC methodology*. For example, dedicated KBC methodologies may be oriented towards quality, productivity, safety, etc.

1.2 The Safe-KBS life cycle

The Safe-KBS life cycle is an embedded KBS life cycle that is a software life cycle relevant to the development of a knowledge-based software component. More specifically, it will be a dedicated KBC methodology, oriented towards the issue of safety. It shall be developed into three steps.

Initially, the Safe-KBS life cycle shall be developed at the level of processes, thus constituting a KBC life cycle model. At this level, the Safe-KBS life cycle will be as far as possible general and independent of any specific KB technology (such as, for example, forward rule production rules, causal nets, event graphs, etc.). Of course it will not be independent, however, from the basic features of the KB technology (the existence of an empty system and a separate knowledge base, the conceptual and logical levels of system design, etc.) and from the fundamental issue of safety that is considered as the cornerstone of the whole methodological framework. This first step, as developed in **[Safe-KBS R3.2-a]**, shall be summarised in the following of the present section.

Later, the Safe-KBS life cycle shall be developed further at the level of activities, thus providing, for some KBC process, a detailed definition of the relevant plan. Also process plans will be mostly of general nature. This second step has been developed in **[Safe-KBS 3.2-b]** as far as the development process is concerned, and is going to be developed for the V&V process.

At the third step the Safe-KBS life cycle shall be detailed at the level of KBC methodology (for some life cycle processes). The task level will be developed as appropriate and necessary in order to achieve a fully operative definition of the life cycle and to support a correct link with the relevant support packages. This level of development of Safe-KBS life cycle will be independent of any specific KB technology and will focus only on those methods, techniques and tools generally applicable in the KBS field. The methodological development step is still in progress and the impact of FM here presented prepares the root for the integration of formal and non-formal V&V methods and techniques which is ongoing at the methodological layer. Of course even the evaluation of TRIO as a candidate FM for KBC development belongs to this methodological step.

The Safe-KBS life cycle model includes seven *processes*, whose identification is based on a concept of process which

- is intended to capture an important dimension of the life cycle, according to the requirements of KB and safety critical software;

- is understood as a collection of activities and tasks that share a common issue;
- explicitly refers to a specific, distinct agent in charge of executing it.

The Safe-KBS life cycle processes are the following: Development, Verification and Validation, Safety Management, Quality Management, Certification, Configuration Management, Project Management. From a formal perspective, all the seven processes defined above are understood at the same hierarchical level. No process has a leading role with respect to the others: all processes interact through input/output relationships. In practice, however, the development process has a primary function in the life cycle: this will be apparent both from the complexity of its internal structure and from the high number of relationships that link it to all other life cycle processes. All the seven processes defined above span over the entire life cycle, from the start instant to the withdrawal instant. Due to the aim of this work the activity decomposition is reported for the Development and V&V processes only.

The *development* process defines the activities necessary for requirements specification, design, construction, release, operation, maintenance and retirement of a KBC. It is responsible for the realisation of the main products of the life cycle to be reviewed, approved and managed by the other processes of the life cycle according to their specific competence. It is decomposed into twelve activities, namely

D1. definition of detailed development process plan: D1 includes two main tasks: tailoring of the methodology to the case at hand and defining the expected duration of activities and their due dates. D1 produces in output the detailed development plan. D1 is also concerned with the revision of the detailed development plan, if this turns out to be necessary according to later revealed project management decisions.

D2. requirements specification: D2 deals with the specification of the functional, technical and operational requirements of the KBC. It takes in input the specification of inherited requirements from the system level. D2 produces in output the requirements specification. Whenever appropriate, the use of a formal specification language supporting D2 is greatly encouraged.

D3. conceptual design: the main goal of conceptual design is to analyse domain and problem-solving knowledge and to develop a formal, abstract, conceptual representation, called the *conceptual model*. A conceptual model describes both the types and organisation of domain knowledge (concepts, attributes, relations, constraints, operations, events, etc.) and the ways it is used to solve problems (problem types considered, problem-solving strategies, etc.). The conceptual model serves two specific purposes:

- it is the starting point used by the designers for the construction of the logical model of the KBC;
- it is the basic tool used by the knowledge engineers during the construction of the knowledge base, for interpreting and coding domain knowledge.

D4. logical design: the main goal of logical design is to focus on the technical choices necessary to implement the KBC and to develop a complete and detailed technical description of the system, called the *logical model*. Logical design includes:

- the architecture, the knowledge representation techniques, and the reasoning methods adopted to design and implement the KBC;
- the choice of the most appropriate development tools;
- the detailed technical design of the KBC.

D5. construction of the empty system: D5 includes two distinct cases: either the empty system is bought from the market and tailored to the specific application at hand, or it is built from scratch for the application considered. D5 takes in input the logical design. D5 produces in output the running empty system.

D6. construction of the knowledge base: D6 is concerned with the various tasks necessary to develop the knowledge base of the KBC. Basically, it includes the following tasks: knowledge elicitation and modelling, knowledge coding and integration and checking, knowledge base checking testing and refinement.

D7. final verification and refinement: D7 is concerned with the final verification and refinement of the complete KBC with test cases proposed by the production team. Note that D7 should not be confused with different activities of the verification and validation process.

D8. integration-related activity: D8 is aimed at supporting the integration of the KBC in the software system. It is concerned with KBC level tasks that might be necessary for integration. Integration is managed and carried out at system level. D8 also includes field verification, validation and refinement tasks.

D9. release-related activity: D9 is aimed at supporting the release of the software system as far as the KBC is concerned. It is concerned with KBC level tasks that might be necessary for release. Release is managed and carried out at system level. D9 also includes acceptance and refinement tasks, writing of manuals, and training of the users.

D10. operation-related activity: D10 is concerned with all KBC level tasks relevant to the operation of the software system. In particular, it includes the following tasks: collection of user reports, analysis of KBC history, definition of maintenance requests.

D11. definition of a maintenance intervention: D11 analyses the outputs of D10 and defines the goals, scope, and constraints of a maintenance intervention.

D12. retirement-related activities: D12 is aimed at supporting the retirement of the software system as far as the KBC is concerned. It is concerned with KBC level tasks that might be necessary for retirement. Retirement is managed and carried out at system level.

The *verification and validation (V&V)* process defines the activities necessary for determining (i) whether the product of an activity of the development process fulfils the stated requirements (verification), and (ii) whether the product of an activity of the development process fulfils its specific intended use (validation). The distinction between verification and validation stems from two facts:

- in general, one is not allowed to assume that the requirements specifications initially stated for a KBC are a complete, consistent and correct expression of its specific intended use, i.e. of the real needs of the individuals and of the organisation for which the product is developed;
- it is not possible, in general, to prove the equivalence of the different intermediate products of the life cycle with their stated requirements.

Since our focus here is on KBC life cycle, we can assume that the original requirements specification includes two components: one inherited from the system level through a formalised document and another derived at component level directly from the analysis of the needs of the individuals and of the organisation for which the KBC is developed. We call these two components the *inherited specifications* and the *derived specifications*, respectively.

Verification applies to any product (i.e., activity output) for which a requirements specification exists. Therefore, in general, all the products of the development activities may be subject to verification except the requirements specification, for which no specification exists since it is intended as the first initial specification of functional, technical and operational requirements at KBC level. For each development activity the stated requirements are expressed in the requirements specification plus the requirements coming out by the immediately precedent activity.

Validation applies to any product (i.e., activity output) for which an assessment against the needs directly expressed by the individuals/organisation for which the KBC is developed is possible and meaningful.

The V&V process is decomposed into nine activities, namely:

V1. definition of detailed verification and validation plan: V1 concerns tailoring and management tasks of the V&V process. V1 takes in input safety requirements and project resources and produces in output a preliminary version of the detailed V&V plan, to be submitted for revision to the safety, quality, project management and certification processes. V1 also handles later requests of revision coming from the project management process.

V2. validation of requirements specification: V2 concerns

- the traceability of system level requirements into component level requirements
- the evaluation of requirements specification for correctness, consistency, completeness, understandability, accuracy, feasibility, testability.

V3. verification of conceptual design: V3 concerns

- the traceability of stated software requirements into conceptual design
- the evaluation of the conceptual model which can be inspected by domain experts without the burner of implementation details.

V4. verification of logical design: V4 concerns

- the traceability of KBC requirements and conceptual design into logical design
- the verification of the logical design
- the elaboration of the software test plan to be executed in the verification of the implemented KBC.

V5. verification and validation of the empty system: V5 concernes

- the traceability of KBC requirements and logical design into the empty system
- the V&V of the empty system.

V6. verification and validation of the knowledge base: V6 concerns

- the traceability of KBC requirements and conceptual design into the knowledge base
- the verification of the knowledge base supported by two (types) of techniques, namely inspection and structural verification. Inspection techniques **[Meseguer 96]** aim at detecting semantically incorrect knowledge in the KB and are performed manually by a human who has expertise in the application domain (but independent of experts involved in the KB construction). Structural verification techniques analyse the syntactic properties of the knowledge base checking for internal coherence ; they support the automatic verification of types of anomalies such as redundant, conflicting or missing knowledge
- the verification of the KB against an explicit knowledge model (structural, functional, behavioural, causal, fault, geometrical, associative).

V7. verification and validation of the KBC: V7 concerns

- the traceability of KBC requirements into the implemented KBC
- the empirical testing **[Meseguer 96]** of the KBC which aims at checking its correctness by executing the KBC on a finite set of test data. The selection of the test set is crucial to the effectiveness of the testing process
- the evaluation of the KBC behaviour against the human expert and/or other sources of knowledge: its major goal is

to assure that the KBC complies with the semantics of the real world by making use of an implicit knowledge model.

V8. verification and validation of the integrated KBC and **V9. verification and validation of the released KBC** concern

- the traceability of integration(release) requirements (expressed at system level) into the integrated(released) KBC
- the coding and execution of tests
- the empirical evaluation **[Meseguer 96]** of the integrated(released) KBC which addresses the relation between the operational KBC and the final user. Typical evaluation issues are technical performance, acceptability, inclusion in the organisation, responsibility issues.

As stressed by its activity decomposition, the V&V process is tightly coupled with the development process. In particular for each activity of the production and release phases of the development process - except the activity "definition of detailed development plan" - there exists a correspondent activity of the verification and validation process. The V&V process activities analyse and review the development products and the results of such review provide the criteria for transiting from one activity of the Development process to the following one. The development process can include specific activities or tasks dedicated to verification or validation issues. Verification and validation, however, is intended as a separate process, in charge of a different team who carries out specific activities on the products of the development process. It does not overlap with the activities of the development process, but rather it pursues different issues. While Development is committed to produce good products (and, therefore, it must include some specific activities or tasks dedicated to verification or validation), V&V is responsible for determining whether these products fulfils the stated requirements (verification), or their specific intended use (validation).

In the following section, the impact of FM on the first two steps of the life cycle development is described. As far as the third development step is concerned, it inherits all the methodological aspects associated to TRIO languages and tools. Therefore it shall be treated in Section 2 dealing with TRIO.

1.3 Impact of formal methods on the Safe-KBS life cycle

It is widely recognised that it not suitable (neither profitable) to specify all the requirements of a safety-critical complex application in a formal way. On the other hands, it is indubitably true that a subset of very critical functions of such an application can greatly benefit from a formal approach to their requirements specification. In his recommendations for the certification of safety critical systems, Rushby suggests that the use of formal methods should be limited to "those aspects of design that are least well covered by present techniques. These arise in redundancy management, partitioning, and the synchronisation and co-ordination of distributed components, and primarily concern fault tolerance, timing, concurrency, and nondeterminism." **[Rushby 95]**.

By restricting the scope of our considerations to the KB software technology, it could be observed that

- KB software exhibits all the complex design aspects cited by Rushby. In particular KBC contribute to the complexity of a system with their decision procedures and uncertainty handling
- KBS complexity is not satisfactorily dominated by currently available KBS V&V techniques **[Meseguer 96]**.

The analysis of the results of the safety analysis conducted on the Safe-KBS applications reveals that most of times safety requirements are expressed in terms of V&V requirements on KBS functions concerning either the inference steps or time constraints of the reasoning process. Sometimes safety requirements add new functionality to existing KBS functions.

Given the above considerations, the impact of FM on the Safe-KBS life cycle is based on the following assumptions:

- the phrase *safety critical* KB software has to be intended in its restricted meaning, i.e. to refer KB software used to implement a function or component characterised by the highest level of integrity. Therefore the term safety critical KB software should not be used as a synonym of the term *safety related* KB software, which instead refers KB software used to implement functions or components of any safety integrity level
- not all the KB software of an application considered safety critical are actually *safety critical*. Given a KBS in the safety critical context, its safety analysis reveals that in general only part of its KB software is actually safety critical, the others being only safety related (but non safety critical) or neither that
- using any FM at its maximum potentiality is not enough to guarantee the software integrity level required by safety critical KB software. In other words, the definite role of FM application in the development of safety critical KB software is to increase the degree of confidence in achieving failure rates on the order required by the catastrophic and hazardous levels
- V&V techniques provided by FM should not replace, but supplement the existing KBS V&V practise.

The above Safe-KBS assumptions are interpreted by the following certification-oriented prescriptions which shall be used in the tailoring of the development plan to the case at hand (activity D1.)

- the use of a FM is *mandatory required* for the development of safety critical KBC software. FM are *only recommended* for safety related (non safety critical) KBC software
- by merging the safety and KBS complexity assumptions, FM shall be applied to this requirements categorisation:
  - functional and time-related requirements of existing KBC functions/tasks which are safety-related

- functional requirements of KBC functionality newly introduced by the safety analysis
- requirements about the dynamic of the KBC reasoning process

- the specific most appropriate FM for the given KBC shall be selected among a set of reliable FM made available from both the SE and KE communities
- the *highest degree of rigour* in the application of the selected FM is proposed for safety critical KB software, i.e. the maximum potentiality of the FM shall be used. Specification, validation and verification activities shall make use of the FM and its supporting tools. Lower levels of rigour in the application of the selected FM shall be used for safety related (but non safety critical) KB software
- formal specification and V&V techniques shall be integrated with informal KBS techniques
- as the degree of confidence in achieving high integrity levels can not be measured in practise, others more sound techniques, based on different technologies providing robust forms of diversity, should be applied in conjunction with KB software developed using FM to obtain more certain probabilities.

The integration of FM in the current state of development of the Safe-KBS life cycle implies:

- the selection of the processes and activities of the life cycle model which FM are relevant for
- the specialisation of those activities as required by formal methods
- the refinement of the development tasks into actions as appropriate to support a correct link with the methods, techniques and tools related to formal methods
- the definition of the methods, techniques and tools related to FM and their association to the relevant activities, tasks and actions.

FM mainly concern four processes of the Safe-KBS life cycle model, namely Development, V&V, Safety Management, Certification, with a high impact on the Development and V&V. FM also impacts on some system level activities on which the Safe-KBS life cycle model is based. It should be remembered that the Safe-KBS life cycle has been conceived as an embedded KBS life cycle whose activities interact with a set of system level activities. The system level activities belong to the external life cycle which is relevant to the development of the global KBS which the KBC is part of. The system level impact of FM involves those activities concerning the KBS specification and validation.

Let us start from describing the impact of FM on the Development process. Several Development activities have a specialised way to be performed that greatly benefits of the existence of a formalised specification of requirements and FM supporting tools. Some tasks of these activities are going to be reformulated and decomposed into FM specific actions, if necessary. The following amendments shall be added to the above activities formulations.

D1. definition of detailed development process plan: *both the tailoring of the methodology and the activities/tasks scheduling depend on the scope of application of FM and in turn by the safety requirements specific of the case at hand. The set of certification-oriented prescriptions shall guide the definition of the FM scope together with its level of rigour.*

D2. requirements specification: *the selected FM is applied to formally specify the requirements of those KBC functions identified in D1. Formal specification is an iterative process consisting in the revision and refinement of a (possibly incomplete) statement of the specification by means of formal V&V techniques, such as model generation and history checking. The iteration concludes only when the specification becomes complete and correct. A unique document on requirements specification shall be provided as a final product of this activity, presenting in an integrated way the informal requirements specification endowed with their (possibly partial) formalisation.*

D3. conceptual design: *being the crucial element of a KBC specification, (the relevant parts of) the conceptual model shall be formalised in the selected FM.*

D7. final verification and refinement: *the availability of formal specifications constitutes a valid support to the verification of the safety critical parts of the implemented KBC. Specifically formal specifications can help greatly in the selection of the input data for the KBC final testing.*

By interpreting the supplemental role of formal V&V techniques with respect to KBS V&V practise, the below amendments to V&V activities aim at capturing a twofold contribution provided by FM, that is in terms of:

- new V&V techniques (formal proofs) to be applied on specification-oriented life cycle products, such as requirements, conceptual model, logical model
- extended versions of existing V&V techniques for KBS (such as inspection, structural verification, empirical testing) to be applied on implementation-oriented life cycle products, such as empty system, KB, released KBC.

V1. definition of detailed verification and validation plan: *the use of the selected FM at the specified level of rigour radically influences V&V activities and the resources needed.*

V2. validation of requirements specification: *formal V&V techniques supported by the selected FM shall be used to validate the preliminary version of requirements specification provided by the Development team.*

V3. verification of conceptual design: *formal V&V techniques supported by the selected FM shall be used to verify the preliminary version of the conceptual design specification provided by the Development team. If the formal conceptual model is to be verified according to structural properties (such as consistency and redundancy), then these properties need to be defined in terms of the selected FM. FM provide additional properties to be checked in* *structural verification. For example, the modular architecture and the declaration of hierarchies of types provided by languages like $(ML)^2$ and $TRIO^{+*}$ permit to check the violation of modularity and type mis-matches. The specification of KBC conceptual design in the selected FM allows establishing the link between (formally specified) KBC requirements and their corresponding conceptualisation.*

V4. verification of logical design: *KBC requirements/conceptual design specified in the FM shall support traceability whenever the logical design is expressed in a format compatible with that FM. Formal specifications shall support the elaboration of the software test plan for the KBC testing by helping in the selection of the test set. On one hand a correct formal specification provides the intended I/O behaviour of the KBC, therefore it contains all the information needed to perform functional testing. On the other hand, the level of structure in formal specifications can be used to support structural testing.*

V5. verification and validation of the empty system: *KBC requirements/conceptual design specified in a FM shall support traceability whenever the empty system is implemented in a programming language compatible with that FM*

V6. verification and validation of the knowledge base: *KBC requirements/conceptual design specified in a FM shall support traceability whenever the knowledge base is implemented in a knowledge representation formalism compatible with that FM. Furthermore FM shall support techniques to verify the implemented knowledge base against an explicit, formal conceptual model. As a formal conceptual model describes how the KBC I/O behaviour can be achieved, it provides, to some extent, elements of the KBC structure. After implementation, these elements can be checked to verify whether their functionality meets their specification. Inspection and structural verification techniques can still be employed to detect structural discrepancies between specification levels and implementation.*

V7. verification and validation of the KBC: *the high degree of structure provided by a formal (KBC requirements/conceptual design) specification can be used manually to check for the detection of corresponding structures in the implemented KBC. Tests coding and execution greatly benefit from the existence of formal specifications given the possibility to synthesise test data automatically (see activity V4.).*

2. The TRIO methodological support

TRIO techniques and tools are divided here into three support packages, in order to allow a correct link with the Safe-KBS activities and tasks summarised in the previous section.

2.1 TRIO specification method

The development tasks “first statement of requirements formal specification” and “revision and refinement of requirements formal specification” could be supported by the TRIO specification support package that includes:

- a basic logic language referred to as Basic TRIO for specifying in the small **[Bertani 96]**, whose semantics is suitable to express system behaviours and quantitative time constraints typical of real time and reactive systems
- an object-oriented extension referred to as $TRIO^{+*}$ for specifying in the large **[Ciapessoni 95]**, which supports writing modular reusable specifications of complex systems
- a further ontological extension **[Ciapessoni 95]**, which includes predefined higher level, application-oriented notions such as events, states, processes, pre- and post-conditions.

Basic TRIO is a first order temporal logic language that supports a linear notion of time: the *Time Domain* is a numeric set equipped with a total order relation and the usual arithmetic relations and operators (it can be the set of integer, rational, or real numbers, or any interval thereof). TRIO formulae are constructed in the classical inductive way, starting from terms and atomic formulas. Besides the usual prepositional operators and the quantifiers, one may compose TRIO formulae by using a single basic modal operator, called *Dist*, that relates the *current time*, which is left implicit in the formula, to another time instant: the formula *Dist(F, t)*, where *F* is a formula and *t* a term indicating a time distance, specifies that *F* holds at a time instant at *t* time units from the current instant. For convenience, TRIO items (*variables, predicates,* and *functions*) are distinguished into time-independent (TI) ones, i.e., those whose value does not change during system evolution (e.g., the altitude of a reservoir) and time-dependent (TD) ones, i.e., those whose value may change during system evolution (e.g., the water level inside a reservoir).

Several *derived temporal operators* can be defined from the basic *Dist* operator through prepositional composition and first order quantification on variables representing a time distance. A sample list of such operators is given in the table reported below, with a short definition and explanation.

Operator	**Definition**	**Explanation**
Futr(A, d)	$d>0 \wedge Dist(A, d)$	A holds d time units in the future
Past(A, d)	$d>0 \wedge Dist(A, -d)$	A holds d time units in the past
AlwF(A)	$\forall t\ (t > 0 \rightarrow Dist(A, t))$	A will always hold
Alw(A)	$\forall d\ Dist(A, d)$	A always holds
SomF(A)	$\exists d\ (d>0 \wedge Dist\ (A, d\ ))$	A holds sometimes in the future
Som(A)	$\exists d\ Dist\ (A, d\ )$	Sometimes A holds
Lasts(A, d)	$\forall d'(0<d'<d \rightarrow Dist(A, d'))$	A will hold over a period of length d
Lasted(A, d)	$\forall d'(0<d'<d \rightarrow Past(A, d'))$	A held over a period of length d in the past
WithinF (A,d)	$\exists t\ (0<t<d \wedge Dist(A, t)\ )$	A will happen within d time units in the future

A sample of TRIO derived temporal operators

TRIO^{+*} enriches basic TRIO with concepts and constructs from object-oriented methodology. Among the most important added features are the ability to partition the universe of objects into classes, to introduce inheritance relations among classes, and to exploit mechanisms such as genericity to support the reuse of specification modules and their incremental development. TRIO^{+*} is also endowed with a graphic representation of classes in terms of boxes, lines, and connections to depict class instances and their components, information exchange, and logical equivalence among (parts of) objects. Classes denote collections of objects that satisfy a set of axioms. They can be either *simple* or *structured* –the latter term denoting classes obtained by composing simpler ones. A simple class is defined through a set of axioms premised by a declaration of all items that are referred therein. Some of such items are in the *interface* of the class, i.e., they may be referenced from outside it in the context of a complex class that includes a module that belongs to that class.

On the basis of the first experiences in the application of TRIO^{+*} to real-life industrial projects, the language was further enriched by means of so-called *ontological constructs*, which support the natural tendency to describe industrial systems in a more operational way, i.e., in terms of states, transitions, events, etc. An *event* is a particular predicate that is supposed to model instantaneous conditions such as a change of state or the occurrence of an external stimulus. Events can be associated with *conditions* that are related causally or temporally with them. From the causal viewpoint, a condition for an event can be *necessary* or *sufficient*; from the temporal viewpoint, conditions are divided into *preconditions*, that refer to the time instants preceding the event, and *postconditions*, that refer to the instants following it. A *state* is a predicate representing a property of a system. A state may have a duration over a time interval; changes of state may be associated with suitable predefined events and conditions. Altogether, events, states, and conditions define a comprehensive model of the system evolution. As an additional refinement mechanism, TRIO^{+*} provides the notion of *process*. A system entity can be viewed, at a certain level of abstraction, as an event or a state, therefore as an atomic item having no further structure. A more precise or detailed system modelisation may however view such entities as combinations of other events, states, and conditions that are temporally or causally related to the original, more abstract notion of event or state.

The concepts, the syntax and the specification technique of the extended TRIO formalism are supported by its Graphical/Textual Editor**.** A detailed description of the features of this tool, whose new version will be released in 1998, can be found in its specification document **[Bertani 97]**.

2.2 TRIO validation method

The development task “revision and refinement of requirements formal specification” and the V&V activities could be supported by the TRIO validation support package consisting of a set of validation techniques, classified as follows:

- *model generation:* given a TRIO specification, sample models can be derived semi-automatically or in a fully automated way, providing examples of how a system complying with the given specification should behave **[Ciapessoni 94]**. Model generation provides a method to verify the coherence and the adequacy of a specification. The model generation technique can be used to make different types of analysis on TRIO specifications:
  - *truth proof*: to research models which verify the specification. In one sense also the TRIO formulae represent models for the specifications, but it is not easy to understand the possible behaviours of specified systems looking at them. The Model Generator produces interpretations of TRIO formulae using a representation instant/event (the event is the truth value of a predicate in a time instant). It produces all, or a subset of, the possible models of the specification, i.e. it generates possible behaviours of the system
  - *falsehood proof*: to research the models which get false the specification
  - *satisfiability proof*: to establish if the specification has at least a model, i.e. if it doesn't contain contradictions
  - *unsatisfiability proof*: to prove that a specification doesn't have any model
  - *validity proof*: to verify if the specification is a tautology, i.e. in every time instant it admits the empty model (if the specification admits the empty model, then every model is a model for it; for instance 'A | ~A' admits the empty model because 'true A' is a model and 'false A' is a model too)

  - *completion proof*: to establish if a TRIO formula is in contradiction with an history and to find the possible completions of the history w.r.t the formula. Moreover this functionality can be used to verify that a given partial behaviour corresponds to an interpretation where the specification formula is evaluated to true
  - *property proof*: to establish (by refutation) if a given property is a logical consequence of the specification and a given history
- *history checking:* a given system behaviour is compared with the specification for compatibility. In a typical application of this technique the designers describe a set of expected behaviour of the system along with some illegal behaviours and checks whether the former are compatible with the specification while the latter are not.

The techniques described above are actually complementary and in principle could be integrated in a unique validation method/tool. Currently they are implemented as two separate tools, namely the Model Generator and the History Checker **[Mandrioli 95]**.

2.3 TRIO verification method

The development task “KBC final testing”, and the V&V activities “verification of logical design” and “V&V of the KBC” could be supported by the TRIO support package for the *test case generation***.**

Once a system has been implemented, it must undergo a verification phase in order to assess its correctness. The most used technique employed to verify a software product is testing. However, the most critical point is to select the input data that will be used to test the software in an appropriate way. Since a TRIO specification is also a description of the functionality of the system, it is quite straightforward to use the specification to select the input data to be used as test cases. Moreover, the executability of a TRIO specification makes possible to compute the expected output, and thus to compare it to the output produced by the actual system.

The TRIO approach to the construction of a test suite consists in the definition of appropriate *filtering* and *strategy* mechanisms to allow the generation (that can be automatic or interactive) of a set of models relevant to test the implementation. The problem of generating only relevant models has been studied in **[Mandrioli 95]**. The main idea presented there consists in dividing the problem into two sequential steps: the interactive generation and composition of *partial test cases*. Such an idea is supported by the Test Case Generator.

3. Comparison of TRIO with others FM

A classification of the TRIO method is herein proposed in terms of a set of evaluation dimensions, which represents the union of those dimensions proposed in **[Aben 95]** and **[Fensel 95]**. A brief explanation of some dimensions anticipates the classification table and some TRIO-related comments. The selection of FM to be included in the table was based on a homogeneous orientation criterion that is only model-oriented methods have been included.

A first group of dimensions concerns generic FM features, such as SE vs. KE origins (native field). The domain spectrum defines the scope of application of the FM with respect to the application area (narrow vs. broad). The method spectrum defines the scope of application o the FM with respect to the activities of the software engineering process (narrow vs. broad). Orientation is defined as the FM viewpoint which may be model-oriented, property oriented and behaviour oriented. The main mathematical basis are algebra, process algebra, logic and set theory.

The second group concerns criteria characterising the purpose of a FM. Horizontal structuring refers here to the composition of a formal specification from a number of smaller specifications

The third group concerns features of the conceptual model underlying a FM. The last two are about the representation of dynamics in terms of the characterisation of a state and its transitions, and the definition of control. Non-constructive and constructive means to specify control over state transitions are distinguished. A non-constructive specification of control defines constraints for legal control flows. They exclude possible control flows but do not define actual ones. Constructive specifications of control flow define directly the actual control flow of a system and apply a variant of the closed-world assumptions. Two variants can be distinguished for the constructive definition of control: globally by procedural languages and locally by rules.

	VDM	Z	TRIO	DESIRE	KARL	$(ML)^2$
Native Field	SE	SE	SE	KE	KE	KE
Domain spectrum	broad	broad	broad	Narrow	narrow	narrow
Method spectrum	broad	narrow	broad	broad	broad	narrow
Orientation	model	model	model	model	model	model
Mathematical basis	logic	Set theory	logic	logic	logic	logic
Specification type	functional & dynamics	Functiona l	functional & dynamics	Dynamics	dynamics	dynamics
Conceptual model	no	no	no	Yes (own)	yes (KADS)	yes (KADS)
Horizontal structuring	no	yes	yes	yes	yes	yes
Proof obligations	yes	yes	yes	no	no	no
Refinement calculus	yes	no[a]	no	no	no	no[b]
Prototyping	no	no	no	yes	yes	no

Terminological knowledge	poor	poor	poor[c]	poor	rich	poor
Generic inferences	yes	yes	yes	yes	yes	yes
Object/meta level	no	no	no	yes	limited	yes
Notion of state	record of dynamic variables	state schemas	state predicates	Termination states of modules	fixed set of dynamic logic variables	fixed set of dynamic variables
Control	constructive & global		non-constructive	Constructive & local	constructive & global	constructive & global

a. Only discussed in [Wood 93].

b. A refinement calculus related to $(ML)^2$ has been defined in [Aben 95].

c. The terminological level of TRIO is classified as poor due to the lack of a pre-defined conceptual model. However, thanks to its temporal and process-oriented constructs TRIO is much more expressive than all FM based on order sorted first order logic.

From the comparison of TRIO with some KBS-purpose FM it emerges that TRIO nature shares more with $(ML)^2$ than others. They both offer the same support for the terminological knowledge modelling. Even though $(ML)^2$ has been developed having the KADS expertise model clear in mind, the primitives it provides are strictly based on a logic-oriented ontology (it makes use of sorts, functions, predicates, logical operators and quantifiers). TRIO and $(ML)^2$ shares the idea that expressive power is of primary concern in specification languages with respect to executability. They both are based on a model-theoretical semantics of predicate logic whose execution supports the specification evaluation by automatic generation of formal proofs (they both do not support prototyping). Given their high level of expressivity, they both have to face with the efficiency issues arisen by the automatic proofs on real-sized specifications.

Differently from $(ML)^2$, and due to its origins in traditional software, TRIO provides a functional specification of the entire system which is not based on any pre-identified (informal) conceptual model. However, even though the KBC conceptual model underlying TRIO has never been stated explicitly, for sure it could be derived from the conceptual level of its semantics and could resemble to the Object-oriented Modelling Technique OMT from SE **[Rumbaugh 91]**. The general-purpose conceptual model underlying TRIO can be used to formalise special-purpose conceptual models, such as those defined in the KE field. This is exactly one aim of the present work. A typical KE conceptual model distinguishes static domain knowledge from dynamic control knowledge. Generic knowledge-level control over the use of functionally specified sub-steps during the reasoning process can be obtained in TRIO by either combining its basic logical, temporal and process operators, or defining new operators in terms of them.

Again differently from $(ML)^2$ (and partially similarly to KARL) TRIO integrates object-oriented and process-oriented constructs into a logical framework which does not distinguish the object(domain)-level from the meta(domain)-level. However this is considered a reasonable trade-off to accept when efficiency of automatic proofs is pursued as the second aim (together expressivity) substantiating the existence of TRIO.

4. Experiences from the application of the Safe-KBS methodology and TRIO to a safety-critical KBC

4.1 Brief description of the FINDER application

The mission of a commercial aircraft is to transport passengers or fret safely and economically from a departure airport to a destination airport. To do this the crew prepares a flight plan composed of a list of navigation points (called waypoints) from the departure airport to the destination airport. Additional information describes the vertical profile (cruise altitude and speed, descent profile, etc). The number of waypoints depends on the distance to cover and on the complexity of the airspace to overfly. This flight plan is agreed with Air Traffic Control before departure.

FINDER is a knowledge based prototype of a future on-board decision-aid system which would assist the crew with respect to the flight plan management. Today this management task is fully supported by the crew.

This task includes acquisition and interpretation of contextual data, situation analysis (impact of an event on flight plan), implementation of a flight plan modification strategy (search for alternatives, evaluation of alternatives, selection and execution of an alternative). When performing this task there is an important demand of cognitive resources sometimes combined with a high level of stress. FINDER helps analysing the situation and proposes solutions. However FINDER is not supposed to make decision in place of the crew or to give orders to the crew. To be executed these solutions have to be agreed by ATC and validated by the crew. To reach FINDER objective four main functions have been identified, namely Monitor, Replan, Manage dialogue and Negotiate.

4.2 Description of the experimentation with TRIO

FINDER functions shall be implemented by a software architecture composed by a set of technologically heterogeneous software modules **[Safe-KBS R1.4]**. According to such technological design, FINDER can be considered a KBS composed of six KBC performing all event monitoring and several flight replanning sub-tasks. Following the Safe-KBS prescriptions introduced in section 1.3, the minimal scope of FM in the FINDER KBS is given by the safety critical parts of its KBC. By considering the SIL produced by the Preliminary System Safety Analysis, it follows that FM should be mandatory applied at least in the development of five KBC. The KBC performing event monitoring and situation detecting has been chosen to present the peculiar aspects of the Safe-KBS methodology and TRIO. Such a KBC is characterised by five safety critical sub-components performing the following sub-tasks: Monitor airport weather, Monitor enroute weather, Monitor trajectory, Monitor airports adequation, Assess situation.

It should be noted that most of the Safety Parameters identified by the Functional Failure Analysis of FINDER **[Safe-KBS R1.3]** express requests towards the V&V process and about the reasoning knowledge. For instance, they explicit the need to *check the correctness* of the *logic* used to detect alarming situations (hazardous airport weather, bad enroute weather, restricted air space), the *extrapolation of data* used to detect alarming situations, the *algorithm* used to detect alarming situations.
In the experimentation, temporal and object-oriented constructs have been used to structure the functional, temporal and performance requirements, whilst the high level conceptual design is obtained as a specialisation of formal requirements. The abstraction and ontological constructs have been combined to describe the conceptual design at two abstraction levels: events that are viewed as atomic at the first level become structured processes at the second level.

Conclusions

The very final objective motivating the work presented in this paper is to stress the need of using formal methods in the development of knowledge-based software components embedded into safety critical applications. The application of formal methods thus represents a good chance to have the KB technology accepted both by designers/developers of safety critical software and by Certification Authorities in charge of approving software systems embedding KBC. At this aim the basic Safe-KBS methodology has been defined which makes explicit the role of formal methods by means of a disciplined schemata.
The TRIO formal method has been experimented on a safety critical KBS in the avionics field at the aim to evaluate it as a candidate support package of the Safe-KBS methodology. The TRIO experimentation consisted in use of the TRIO language to formalise some critical parts of the application. It is the base on which TRIO formal proofs shall be exercised in order to demonstrate that the specification actually fulfils the requirements stated by the safety analysers.

References

M. Aben, *Formal Methods in Knowledge Engineering*, PhD dissertation, University of Amsterdam, February 1995.

A. Bertani, E. Ciapessoni, *TRIO Model Generator: User Manual*, Deliverable D3.1 of ARTS Trial Application of the ESPRIT project n. 20695, 1996.

A. Bertani, E. Ciapessoni, *Specifica dei requisiti per l' adeguamento dell' Editor Grafico di TRIO+* (TGE)*, CISE/ENEL Reasearch Report (in Italian), 1997.

J.P. Bowen, M.G. Hinchey, *The Use of Industrial-Strength Formal Methods*, 1997.

E. Ciapessoni, E. Corsetti, E. Crivelli, M. Migliorati, *Checking satisfiability of TRIO specifications*, in Proceedings of the Workshop on temporal logic associated to the ITLC 94, Bonn, Germany 1994.

E. Ciapessoni, D. Mandrioli, A. Morzenti, P. San Pietro, *Manuale di TRIO+**, ENEL Research Report, (in Italian) November 1995.

E. Ciapessoni, A. Coen-Porosini, E. Crivelli, D. Mandrioli, P. Mirandola, A. Morzenti, *From formal models to formally-based methods: an industrial experience*, accepted to Transaction on Software Engineering and Methodologies, 1997.

E. Ciapessoni, A. Coen-Porosini, *TRIO: an environment supporting the verification/validation of real-time systems requirements*, paper unpublished, 1997.

D. Fensel, F. van Harmelen, *A comparison of languages which operazionalise and formalise KADS model of expertise*, Knowledge Engineering Review, Vol. 9 , 105-146, 1994.

D. Fensel, *Formal Specification Languages in Knowledge and Software Engineering*, Knowledge Engineering Review, Vol. 10, No. 4, December 1995.

IEC 1508 95 IEC, International Electrotechnical Commission, Draft International Standard 1508: *Functional Safety: Safety-Related Systems*, Geneva, Switzerland, 1995.

D. Mandrioli, S. Morasca, A. Morzenti, *Generating Test Cases for Real-Time Systems from Logic Specifications*, ACM Transactions on Computer Systems, Vol. 13, No. 4, pg. 365-398, November 1995.

P. Meseguer, A.D. Preece, *Assessing the Role of Formal Specifications in Verification and Validation of Knowledge-Based Systems*, Proc 3rd IFIP International Conference on "Achieving Quality in Software" (AQuIS'96), pg. 317-328, Chapman and Hall, 1996.

UK Ministry of Defence: *Interim Defence Standard 00-55: The procurement of safety critical software in defence equipment*, Part 1, Issue 1: Requirements; Part 2, Issue 1: Guidance, April 1991.

A. Morzenti, D. Mandrioli, C. Ghezzi, *A Model Parametric Real-Time Logic*, ACM Transactions on Programming Language and Systems, 14, 4, pg. 521-573, October 1992.

Safe-KBS R1.1, Esprit Programme Project No. 22360, Task 1.1 report: *Software Functional Requirements*, 1996.

Safe-KBS R1.3, Esprit Programme Project No. 22360, Task 1.3 report: *Functional Failure Analysis*, 1997.

Safe-KBS R1.4, Esprit Programme Project No. 22360, Task 1.4 report: *Preliminary System Safety Analysis Folder*, 1997.

Safe-KBS R3.2-a, Esprit Programme Project No. 22360, Task 3.2 report: *Safe-KBS life-cycle model description document*, 1997.

Safe-KBS R3.2-b, Esprit Programme Project No. 22360, Task 3.2 report: *First Safe-KBS development process plan*, 1997.

RTCA/Eurocae DO178-B/ED-12B: *Software considerations in airborne systems and equipment certification*, Requirements and Technical Concepts for Aviation, Washington, DC, December 1992. This document is known as EUROCAE ED-12B in Europe.

J. Rushby, *Formal Methods and the Certification of Critical Systems*, SRI Technical Report CSL-93-7, December 1993 (300 pages).

J. Rushby, *Formal Methods and their Role in the Certification of Critical Systems*, SRI Technical Report CSL-95-1, March 1995 (300 pages). This is a shorter (50 pages) and less technical treatment of the material in **[Rushby 93]**. It will become a chapter in the FAA Digital Systems Validation Handbook (a guide to assist FAA Certification Specialists with advanced technology issues).

F. van Harmelen, J. Balder, *$(ML)^2$: A formal language for KADS models of expertise*, Knowledge Acquisition, 4 127-161, 1992.

F. van Harmelen, D. Fensel, *Formal Methods in Knowledge Engineering*, Knowledge Engineering Review, Vol. 10, No. 4, December 1995.

A.I. Vermesan, T. Bench-Capon, Techniques for the Verification and Validation of Knowledge-Based Systems: A Survey Based on the Symbol/Knowledge Level Distinction, Software Testing, Verification and Reliability, Vol. 5, 233-271 (1995) John Wiley & Sons, Inc.