The Advanced Knowledge Transfer System

Larry Lucardie
Johan De Gelder
Adriaan Huijsing

TNO Building and Construction Research
Section Knowledge-Based Systems
P 0 Box 49
2600 AA Delft
NETHERLANDS

The joint application of decision tables and Prolog seems to meet all the necessary requirements to be met by a language or modelling knowledge. Despite the In h complementarity of decision tables and *Prolog, it appears that they still yield a language with certain drawbacks. The Advanced 'knowledge Transfer System TS) has been developed to take advantage of this complementarity and simultaneously eliminate these drawbacks. To show the capabilities of AKTS three knowledge-based systems in the building and construction sector are described which recently have been developed using AKTS.*

Keywords: knowledge-based systems, modelling language, decision tables, Prolog

I Introduction

The last decade knowledge-based systems are becoming increasingly important for the daily practice of many organisations. Today they even have attained a permanent and secure role in trade and industry (Hayes-Roth & Jacobstein, 1994). Yet, the development of knowledge-based systems that are applicable in the daily practice of organisations, remains problematic. Many prototypes of knowledge-based systems simply never go into usage. A precondition for the specification, design and implementation of operational knowledge-based systems is the availability of an adequate language. Such a language should not only offer expressive power for the modelling of knowledge, but should also provide validation and simulation facilities.

The aim of this article is to describe a computer tool that supports the specification, design and implementation of knowledge-based systems by offering decision tables and Prolog as a modeling language. The structure of the paper is as follows. First, we analyse the benefits and drawbacks of decision tables and Prolog as a language to model knowledge (Section 2). Then, we describe the Advanced Knowledge Transfer System (AKTS) as a tool taking benefit of the integration of decision tables and Prolog, while avoiding associated drawbacks (Section 3). To illustrate the possibilities of AKTS we shortly discuss three examples of knowledge-based systems in the building and construction sector that have been developed using AKTS (Section 4). The article is rounded off with conclusions concerning the modelling, validation and simulation facilities of AKTS (Section 5).

2 Decision tables and Prolog as a modelling language

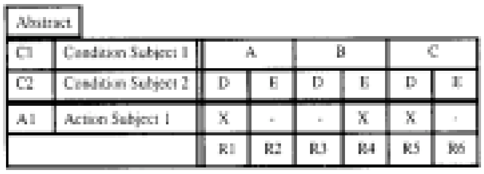
Some authors claim that decision tables and Prolog together offer a range of powerful formalisms and techniques allowing a formally unambiguous description of realworld phenomena that is close to natural understanding (Reilly, Salah, & Yang, 1987, p.30). In this section we will investigate this claim. We first describe the potentials of decision tables as a modeling language and then proceed by discussing Prolog's capabilities as a modeling language and the joint use of decision tables and Prolog.

2.1 Decision Tables

A decision table can be informally defined as:

'...a table that represents the exhaustive whole of mutually exclusive conditional statements within an a priori defined problem domain.' (Verhelst, 1980, p.9)

An example of a decision table is presented in Figure 1. The table named *Abstract,* refers to a fictitious domain. The component to the left of the double vertical line is called the *stub.* The first part of the stub, the part that is located above the double horizontal line, contains *condition subjects.* The second part of the stub, located below the double horizontal line, contains *action subjects.* The component to the right of the double vertical line displays six conditional statements about our fictitious domain. These statements are called *rules.* They are displayed by means of columns. Rules describe the connection between condition subjects and action subjects. Above the double horizontal line the rules contain a *condition alternative* for each condition subject. Below the double horizontal line the rules contain an *action alternative* for each action subject. According to Verhelst's definition a decision table should be *exhaustive* and *exclusive.* Exhaustiveness means that, within the domain of the decision table, every possible combination of condition alternatives should be accounted for. Exclusiveness means that no situation is permitted to be described in more than one rule.



Abstract

C1	Condition Subject 1	A		B		C	
C2	Condition Subject 2	D	E	D	E	D	E
A1	Action Subject 1	X	-	-	X	X	-
		R1	R2	R3	R4	R5	R6

Figure 1: A graphical Sketch of a Decision Table

For most applications a single decision table is not sufficient. In the majority of cases we need a decision table system: a set of at least two decision tables in which each decision table is linked to another table belonging to the same system. We can distinguish two types of links. The first type of link is established by the phenomenon that a condition subject with at least one of its alternatives of one decision table occurs as an action subject with corresponding action alternatives in another decision table. The first table is then called a *head table,* the second table is called a *condition subtable.* The second type of link is established by the phenomenon that an action subject and one of its alternatives of one decision table occurs in the same form in another table. The first table again is called a *head table,* the second table is now called an *action subtable.* Usually, the action subject in question, in this action subtable, is further specified by means of other action subjects. Figure 2 displays an example of a decision table system containing a head table, a condition subtable and an action subtable.

Much of the literature advocating decision tables is devoted to their applicability in all phases of the software engineering life cycle from specification through design, implementation, testing, modification and documentation to maintenance. The advantages of decision tables for software engineering purposes are succinctly expressed by Reilly, Salah & Yang:

the ubiquitous use of decision tables within the life cycle has conferred on them a reputation for compactness, self-documentation, modifiability handling complex logic, redundancy and completeness checking, high degree of non-procedurality, and automatic conversion to code.' (1987, p.191)

However, the employment of decision tables has its drawbacks:

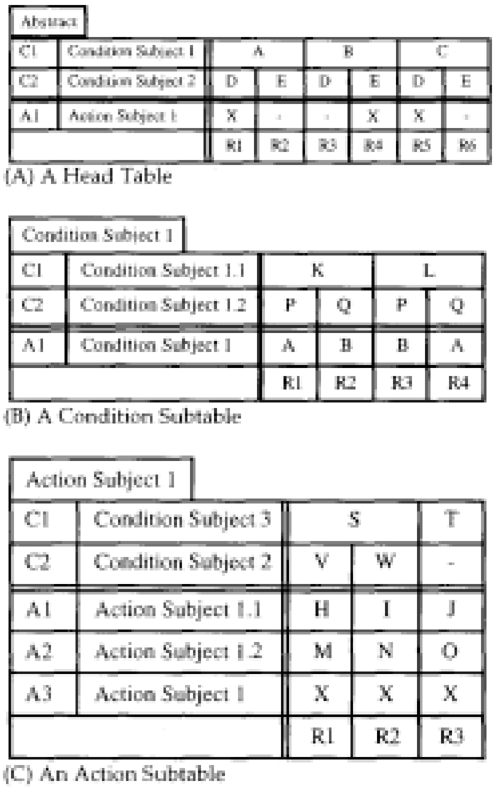
(1) Decision tables are not the most efficient means to model very simple pieces of knowledge. For the modelling of database tables, for instance, it's not effective to use decision tables. In Section 4 we will illustrate this by an example.

(2) They are not fully appropriate for modelling knowledge in which a (frequent) use of recursions, iterations or other repeat structures is required.

(3) Simulation by hand is possible, but not a task which can be undertaken easily. Decision tables do not generate prototypes that can facilitate simulation processes (Davis, 1988, p.1113).

(4) Decision tables do not provide automated checking facilities (Davis, 1988, p. 1113).

(5) Drawing decision tables is a very time-consuming and difficult process. As long as no advanced graphical tools are available to support the complex drawing of decision tables, the acceptation of decision tables will be retarded.



Abstract

C1	Condition Subject 1	A		B		C	
C2	Condition Subject 2	D	E	D	E	D	E
A1	Action Subject 1	X	-	-	X	X	-
		R1	R2	R3	R4	R5	R6

(A) A Head Table

Condition Subject 1

C1	Condition Subject 1.1	K		L	
C2	Condition Subject 1.2	P	Q	P	Q
A1	Condition Subject 1	A	B	B	A
		R1	R2	R3	R4

(B) A Condition Subtable

Action Subject 1

C1	Condition Subject 3	S		T
C2	Condition Subject 2	V	W	-
A1	Action Subject 1.1	H	I	J
A2	Action Subject 1.2	M	N	O
A3	Action Subject 1	X	X	X
		R1	R2	R3

(C) An Action Subtable

Figure 2: A Decision Table System

2.2 *Prolog*

Under the influence of Kowalskis ideas on logic and theorem proving, Prolog was designed around 1970 by A. Colmerauer of the University of Marseilles. Prolog has been developed from the idea that a good programming language is a powerful modeling tool for organising, expressing, communicating and executing knowledge. One of the reasons that we can regard Prolog as a modelling language is that it allows users to *program by description. Well-styled Prolog programs are readable as logical statements saying something about the real -world. Let us illustrate this by a view simple Prolog statements:*

```
vertex(v1,1,1,0).             /* Vertex v1 has co-ordinates 1,1,0 */
vertex(v2,2,2,0).             /* Vertex v2 has co-ordinates 2,2,0 */
vertex(v3,3,1,0).             /* Vertex v3 has co-ordinates 3,3,0 */
edge(e1,v1,v2).               /* Edge e1 links vertices v1 and v2 */
edge(e2,v2,v3).               /* Edge e2 links vertices v2 and v3 */
edge(e3,v3,v1).                /* Edge e3 links vertices v3 and v1 */
                                         connected(E1,E2) :-
edge(E1,_V), edge(E2,V_)       /* Edge E1 is connected to edge E2 if the
                                         final vertex of E1 is the initial
                                         vertex of E2 */
```

These Prolog statements describe a simple triangle. The example is borrowed from Gonzalez et al. who developed a geometric modeler with particular emphasis to update three-dimensional models by means of operations on two-dimensional views. (Gonzalez, Williams, & Aitchison, 1984). Though this use of Prolog for CAD applications is interesting, of much more importance is the underlying idea that knowledge can be modelled using Prolog and that the Prolog statements are executable! Prolog admits the execution of knowledge and thus provides facilities to validate and simulate knowledge. This automatic generation of prototypes yields an important advantage over decision tables which are not executable. An extensive analysis of the Prolog's background as modelling language is provided in (Lucardie, 1994). Prolog also has its weak points. Prolog does not impose, for instance, a methodology for modelling knowledge.

2.3 Decision tables and Prolog

The application of decision tables and Prolog seems to yield a convenient complementarity. Decision tables are a method for organising and documenting knowledge in a logical manner that permits easy inspection and analysis, while Prolog can be viewed as a logical modelling language that admits simulation of knowledge.

However, the joint application of decision tables and Prolog still yields a language with certain limitations. This language does, for instance, still not offer facilities for automated validation and automated simulation of decision tables. Furthermore, essential graphical facilities for drawing decision tables are lacking.

3 The Advanced Knowledge Transfer System

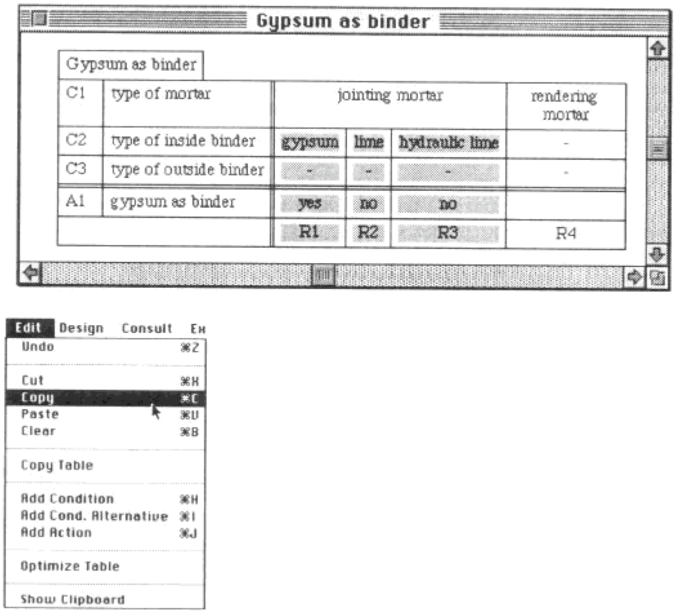
In this section we will describe the modelling, validation and simulation facilities of AKTS.

3.1 Modelling facilities: the graphical decision table editor

To take full advantage of the structuring facilities of decision tables, AKTS stimulates a user to start knowledge modelling with the creation of a decision table system. In the previous section we, however, observed that the adoption of decision tables is retarded because of the fact that drawing and modifying tables is a complicated and timeconsuming process. This is mainly due to the fact that the modification of a part of a decision table often has consequences for other parts of the decision table, so that the complete decision table has to be redrawn. Even a slight modification may require redrawing a complete decision table.

To intercept this problem, AKTS is equipped with a decision table editor that provides a multitude of convenient *graphical* facilities. This editor has a carefully designed multi-window, menu-driven mouse-oriented interface for optimal communication with users. It allows users to quickly reconstruct and modify a complete decision table system. The editor shows 'intelligent behaviour': it knows what a correct table looks like and applies this knowledge to support the user in the drawing process. The editor also knows how to obtain a decision table that occupies a minimal amount of space. This knowledge is used, if necessary, to calculate the measures of a minimal table after each user's action.

Furthermore, AKTS offers a facility to insert Prolog statements. This is necessary to insert knowledge that cannot easily be represented through decision tables. These statements can be used while simulating knowledge. In the next section we will exemplify the use of Prolog through AKTS.



Gypsum as binder

Gypsum as binder					
C1	type of mortar	jointing mortar			rendering mortar
C2	type of inside binder	gypsum	lime	hydraulic lime	-
C3	type of outside binder	-	-	-	-
A1	gypsum as binder	yes	no	no	
		R1	R2	R3	R4

Figure3: Selecting a Subtree in a Decision Table and Copying it into the Clipboard of AKTS

3.2 Validation facilities: the integrity control sub-system

The Integrity Control Sub-system of AKTS is responsible for maintaining integrity of knowledge. It offers possibilities to check

(1) the exhaustiveness and exclusiveness of every decision table and

(2) the formal links between head tables and subtables.

The exhaustiveness and exclusiveness integrity constraints are based on the formal definitions of exhaustiveness and exclusiveness. The integrity constraints for links are based on the formal definitions of (bilateral) connections (Lucardie, 1994).

3.3 Simulation facilities: the inference engine

The inference machine of AKTS is goal-oriented. It departs from a list of goal parameters (Parameters are conditions or actions in a decision table or other variables occurring in a field of application). The inference machine attempts to trace these goal parameters in order of appearance. Tracing a parameter is the process of trying to find a value for a parameter. The inference machine of AKTS uses a backward chaining strategy, though it is possible to influence the inference engine. During this inference process parameters that are not goal parameters also must be traced. These non-goal parameters are relevant for tracing the goal parameters. There are several ways to find a value for a parameter; they are described below.

(1) If the parameter to be traced has an *Ask* first property, its value is simply asked. Whether the user has to make a selection out of a list of possible values or type in the exact value of the parameter depends on the Type property of the parameter. If the value of the parameter can be found in other decision tables, the user has the option of indicating that he does not know the answer.

(2) Another way of finding a value for a parameter is provided by the *When Needed* property. The *When Needed* property is often used if the value of a parameter has to be calculated and the necessary formulas are available in the form of the *When Needed property value. Another use of this property is to design a query to a database as a When Needed* property value. The calculation of the query then returns the value of the parameter.

(3) The value of a parameter can also be found by tracing decision tables. AKTS then tries to look for a decision table that contains the parameter to be traced as an action parameter. The value of the action parameter depends on the condition parameters of the decision table. In order of appearance the condition parameters are traced until the value of the action parameter is found. When the condition parameters are traced again other decision tables can be searched through. In this way a whole system of decision tables can be traced by the inference machine of AKTS.

(4) If still no value is found the inference machine uses the *Default* property if present. The value of the *Default* property becomes the value of the parameter.

(5) If the above strategies fail, the user is finally asked (again) to provide a value. But now the user is forced to provide an answer and does not have the possibility to answer 'Don't Know'.

4 Examples of Knowledge-Based Systems Using AKTS.

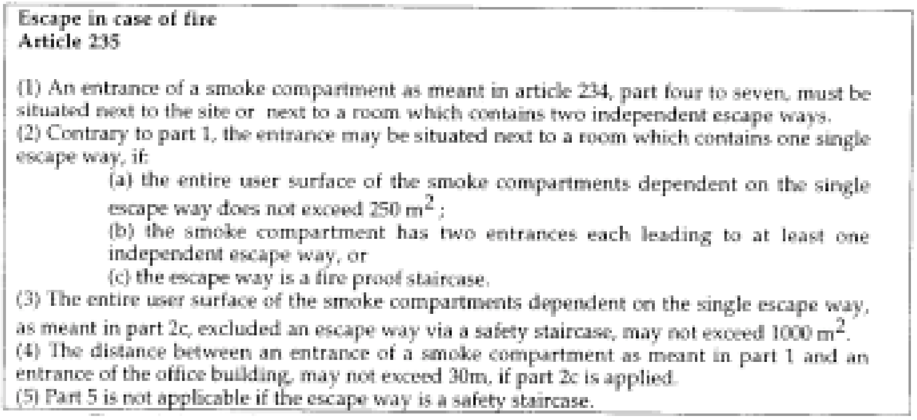
In this section we shortly describe three knowledge-based systems which are specified, designed and implemented using AKTS.

4.1 Fire Safety, Advisory System

TNO Building and Construction Research developed a knowledge based system using AKTS that helps to verify building designs with respect to Fire-Safety Regulations.

Knowledge of Fire-Safety Regulations is critical for architects to design fire-safe buildings and for local authorities to verify fire-safety of building designs. However, effective application of this knowledge in the Netherlands is a problem due to volume, complexity and inaccessibility of the Dutch Fire-Safety Regulations. The Fire-Safety Regulation are known as the most complex part of the Dutch Building Regulations. This often leads to misunderstanding and misinterpretation of the regulations by people in practice.

Figure 4 shows act of article 235 of the Dutch Building Regulations. Article 235 gives rules for escaping from a smoke compartment in case of fire. A smoke compartment is a part of a building having smoke resistant walls. In case of a fire in the smoke compartment people can escape to adjacent smoke compartments that are protected against smoke for a certain period because of the smoke resistant walls.



Escape in case of fire
Article 235

(1) An entrance of a smoke compartment as meant in article 234, part four to seven, must be situated next to the site or next to a room which contains two independent escape ways.
(2) Contrary to part 1, the entrance may be situated next to a room which contains one single escape way, if:

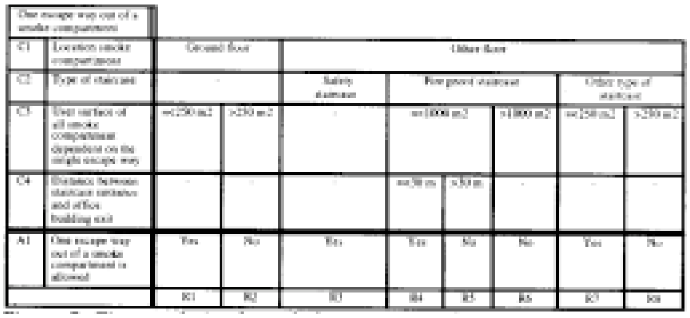
(a) the entire user surface of the smoke compartments dependent on the single escape way does not exceed 250 m^2 ;
(b) the smoke compartment has two entrances each leading to at least one independent escape way, or
(c) the escape way is a fire proof staircase.

(3) The entire user surface of the smoke compartments dependent on the single escape way, as meant in part 2c, excluded an escape way via a safety staircase, may not exceed 1000 m^2.
(4) The distance between an entrance of a smoke compartment as meant in part 1 and an entrance of the office building, may not exceed 30m, if part 2c is applied.
(5) Part 5 is not applicable if the escape way is a safety staircase.

Figure 4: A part of Article 235 of the Dutch Building Regulations

Article 235 prescribes that in principle each smoke compartment should have at least two independent escape ways. This can be realised in two different ways. From an entrance of a smoke compartment people can escape in two different and independent directions (part 1) or a smoke compartment has two entrances which both lead to at least one independent escape way (part 2b).

In specific circumstances it is allowed that a smoke compartment has only one
escape way. Parts 2a, 2c, 3, 5 and 6 of article 235 give rules for these circumstances. Figure 5 shows one of the decision tables of the Fire Safety Advisory System that represents the rules for the allowance of a single escape way from a smoke compartment.



One escape way out of a smoke compartment									
C1	Location smoke compartment	Ground floor		Other floor					
C2	Type of staircase	-		Safety staircase	Fire proof staircase			Other type of staircase	
C3	Used surface of all smoke compartments dependent on the single escape way	<=250 m2	>250 m2	-	<=1000 m2		>1000 m2	<=250 m2	>250 m2
C4	Distance between staircase entrance and office building exit	-	-	-	<=30 m	>30 m	-	-	-
A1	One escape way out of a smoke compartment is allowed	Yes	No	Yes	Yes	No	No	Yes	No
		R1	R2	R3	R4	R5	R6	R7	R8

Figure 5: Fire regulation knowledge represented in a Decision Table

The Decision Table clearly shows the situations in which one escape is allowed. The knowledge in the decision table can easily be validated on correctness, completeness and consistency by experts in the fire-safety domain. Non-experts can easily access the knowledge, especially when this knowledge is accessed using AKTS. Users of the system just answer questions prompted by AKTS and the system gives advise depending on the given answers. Problems of misunderstanding and misinterpretation disappear. With the Fire Safety Advisory System, people in practice don't need to have elaborate knowledge of the fire-safety regulations, but still they are able to apply the regulations in an efficient and effective way.

4.2 Rent subsidy

Under specific people in the Netherlands can get rent subsidy from the Ministry of Public Housing. This subsidy is meant to provide lower income groups with the opportunity to live in dwellings of a certain quality that would otherwise be too expensive. The rules that regulate this subsidy are, however, quite complicated. For this reason, a knowledge based system is developed that supports government officials in the decision-making process concerning the eligibility and height of rent subsidy.

The knowledge-based system consists of two parts. In the first part the eligibility of an applicant is assessed based on age, marital status, income and living conditions. Knowledge to make this assessment is modelled in decision tables using AKTS.

In the second part of the knowledge based system, the height of the rent subsidy is calculated. After some arithmetic preprocessing of the applicants income, the height of the rent subsidy can be found in an ordinary table. There are several of these tables, each of which is applicable in different situations. These tables are not modelled in decision tables, but in Prolog. Of course it is possible to use decision tables, but that would result in very large decision tables with hundreds of rules. Not only would it be a laborious task to construct and maintain these decision tables, the readability of the knowledge model would not be positively influenced.

A small part of one of the rent subsidy tables, modelled in Prolog, is shown below:

```
table1(22500, 3520, 420).
table1(22500, 4000, 900).
table1(22500, 4520, 1380).
table1(22500, 5055, 1860).
table1(22500, 5630, 2340).
table1(22500, 6230, 2820).
table1(22500, 6880, 3300).
table1(22500, 7565, 3780).
table1(22500, 7795, 4200).
```

The first and second argument represent limits to income and rent. The third argument shows the height of the rent subsidy. To interpret the tables, and to perform some additional arithmetic, a small Prolog algorithm was written. A part of this program is shown below:

```
        calc_rent_subsidy(Table_no,Income_applicant,Income..partner,Rent,Rent_subsid
y):
        max(Income_applicant,Income_partner,High_income),
        min(Income_applicant,Income_partner,Low_income),
        max((Low_income-2000),O,Relevant_low_income),
        Income is High _income+Relevantiow_income/2,
        consult_table(Table_no,Income,Rent,Rent_subsidy).
```

The knowledge based system reflects how decision tables and Prolog complement each other. Where decision tables are appropriate for complex decision logic, Prolog can be used for more database-like knowledge structures and arithmetic algorithms.

4.3 Evaluating indoor environments of buildings

Complaints about the indoor environment in offices are widespread. In many cases, finding causes for these complaints is far from easy. For several complaints different causes are possible. Headaches for instance can be due to high temperatures, high noise levels, high concentrations of certain pollutants or inferior lighting conditions. Many factors have to be considered like building materials, heating ventilation and airconditioning system, maintenance aspects, outdoor environment and activities in the building.

Using AKTS a system for diagnosing complaints about the indoor environment in offices has been set up. Experience and literature data with regard to the relations between indoor environment parameters and health complaints have been brought together in decision tables. This has been done in close cooperation between a knowledge engineer and experts on indoor environment.

Four types of indoor environment parameters have been distinguished: indoor climate (temperature, humidity, air velocity), indoor air quality (pollutants), light and noise. The decision tables, which can be seen as a compact representation of the knowledge in a certain field, can be read and understood by the experts. In this way they can easily check knowledge on correctness, completeness and consistency.

When a specific complaint is entered, the diagnostic tool identifies possible causes. This is done by checking whether known causes for the complaint are present in the building. If so, it is checked whether it is likely that the factor or parameter had caused the specific complaint. Questions to gather the necessary information about the building, hvac system, activities etcetera are presented to the user. From the information the possible cause(s) for the complaints are derived. Based on the diagnosis, advices to improve the situation are derived and presented.

Using the indoor environment diagnostic system, evaluations of complaints about the indoor environment can be made quickly, also by non-experts. This diagnosis is a sound basis for choosing measures to realise a healthy building. Users of the system can be people dealing with facility management in office buildings, consulting engineers and people responsible for health and safety at the workplace.

5 Conclusion

The conclusion is justified that, to a considerable degree, AKTS takes advantage of the complementarity of decision tables and Prolog. AKTS not only uses the strong points of decision tables (their structuring capabilities, their well-organised representation of knowledge permitting easy validation and simulation by hand), but also intercepts their weak points (lack of possibilities to incorporate knowledge in databases and arithmetic and recursive definitions) by using Prolog.

It is also justified to draw the conclusion that AKTS, in a convenient way, overcomes the three limitations observed of jointly applying decision tables and Prolog:

lack of facilities for automated validation and automated simulation and lack of facilities for drawing decision tables.

First, AKTS permits automated validation. By dealing with exhaustiveness and exclusiveness and (bilateral) connections between decision tables, the Integrity Control Sub-system of AKTS helps to validate the model. Furthermore, by simulating knowledge in decision tables, validation of the program code is superfluous. The knowledge model is at the same time the program code.

Second, AKTS permits automated simulation. The inference engine of AKTS provides extensive facilities to simulate knowledge including facilities for conducting What-if analyses.

Third, the Graphical decision table Editor of AKTS offers advanced graphical support to quickly construct and modify decision table systems.

The examples show the expressive power and flexibility of AKTS.

6 References

Davis, A. M., A Comparison of Techniques for the Specification of External System Behavior, *(Communications of the ACM, 31(9), 1988); 1098-1115.*

Gonzalez, J. C., Williams, M. H., & Aitchison, E. I., Evaluation of the Effectiveness of Prolog for a CAD application, *(IEEE Computer Graphics and Applications., 1984)*

Hayes-Roth, F., & Jacobstein, The State of Knowledge-Based Systems, *(Communications of the ACM, 37(3), 1994); 27-39.*

Lucardie, G. L., *Functional Object-types as a Foundation of Complex Knowledgebases Systems.* (Computer Science, Technical University Eindhoven, *1994).*

Reilly, K. D., Salah, A., & Yang, C., A Logic Programming Perspective on Decision Table Theory and Practice *(Data and Knowledge Engineering, 2, 1987); 191-210.*

Verhelst, M., *De Praktijk van Beslissingstabellen,* (Deventer: Kluwer, *1980).*