A Model-Based Approach to the Verification of Program Supervision Systems

Mar Marcos[1] †, Sabine Moisan ‡ and Angel P. del Pobil †

† *Universitat Jaume I, Dept. of Computer Science*
Campus de Penyeta Roja, E-12071 Castellón, Spain
Tel.: (+34) 64 34 56 42
Fax: (+34) 64 34 58 48
e-mail: {marcos|pobil}@inf.uji.es

‡ *INRIA Sophia Antipolis*
BP 93, F-06902 Sophia Antipolis Cedex, France
Tel.: (+33) 4 93 65 78 47
Fax: (+33) 4 93 65 79 39
e-mail: Sabine.Moisan@sophia.inria.fr

Abstract This paper is concerned with the verification of knowledge-based systems for a task called *program supervision* (PS). Our aim is to highlight how the model on which a particular PS system is based influences the properties that the knowledge base must fulfil. Adopting this point of view allows us to go beyond verification techniques that do not exploit the intended use of the embodied knowledge. We distinguish the following elements in the specification of a knowledge-based system: task definition, problem-solving method, and domain model. Our verification approach is based on the PS ontology and on the requirements that both the PS task and problem-solving method enforce on the domain knowledge of a specific application. The definition of the verification issues for the knowledge base is done in a way dependent both on the PS task and the particular PS problem solving method, but independently of the knowledge representation scheme or the specific application. This work in the modelling and property determination for the PS task has as additional outcomes a categorisation of the properties and some guidelines to determine the adequacy of the expert's knowledge with respect to the requirements of the PS problem-solving method. We also intend to rely on this approach in our future work on the verification of the PS problem-solving methods.

Keywords: Program Supervision Task, Task Modelling, Verification of Knowledge Bases.

[1]This work is being carried out at INRIA and is partially supported by the grant PF95 19899400 from the Spanish Ministry of Education and Culture.

1 Introduction

This paper is concerned with the verification of knowledge-based systems for a task called *program supervision* (PS). PS consists in the automation of the use of an existing program library, independently of any individual application.

PS systems are in general intended to work autonomously, i.e. without (or with little) human interaction, and in a varying environment. Therefore a minimum degree of reliability must be guaranteed. The required reliability makes the application of verification and validation techniques necessary.

A PS system is often in the form of a knowledge-based system composed of a reusable inference engine plus an application-dependent knowledge base containing the knowledge about the programs in a library and their use. To assure the reliability of a PS system the verification of the knowledge base is essential. The aim of this paper is to highlight how the model on which a particular PS system is based influences the properties that the knowledge base must fulfil. Adopting this point of view allows us to go beyond verification techniques based on the syntax of the knowledge representation language used. Knowledge base verification is strongly based on knowledge properties enforced by the model, e.g. the necessary types of knowledge [Marcos *et al.*, 1995], and not only on the knowledge representation scheme used. Since we rely on the underlying model of a PS system to verify knowledge bases we use the term *"model-based verification"* to denote our approach.

The plan of this paper is the following. Section 2 describes PS and the PS systems under consideration, and exposes their verification needs. Section 3 justifies our approach for the verification of PS systems. Section 4 focuses on the modelling of the PS task. Finally, sections 5 and 6 present some results and future work.

2 Program supervision

Program supervision consists in the automation of the use of an existing program library. PS sytems are often knowledge-based systems which encapsulate the knowledge about the correct use of the programs. To achieve a user's request, a combination of the programs has to be chosen, scheduled, executed and their execution controlled.

Several systems have been used to supervise image processing libraries for different applications: road scene obstacle detection (OCAPI [Thonnat *et al.*, 1994]), visual inspection for flaw detection in metal components (VSDE [Bodington *et al.*, 1992]), and analysis of interplanetary images (MVP [Chien, 1994]). Despite their differences, all these systems have to tackle with various kinds of knowledge that are inherent to the task, and with a complex knowledge organisation.

From the experience gained from our team's previous work on PS systems a "general" approach has been derived which is the framework of this work. In the next subsections this approach is summarised and its particular verification needs are described.

2.1 General framework for program supervision systems

A PS system is composed of a reusable PS engine, a knowledge base capturing the expert knowledge about the use of a library of programs, and the library itself.

Knowledge base The knowledge base encapsulates expertise on programs and processing. It contains instances of PS specific concepts such as: goals to achieve, operators (corresponding to programs and their use), program parameters and data. In PS systems intended to work autonomously, the knowledge base can also contain various criteria to perform automatically different actions. Examples are: choice among possible operators, initialisation of operator parameters, evaluation of results, dynamic repair, etc.

The most important concept is *operator*. Operators describe either programs from the library or more or less complex combinations of them, as well as information needed to apply them in different situations: function, characteristics, input and output arguments, parameters, pre and postconditions, and effects. Various *criteria* are also attached to operator description, e.g. to specify how to initialise and adjust the parameter values before execution, how to evaluate output arguments after execution, or how to react in case of unsuccessful execution.

Two types of operators can be distinguished. A *primitive operator* describes a library program, so it contains information about the calling syntax. A *compound operator* is described by means of a set of suboperators (decomposition), which can be in turn compound or primitive ones. The decomposition of a compound operator can be of different sorts: sequential, iterative, parallel, or choice. Compound operators have additional information, e.g. to specify how to choose the appropriate suboperators. By means of decompositions operators are described at different levels of abstraction. The description of an operator at all its levels of abstraction (made up by its decomposition and the decompositions within it, and so on) constitutes a *skeleton of plan*.

Many of the above concepts are quite general, e.g. operator, and are common to all PS systems although with variations on details. Other concepts, however, exist in few systems and have been included with the aim of generality.

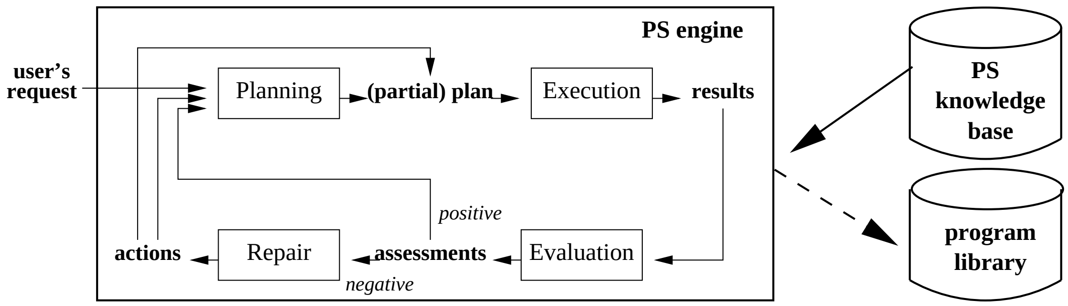

Figure 1: General behavior of a program supervision engine

PS engine The behavior of a PS engine can coarsely be divided into four steps (figure 1). An initial planning step determines the best (partial) plan to reach the goal defined by the user's request. Then the execution of the (partial) plan is triggered, i.e. the individual programs in the plan are executed. The results of the program execution are passed to an evaluation step that assesses their quality. This evaluation can be performed either automatically by using the expert's knowledge, or interactively by the user. Finally if the assessment on results is negative, a repair step decides the appropriate corrective action. The action to undertake may be either to re-execute the current program with different parameter values or to modify the plan.

Although the described behavior is quite general, variations are possible. At a high level, different controls over the steps can be used, e.g. planning and execution may be interleaved because each planning step may depend on information that is only available after the execution of previous programs in the plan. At a lower level, some basic steps can be performed in a more or less complex way. Examples of differences in behavior will be given in section 4.

2.2 Verification needs of program supervision systems

To ensure the reliability of a PS system we focus on the verification of the knowledge base built by the expert. The implementation of the concepts presented in the previous subsection uses an hybrid knowledge representation scheme. In this scheme the knowledge concepts like *operators* are represented by structured objects, whereas *criteria* are represented by rule bases in most cases. Traditional verification techniques based on the syntax of the knowledge representation scheme are limited because they do not exploit the intended use of the embodied knowledge. To overcome this limitation we first focus on the utilisation of the different kinds of knowledge. Afterwards the appropriate verification techniques (e.g. decision table techniques) will be chosen for each knowledge concept (e.g. criteria for initialisation) and applied within a context that takes into account the concept utilisation. Knowledge utilisation and representation are complementary aspects that will be considered for knowledge base verification.

Given a knowledge base and the model of the PS engine that will work with it, our aim is not only to verify the consistency and completeness of the knowledge base, but also to check the adequation of the embodied knowledge to the way in which it will be used by the PS engine, in order to warn the expert about possible inadequations. For the model of the PS engine to serve as a basis for verification, it must reflect both structural (concepts and their organisation) and functional (behavior) aspects. In the next section this approach is justified.

3 Approach to the verification of program supervision systems

We distinguish the following elements within the specification of a knowledge-based system [Fensel *et al.*, 1996]: task definition, problem-solving method, and domain model (figure 2). The task definition describes the problem to solve and imposes certain assumptions or requirements on the domain knowledge. The problem-solving method describes the reasoning process of the system and goes further into the requirements on the domain knowledge. The domain model contains the domain ontology and the domain knowledge itself. The domain ontology must reflect the requirements of both the task definition and the problem-solving method on the domain knowledge.

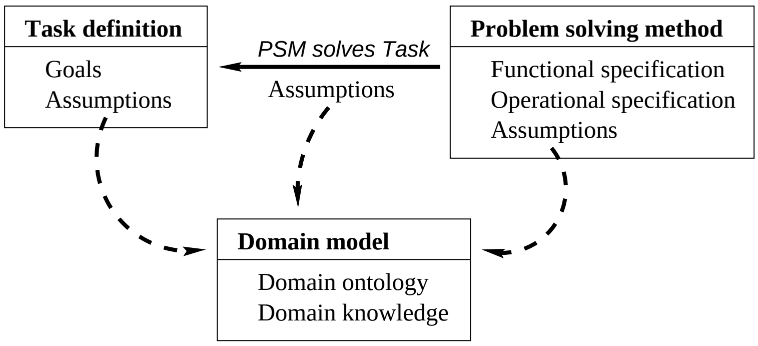

Figure 2: Elements of the specification of a knowledge-based system

Here are some examples of requirements on the domain knowledge in the PS case. The task definition requires the existence of a set of programs and of the knowledge about their use, and the existence of input data. The problem solving method requires that some of the programs in the library have parameters that can be used to tune their behavior, otherwise the repair facilities would be useless. The PS ontology (PS concepts) imposes several structural properties, e.g. operators must have at least one input argument and one output argument.

In this light, we consider the existence of several problem-solving methods to solve the PS task. The domain ontology corresponds to the general concepts that have been introduced in the previous section, and it is wide enough to be considered as fixed. This PS ontology does not address application specific concepts like `stereo-vision-matching` (a program in an image processing application), but the task concepts. For each application, the expert will supply the particular domain knowledge introducing the application specific concepts. As the PS ontology is fixed, it could not reflect all the requirements of a particular PS problem-solving method, and therefore additional requirements on the domain knowledge would be necessary [Benjamins *et al.*, 1996].

Our verification approach is based on the PS ontology and on the requirements that both the task and problem-solving method enforce on the domain knowledge of a particular application. A PS engine implements a particular PS problem solving method (PS method henceforth) which makes use of the PS ontology. In this ontology we have a well-defined knowledge organisation. First, the knowledge base must comply with this ontology definitions. Second, the particular PS method determine additional properties of the knowledge base, i.e. the required knowledge and(or) the characteristics it must fulfil so that the subtasks can be performed properly.

Therefore we rely on the conceptual model of the PS ontology and methods for the definition of the verification issues. This definition is done in a way dependent both on the task and on the particular PS method, but independently of the knowledge representation scheme or the particular application. In the next section we present a part of the PS model and some properties that have been identified from it.

4 Modelling of the program supervision task

We have used the CommonKADS expertise model to describe the problem-solving behavior of PS systems. In CommonKADS, the **knowledge categories** to describe a system are domain knowledge, inference knowledge, and task knowledge [Wielinga *et al.*, 1994].

The *domain knowledge* contains the ontology including the terms referring to the entities that can be distinguished in the task (*concepts*) and to the relationships that can exist between the entities (*relationships*). The *inference knowledge* describes the basic inferences (*inference steps*) that can be made using the domain knowledge, and the roles that it plays in these inferences. *Inference structures* show the way in which inference steps can be combined. An inference step can be decomposed into a new inference structure when it is relevant for the application. Finally the *task knowledge* describes how inferences can be combined specifying the control part.

The graphical notation that we have used to formulate the *domain knowledge* follows the object-oriented OMT notation[2]. To describe the *inference knowledge*, we have used ovals to represent inference steps and boxes to represent knowledge roles.

It must be noticed that our utilisation of KADS is not at the application level but at the task level. This means that the terms in our model are not `stereo-vision-matching` or `left-image`, which belong to a particular application, but `operator` or `data`. The next subsections present extracts from the PS ontology definitions and subtask descriptions, and some properties that have been identified using the descriptions they contain.

[2]Boxes represent concepts (classes) and contain the name of the concept and its properties. Lines represent types of relationships and may also have properties. A concept can be related to one or more refined versions of it (subclasses) constituting a hierarchy, which is represented with a triangle. Ovals indicate expressions.

4.1 The program supervision model

PS ontology Some examples of the PS ontology are presented in the following figures. Figure 3 shows the relations describing the generic information of the operator: function, characteristics, input and output arguments, parameters, pre and postconditions, and effects. It also shows the relationships modelling the criteria for initialising and adjusting the parameter values before execution, evaluating the execution results, or selecting the appropriate repair action in case of unsuccessful execution.

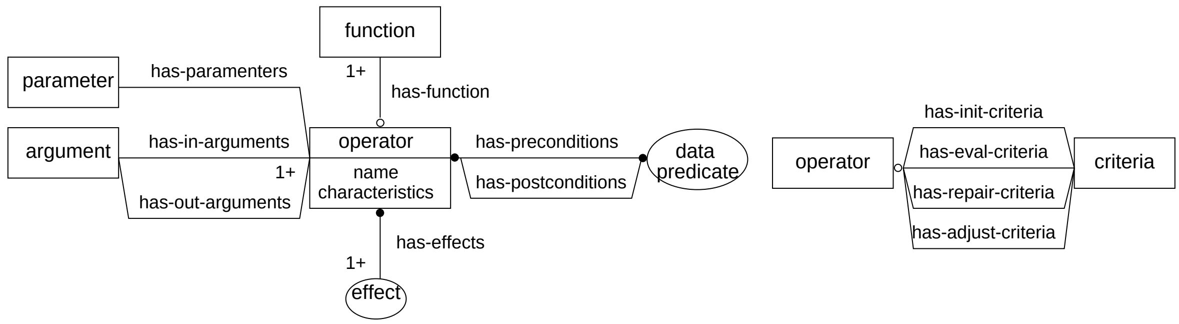

Figure 3: operator and related concepts

The decomposition of a compound operator is modelled using the concept decomposition and its subtypes. A compound operator is associated to a decomposition, as figure 4 shows.

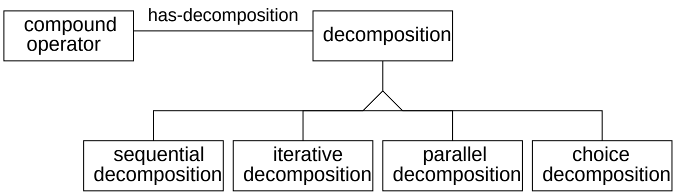

Figure 4: compound operator and decomposition concepts

All decomposition types are related to a suboperator set, but the complementary knowledge vary. For instance, choice decompositions are associated to choice criteria specifying which suboperators are to be chosen depending on the execution state (figure 5).

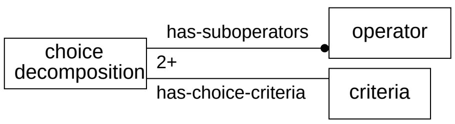

Figure 5: choice decomposition and choice criteria

PS methods As mentioned before, PS methods may vary in several points. The most important ones are the planning strategy and the aspects related to the execution, evaluation and repair of operators. The following figures focus on execution, evaluation and repair mechanisms. This study is based on existing KADS models of PS engines used in real applications ([van den Elst *et al.*, 1994], [van den Elst, 1996]) and on other existing PS engines.

Figure 1 presented the execution, evaluation and repair as independent steps. This is not the case in all systems, e.g. the refinement of compound operators implies a more complex behavior. Figure 6 presents a version of inference structure for refine, execute and repair operator which deals with this case. First the operator parameters are initialised using its initialisation criteria. Then the operator is executed if it is a primitive one, specialised if it is related to a choice decomposition, or decomposed otherwise. After this the results are evaluated, i.e. possible errors are detected and diagnosed. If errors are detected, the repair mechanism is activated. The plan is updated after every operator execution.

Figure 7 presents a version of inference structure that details the step specialise operator. First the suboperators that are candidates for the specialisation in the current state are selected using the choice criteria of the operator. Then one of them is refined, executed and repaired. PS methods may vary in the way of selecting the operator to be treated from the set of chosen suboperators, and in the exhaustiveness of the search in this set.

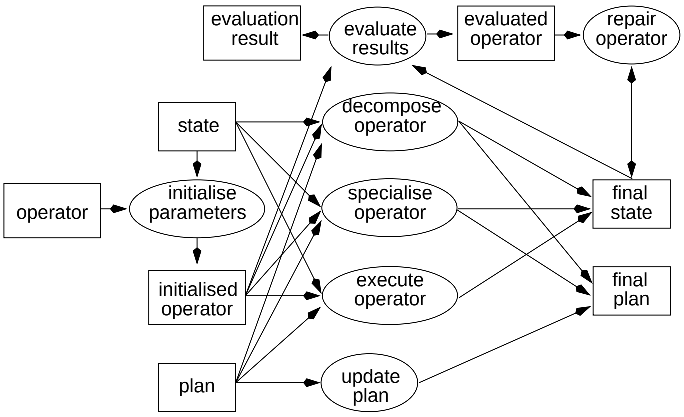

Figure 6: A version of inference structure for refine, execute and repair operator

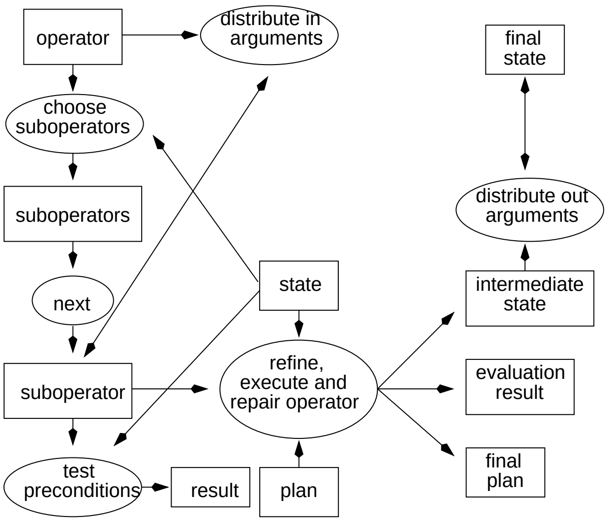

Figure 7: A version of inference structure for specialise operator

4.2 Properties identified from the program supervision model

We have used the descriptions in the PS model to derive the properties to be verified. These properties correspond to the knowledge characterisation given by the ontology and to the information on the knowledge utilisation provided by the subtask description. Particularly, the ontology definitions establish the necessary concepts in the knowledge base and the relationships that must exist among them. Each inference step determines the precise role of the knowledge it uses and imposes additional properties. Moreover, the detailed description of each subtask (inference structure plus control over the steps) makes clear the requirements so that it works properly, or which characteristics, though not critical, are not appropriate to the subtask. Some examples of properties follow.

Properties derived from the ontology definitions An operator must have at least one input argument and one output argument, but it may or may not have any parameter. The function and characteristics of the operator are optional. It must have one or more preconditions and one or more effects, and possibly one or more postconditions. The description of a compound operator at all its abstraction levels should constitute a tree structure where the leaves should be primitive operators. Criteria group together elements with a particular function (e.g. initialisation). They should be free of potential sources of errors such as *redundancy*. The analysis of errors that can prevent criteria from adequately performing their function takes a better place in the context of the role they play.

Properties derived from the roles imposed by inference steps The parameters of an operator must be given a value whenever it is going to be specialised, decomposed or executed. This is done by initialise parameters in figure 6. For this step to be successful, in the initialisation criteria of the operator there should be at least one initialisation element for parameter. Even more, the element(s) initialising a parameter should cover all possible situations in the application context of the criteria (*completeness*). There should not be two elements concluding different values for the same parameter in the same situation (*conflict*). Choose suboperators in figure 7 selects the candidate(s) suboperator(s) for the current specialisation. For this purpose, the choice criteria of the operator decomposition should cover all possible situations and contain at least one element choosing each suboperator (*completeness*). Notice that there is no problem in choosing two different suboperators in the same situation because specialise operator deals with this.

Characteristics proper to the functioning of subtasks In figure 7 the way of selecting the suboperator to be the actual specialisation from the set of candidates can be more or less complex. A smart selection mechanism (other than next) usually demands additional knowledge, e.g. the function or characteristics of operators, which becomes a requirement of the particular method rather than a property of PS. Besides the selection mechanism, the search in the set of candidate suboperators can be exhaustive or not. An exhaustive search is more suitable for choice criteria that choose multiple suboperators, and inversely, such choice criteria are useless when the search is not exhaustive.

5 Lessons learned

From this experience we have drawn different lessons that can be generalised to other tasks.

The first result concerns the aim of this paper and is a definition of the verification issues for the knowledge base which makes use of the description of PS methods. Our claim is that in this way we can go further than we would do otherwise, e.g. there is no conflict in choice criteria choosing two suboperators in the same situation if specialise operator deals with this. Additionally, the definition of the verification issues uses the task terminology which is more meaningful for the expert, e.g. *"initialisation criteria x of operator y are adequate"* instead of *"rule base x of operator y is complete with respect to all parameters"*.

The second result is a categorisation of the properties to be verified. They can be classified according to different points of view that are rather general. The knowledge base properties can make reference to the knowledge *structure* or to the knowledge *role* imposed by the PS method. The former can be directly deduced from the ontology definitions, while the latter are derived from the role that each piece of knowledge play in the reasoning process of the particular method. From another point of view, many of them are *method-independent* characteristics, but other are *method-dependent* ones, which arise when we consider the behavior of the PS method that will use the knowledge base. Finally, the properties can be *compulsory* or not (*desirable*).

Third, we have obtained some guidelines to determine the adequacy of the expert's knowledge with respect to the requirements of the PS task and methods. Lastly, the analysis of different PS methods provides a means of comparing them.

6 Conclusions

The model-based approach that we present exploits the intended use of the knowledge in the PS task for verification purposes. First we have built a model of the PS task and then we have used it to identify the properties that the knowledge base should verify. Many of them are based on the requirements of the particular PS method. Although we have addressed a particular task, the approach is quite general.

We will apply the results of this study to our existing platform for the construction of real PS applications. This platform provides facilities for engine development and knowledge base building. The edition environment for knowledge bases uses a language for knowledge description which has associated a syntax checker. This checker already verifies all the method-independent structural properties while the expert is developing the knowledge base. The work described in this paper constitutes the basis for the specification of a verification tool to check the method-dependent properties. Some of them require time-consuming checks and will be done at the expert's request.

In the future we intend to apply traditional software verification techniques to the verification of PS methods (validation of PS systems). The analysis of PS methods that we have done provides a means of comparing them in terms of the steps that are present, how these steps can differ, and which knowledge they require. It is an open question whether this analysis can help us to determine some meaningful properties of PS methods, besides completeness and termination.

6.1 Related work

In [Valente, 1994] we find a knowledge level analysis of classical planning systems in terms of the types of knowledge they use and how they are structured. The aim is providing a framework for classifying methods in order to facilitate their selection for a particular application. [Barros *et al.*, 1996] complements this work with an analysis of the planning subtasks and the use they make of the different types of knowledge. Our analysis comprises most of these aspects although with a different objective.

A study of the assumptions (what we have called requirements) of a group of problem-solving methods for model-based diagnosis is found in [Fensel and Benjamins, 1996]. Assumptions are viewed as a characterisation of problem-solving methods and as guidelines for the acquisition process of domain knowledge. However, the focus is on their role to reduce the complexity inherent to the diagnosis task and to develop or adapt problem-solving methods, i.e. their role in the development of efficient problem-solving methods.

Our work is similar but focused on the role of assumptions to guide the acquisition process.

In [Fensel *et al.*, 1996] the Karlsruhe Interactive Verifier (KIV) [Reif, 1995] is used to verify the assumptions relating the different parts of the specification of a knowledge-based system. The working example is the task *select the best explanation* and the problem-solving method *hill-climbing*. KIV is used for the verification of the method but also as a tool for the detection of the assumptions necessary to relate the competence of the method to the task definition. This work shows the importance of using a software verification tool not only for the method verification itself but also for the detection of hidden assumptions. We intend to work in this line in the future.

References

[Barros *et al.*, 1996] L. Barros, A. Valente, and R. Benjamins. Modeling Planning Tasks. In *International Conference on Artificial Intelligence Planning Systems*, 1996.

[Benjamins *et al.*, 1996] R. Benjamins, D. Fensel, and R. Straatman. Assumptions in problem-solving methods and their role in knowledge engineering. In *European Conference on Artificial Intelligence*, 1996.

[Bodington *et al.*, 1992] R. Bodington, E. Gallier, and C. Theot. The Vision System Development Environment (VSDE): A software environment for the automatic configuration of inspection systems. Draft version, 1992.

[Chien, 1994] S. Chien. Automated synthesis of image processing procedures for a large-scale image database. *IEEE International Conference on Image Processing*, pages 796–800, 1994.

[Fensel and Benjamins, 1996] D. Fensel and R. Benjamins. Assumptions in Model-based Diagnosis. In *Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.

[Fensel *et al.*, 1996] D. Fensel, A. Schönegge, R. Groenboom, and B. Wielinga. Specification and verification of knowledge-based systems. In *European Conference on Artificial Intelligence workshop on Validation, Verification and Refinement of Knowledge-Based Systems*, 1996.

[Marcos *et al.*, 1995] M. Marcos, S. Moisan, and Angel P. del Pobil. Verification and validation of knowledge-based program supervivion systems. In *IEEE International Conference on Systems, Man and Cybernetics*, 1995.

[Reif, 1995] W. Reif. The KIV Approach to Software Verification. In M. Broy and S. Jähnichen, editors, *Methods, Languages and Tools for the Construction of Correct Software*, number 1009 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1995.

[Thonnat *et al.*, 1994] M. Thonnat, V. Clément, and J. Van den Elst. Supervision of Perception Tasks for Autonomous Systems : The OCAPI Approach. *International Journal of Information Science and Technology*, 3(2):140–163, 1994.

[Valente, 1994] A. Valente. Knowledge-Level Analysis of Planning Systems. *SIGART Bulletin*, 6(1):1–9, 1994.

[van den Elst *et al.*, 1994] J. van den Elst, F. van Harmelen, G. Schreiber, and M. Thonnat. A functional specification of reusing software components. In *International Conference on Software Engineering and Knowledge Engineering*, 1994.

[van den Elst, 1996] J. van den Elst. *Knowledge Modelling for Program Supervision in Image Processing.* PhD thesis, Université de Nice, 1996.

[Wielinga *et al.*, 1994] B. Wielinga, H. Hassan, O. Olsson, K. Orsvärn, G. Schreiber, P. Terpstra, W. van de Welde, and S. Wells. Expertise Model Definition Document. Technical Report KADS-II/M2/UvA/026/5.0, 1994.