Preventing Knowledge Transfer Errors: Probabilistic Decision Support Systems Through The Users' Eyes

Hermi J.M. Tabachneck-Schijf, Petra L. Geenen
Department of Information and Computing Sciences,Utrecht University
PO Box 80.089,3508 TB, Utrecht, The Netherlands
{hermi, petrag}@cs.uu.nl

Abstract

Development and use of probabilistic decision support systems benefit by a good communication between the developer on the one hand, and the user and the domain expert on the other hand. Communication is difficult because large differences in training and experience exist between the two. This necessitates *user-centered design* of the representations used in this communication, and attention to the translation of user terms to model terms. A systematic approach to developing user-centered representations and preventing knowledge transfer errors is outlined in this paper. We demonstrate how five heuristic guidelines can be fruitfully applied in different developer-user interaction situations in different phases of decision-support system construction.

1 INTRODUCTION

Increasingly, developers of decision-support systems (DSSs) that build upon a probabilistic network realize that many end-users do not understand the classical probabilistic network representation, comprised of nodes, edges and conditional probability tables, well enough to work with it in an application. Although quite a few developers have attempted to adapt this representation so that the end-user may have a better understanding of it, sometimes ending up with quite different representations, (e.g. Long et al., 1996; Hernando et al., 2001; Kappen et al, 2002), we have not come across a systematic approach to tackling this representation problem. In addition, our research group has noticed problems with the transfer of knowledge from the user and the expert to the developer. As errors in this transfer can have consequences for the structure and content of the probabilistic network, paying attention to possible error sources can pay off. A systematic approach that addresses prevention of errors in knowledge transfer in both directions increases the probability that the design of the resulting model and representations are optimal.

We believe that the source of the lack of understanding lies in differences in the knowledge possessed by the developer and the end-user, on which we will elaborate in section 2. Knowledge-transfer problems do not only occur in the phase where the end-product reaches the end-user. There are many phases during the construction of the probabilistic network and the design of the DSS in which knowledge transfer between developers and domain experts or end-users is necessary and errors may be introduced.

We have identified five knowledge-transfer phases. The first two phases concern the manual construction of a probabilistic network for which usually the knowledge of domain experts is elicited. The interactions between the domain expert and the developer in these two phases are needed even if the system is only intended for AI research. In the first phase, knowledge needs to be elicited to determine the network's content and structure. For this, information about the relevancy of, values of, and interrelations between the stochastic variables is needed, as well as the probabilities to be assigned to the relations. The knowledge elicited from the domain expert has to be translated, i.e., to be put in terms of the model.

In the second phase, the model has to be evaluated as to its quality and diagnostic value. The model's reasoning should approximate that of an expert – well, actually an improved expert without decision or reasoning biases and with the ability to handle very complex computations. Two possible developer-expert interactions are distinguished here, one concerning the performance and the other the debugging of the model. Checking performance can be done using real cases, but the performance of the expert in such cases cannot be considered a gold standard as it contains the very errors the model tries to prevent. However interesting, that problem falls outside the scope of this paper. There has, to our knowledge, not been much research in whether and how to involve the expert in debugging the model. Especially in debugging is the format of the information, a probabilistic network and other mathematical formats, very difficult to understand by experts. Our research group is currently researching representation formats with which the expert can be more fruitfully involved in both performance checking and debugging.

Phase three, four and five relate to the design of a DSS interface that is meant for end-users which are not mathematicians or computer scientists. Of interest in these phases is first with whom the developer should interact. Although domain experts generally also qualify as end-users, they cannot be considered equal to end-users because they will unavoidably know more about the domain of probabilistic reasoning in the network by the time their interaction with the developers of the network comes to an end. Because of this knowledge they may not interpret statements about probability in the same way as still naive end-users. Naive end-users are therefore preferred. Because experience has a great influence on acquired knowledge, reasoning power, and on how someone will carry out their tasks (Anderson, 2004), we strongly feel that the *actual* projected users should be consulted in these three phases, and not the domain expert or, e.g., easier available students in the domain.

The third phase consists of gathering the needed evidence for the model: data-entry. In data-entry, the format in which information about values of the variables is elicited is the issue. This representation should be understood by the user, but yet the information in it must be true to the contents of the probabilistic network. Moreover, the answers given by the end-users are possibly not in the terms of the network, and translation back to network terms may be needed. Also, the data-entry order should comply with the normal working order of the user. Our solution for the data-entry is to apply *user-centered design* to the data-entry module's interface. In user-centered design, an interface is designed in collaboration with the end-user and thus specifically targeted to that type of end-user.

Showing the results comprises the fourth phase of interaction. Here, the issue is to design a representation that communicates the results in such a way that they are both mathematically correct *and* understandable by the end-user. For most DSSs published in the literature, the results are communicated by simply returning the probabilities or providing an ordered list of the most likely outcomes (e.g., McKendrick et al., 2000; Onisko et al., 1999); some have attempted to improve on this by color-coding, e.g. in the TraumaSCAN-web project (Ogunyemi et al., 2006).

Lastly, in phase five, the results will need to be explained and justified. Making it clear to the end-user how the model came to its conclusions, justifying the conclusions, and clarifying what the limitations of the model are in this is a challenge that has been recognized by others (reviewed by Lacave and Diez, 2002). A summary of the five phases can be found in figure 1.

Phase	*Representations*	*Translations*
1	repr. for knowledge elicitation interactions with experts	translation from expert knowledge to model content and structure
2	repr. for model evaluation interactions with experts	translation from model content and structure to user language
3	repr. for data entry by the user	translation of evidence needed to user language
4	repr. for dissemination of outcomes to users	translation of computational outcome to user-compatible format
5	repr. for explanation and justification of the outcome to users	translation of model processes to user-compatible format

Figure 1: The Five Phases with Example User-Centered Representations and Knowledge Format Translations

To the best of our knowledge, no systematic approaches on how to prevent knowledge transfer errors from developer to end-user and vice versa have been published. In this paper, we are introducing such an approach in order to prevent knowledge transfer errors by improving the representations used on the interface to the end-user and creating awareness of needed knowledge translations. We look at the probabilistic DSS, so to speak, 'through the users' eyes'. We introduce heuristic guidelines based on user-centered design principles specifically targeted to prevent knowledge-transfer errors between the developer and the expert or end-user during different phases of the construction of the probabilistic network and the design of the DSS. We distilled these guidelines from our practical experience and from human-computer interaction theory. These five guidelines comprise preserving the precision of the probabilistic information, using language and a workflow that is compatible with the user's profession, using natural language, hiding difficult to understand technology and making the system as efficient as possible.

The outline of the paper is as follows. First, we explain the likely sources for the knowledge-transfer problems in human-computer interaction theoretical terms. Next, we introduce the five heuristic guidelines that proved especially important for the design of a non-probabilistic representation to a probabilistic DSS. In sections three and four, we will elaborate on phases one and three because our representation work to date has concentrated on these two phases. For each of these phases, we will describe the interaction between the developer and the expert or the end-user, explain the knowledge transfer errors that may occur, show solutions generated by others and ourselves where possible, and show how and where these solutions fit in with our guidelines. Finally, concluding observations will be made and we will outline possibilities for future work.

2 THEORETICAL BACKGROUND: CAUSES OF KNOWLEDGE TRANSFER ERRORS

Below, as an example, we will use a physician as a typical type of end-user, who is to use a diagnostic DSS. However, the same case could be made for lawyers, psychologists, veterinarians, most managers, most non-mathematical academicians etcetera, in combination with diagnostic or other types of DSSs. We briefly lay out exemplars of goals and tasks that the physician wants to accomplish with a DSS and compare these to those of academic developers of a DSS want to accomplish. We also describe the differences in the knowledge of the developer and the physician, and explain why this makes such a big difference in interpreting the classical probabilistic network representation. We will argue that a complete redesign of the representations (be they, for the different interactions, put on paper or a computer) is needed for those phases in which interaction of a developer and the physician occurs. We will also argue that the current representation can and should be maintained for the developers' tasks.

2.1 GOALS, TASKS AND KNOWLEDGE BASES

Physicians are trained as professionals who apply their medical knowledge to diagnose and treat disease. A knowledge base (a cognitive science term) consists of what a person learns and what they experience. Physicians' knowledge base consists of coursework from their academic training, centered on improving the health of a human being: anatomy, biochemistry, disease patterns, medications, etcetera. No probability theory. In addition, their knowledge base contains practical experience, for instance patient-doctor interactions, diagnoses, treatments, and doctor-doctor consults, much of it in case formats.

Physicians' primary goal is to improve patient care. They realize that a DSS may help them to avoid wrong decisions that can at best delay correct treatment and at worst kill the patient. Tasks they seek help with in a DSS are diagnosis and treatment decisions. They seek something akin to a competent colleague, who offers a second opinion and who can explain in understandable terms how he came to the opinion. Physicians also want a tool that either saves them time, or, if it takes time, offers another benefit that they are convinced weighs up against the invested time. In addition, it is likely that physicians will want the DSS to generate explanations, on demand, that help them understand the computational outcomes.

The people who develop medical DSSs, on the other hand, are generally PhD-level computer scientists. Their academic training centers on improving the workings of a computer by improving the software. Training involves, among others, mathematics, logics, computer programming, and algorithm design. No medical knowledge. As scientists, they also possess much practical knowledge on how to do research.

The primary goals of academic developers in constructing probabilistic-network-based DSSs is to push the state-of-the-art: to solve challenging and novel computational problems and problems underlying the construction of probabilistic networks, and to publish the results in their scientific community. These scientists also derive much pride from delivering a high-quality, highly accurate system. To avoid having to bear responsibility for incorrect decisions, developers also want to help physicians to understand the limitations of the tool. For example, its knowledge base may not be complete.

Tasks within these goals are to select a probabilistic-network to construct that offers an interesting computational problem, and next to solve the problem and develop a robust probabilistic network with the aid of the knowledge of the domain experts. Developers then need to construct a way to make data-entry possible so that the network can compute an outcome to be communicated to the user.

It is clear that the goals and tasks of the two types of people, developers and physicians, do not overlap much. Aside from both wanting a model that works with a high accuracy, their goals and tasks are nearly totally different. Their knowledge base and *mental model*, at least where it concerns the DSS, also are completely different. A mental model, another cognitive science term, is an explanation in someone's thought processes of how something works or is put together in the real world. The developer is expert in the computational part, the physician in the content part.

The problem lies in the fact that a person always interprets that which he perceives through the filter of his own knowledge base, as if it were through different glasses. One's knowledge base 'colors' all incoming information, because this information is immediately enriched by, and interpreted with the help of the existing knowledge. The brain does this automatically, and one cannot 'turn off' the existing information in one's knowledge base (Anderson, 2004). When a person *recognizes* a piece of incoming information, it was already present in the knowledge base; that knowledge is then activated. If there is knowledge associated with that information, it too will be activated, somewhat like circles spreading from a stone thrown in a pond. The result of this is that the designer constructs a system based on his 'design model', while the user interprets a system on his 'user's model' - almost in any application there exists a large gap between the two (Norman, 1988). In the next section, we look at the common probabilistic network representation through the user's and through the developer's glasses.

2.2 SWITCHING GLASSES FROM DEVELOPER TO USER

Physicians, when looking at the common probabilistic network representation, see a large number of very familiar medical terms, each in a little box connected by lines to other boxes. They certainly do not 'see' a probabilistic network, because there is nothing in their knowledge base that can recognize, identify or make sense of a probabilistic network. In each box are also some numbers to be seen. The vast majority of physicians will not be conversant with probability theory, and will therefore not be able to interpret this information. Even if they understand probabilistic terms, they will not understand them to the deep level that developers will understand them. Physicians interpret the edges as diagnostic lines, pointing from symptoms and test outcomes to disease diagnoses. To them, the arrows are pointing in the wrong direction. Physicians know which data go with which medical term, and know whether the data they enter are valid or not, and whether those are within a normal range or not. Physicians cannot see whether computed changes in the network make sense because those are expressed in probabilistic terms. Yes, they perceive that the numbers are changing, but attaching meaning to that is not supported by their knowledge base. Trying to carry out medical tasks within such a foreign representation is extremely difficult and time-consuming. Physicians need a representation that they can recognize, interpret and react to on basis of their own knowledge base. Because their knowledge base is much different from the developers', such a representation will necessarily differ completely from the common probabilistic network representation.

Now we put on the glasses of the developer. Developers see a probabilistic network with nodes and edges; each node has probabilities associated with it. Developers' knowledge base allows them to automatically and quickly recognize and interpret the edges as interrelations of the variables. Developers probably do not know *exactly* what is really meant with particular terms supplied by the domain expert. Developers do not know whether entered patient data are valid (e.g., whether a test's outcome numbers makes sense), normal or abnormal. Nor do they have any knowledge to interpret the content of the outcome. For instance, what is the difference between grading and staging a cancer? And even if they understand the outcome, they will not understand it in the deep way that expert physicians will understand them, immediately realizing all the consequences of the diagnosis in terms of treatment options, life expectancy, etcetera. One of the real pitfalls is that developers, while eliciting the experts' knowledge of the domain, may start to feel that they understand the domain, and may even fill in information that the expert has not explicitly communicated. This false feeling of understanding has been coined 'shallow understanding': people feel they understand something until they are confronted with conflicting evidence, either from their own knowledge or from incoming information (Tabachneck, 1992). The only part of the model that the developer deeply understands is the computational part. With the help of the common probabilistic representation, the developer can see whether the network propagates changes in probability values, but not whether either the original values or the computed changes make any medical sense.

Above, we have clarified why different goals, different tasks and different knowledge bases require a different view on the data: a different representation. The representations meant for the end-user should be designed so that the end-user understands them; in cognitive science terms, so that they align with their mental model of their domain knowledge and task execution. This means that developers, who have the knowledge to interpret and use common probabilistic network representations, should probably continue to use them if they are the end-users. This also means that other types of users should be provided with representations that suits their knowledge base and mental models.

We hope that we have also motivated why constructing user-centered representations requires frequent and intensive contact with the user. Only the actual users can interpret developed representations with their specific glasses. It is not helpful for developers to try to interpret developed representations as if they were the user, because they have different glasses, and those glasses are not removable. It would take a minimum of ten years of medical training and practice to acquire medical-expert glasses.

We now switch to the part of the paper where we outline our systematic approach to developing user-centered representations. The core of the approach lies in applying five heuristic guidelines that we found particularly useful for constructing user-centered representations for DSSs in each phase of contact with domain experts or end-users.

3 HEURISTIC GUIDELINES FOR DESIGNING USER-CENTERED REPRESENTATIONS

Although other human-computer guidelines also apply, we feel that the five presented below proved especially relevant to the changes that have to be made to align information to a particular knowledge base and mental model. The first heuristic was coined by us. The other four are heuristics well-known in human-computer interaction theory.

(H1) *Preserving Precision*: The first guideline is specific to designing representations for probabilistic models. It concerns the need to preserve the precision of the stochastic variables used in the model. The problem is that physicians don't think in precisely defined stochastic variables but in hazy categorical concepts. The error in knowledge transfer occurs here because, when presented with a variable name, physicians see a categorical concept and the developer sees a precisely defined stochastic variable. As a consequence, great care must be taken to translate the model's terms to new terms, so that the precise definition is understood exactly as precisely by the physician. Otherwise, for instance, data entered will not be correct and the probability returned will not be correct either. Similarly, the physician will enter the data according to what he knows. There is a good chance that this will not precisely overlap with what the model wants to know. Another translation yet may be needed to align the entered data with the model's precise terms. In other words, preserving precision is of vital importance.

(H2) *User Compatibility*: The second guideline states that the information presented in the interface should align with the user's mental model. 'Speak the user's language' (Nielsen, 1993), 'know thy user' (Mayhew, 1992), and 'workflow compatibility' (Mayhew, 1992) are three well-known design heuristics that bear on this guideline. This guideline concerns changes that are specific to one type of user only, for instance, physicians or lawyers. In practice, these heuristics mean to use the actual professional language of the physician in the presented information and not the language coined by the developers; they further mean to fit the information flow to the usual workflow of the physician.

(H3) *Natural Language*: The third guideline is to follow the principle of 'simple and natural dialogue' (Nielsen, 1993). This guideline applies to any type of user, and means that the information presented to users should as much as possible resemble simple, natural language and natural types of reasoning as used within the user's culture. Probabilistic language has little overlap with simple, natural language, again because of the precise definition of each term in probability theory and the categorical and conceptual nature of words in common natural language. Think, for instance, of the word 'table', and now think of the many types and shapes of tables that you know, which all fall within your concept of 'table'. However, a Chinese will have a different concept of table than a Dutchman, even though they will both use the same word if they speak English together.

(H4) *Invisible Technology*: 'Effective interfaces do not concern the user with the inner workings of the system' (Tognazzini, 2003), or 'invisible technology' (Mayhew, 1992), constitute the fourth guideline. In the case of DSSs, the technology consists of the probabilistic network. It is a good idea to hide anything having to do with probability theory, and replace it with something that users normally use within the execution of their tasks, while preserving precision of course.

(H5) *Efficient System*: Lastly, the application should be as efficient as possible. 'Look at the user's productivity, not the computer's' (Tognazzini, 2003) and 'Ease of learning and ease of use' (Mayhew, 1992) are underlying design heuristics. This guideline is important because the users of DSSs are demanding users whose time is precious. The faster and more conveniently they can interact with the system, the better they will comply with the system's demands and the more they will use the system.

In the next two sections we will elaborate on phases 1 and 3 of the interaction with the domain expert or the end-user and explain how these guidelines were applied in our and others' research. In both phases, examples of tasks will be given, types of interactions within these tasks will be specified, and potential sources of knowledge transfer errors will be identified. We will present examples of solutions from our or others' research, but these are merely examples meant to inspire and show how the guidelines could be implemented, as there is not just one correct or best solution. Good solutions will depend on the application and the particular type of user.

4 PHASE 1: KNOWLEDGE ELICITATION

The developer hands the domain expert physician a piece of paper. The paper says: $Pr(\text{Invasion} = \text{T2} \mid \text{Shape} = \text{polypoid}, \text{Length} < 5\text{cm})$. *It takes the developer a good 10 minutes to explain to the physician what exactly it says on the paper. Finally, the physician provides the needed information. After the tenth explanation of what is on the paper, the developer is desperate with the slow progress of the physician. At this pace, she will have to interview the expert for weeks.*

During the knowledge elicitation phase, the developer mainly interacts with the domain expert.

Example 1: Determine the graphical network structure.

Interview domain experts about relevant processes in the domain in order to identify the relevant variables and the values they may adopt. Error sources: (1) Experts dutifully explain the medical processes, but uses medical language that is perfectly understandable by them but very difficult to understand by the developer; (2) developers may start to feel that they understand the domain and fill in unelicited information. The guideline to be applied for the first problem is 'natural language'; also, it helps to develop a sensitivity to the communication problem. Usually, the expert slowly learns to tune explanations to a more natural language; the developer slowly learns to interpret some medical terms. The expert hardly ever makes progress towards understanding probability theory; in this way the process is asymmetric. To entice the physician to phrase his knowledge in 'natural language' terms, a few hints may be given, for instance asking the physician to phrase the explanation as if he were explaining things to his mother, giving out a warning that the elicitation process will take much time, and frequently indicating that you, as developer, do not understand what the expert says. The developer should never make the mistake of filling in information based on his own acquired medical knowledge. Remember, it takes 10 years to acquire the deep knowledge of the expert! One helpful process to check your own assumptions is to provide feedback to the expert in the form of his own statements, translated to include the precision needed for the model. For instance, the physician may have said: “If more symptoms are present, the stage of the cancer is worse”, which could be translated to: “If more symptoms are present, it is ALWAYS the case that the stage of the cancer is worse”. If the developer does not recognize the error sources in this phase, the processes modeled in the network will not mirror the real processes.

Example 2: Take the information elicited from the physician and extract the needed variables and values.

Discuss with the physician whether a variable the developer has defined still covers the concept the physician has defined. Error source: physicians think in often vague categorical concepts, whereas the network needs precisely defined stochastic variables. The guidelines to be applied here are ‘preserving precision’ and ‘natural language’. Aligning the two different concepts, one probabilistic and one medical, takes careful negotiation. In our experience, this may take much iteration. Solutions lie in trying to understand the other’s concepts by re-representing them, for instance by defining the concept’s boundaries, drawing, or making a table. Another solution is to use the guideline ‘user compatibility’ by entering the physicians’ world and observing the symptoms and test results in the way the physician would. For instance, the developer might ask to see a series of X-rays of different developmental stages of a tumor. In this way, the developer may be able to come closer to understanding the physician. If the expert is unable to recognize stochastic precision in a variable name, and the developer cannot compensate for that, and, vice versa, the developer cannot see the conceptual nature of a physician’s term, the developer and the expert will be talking about two different concepts with all consequences thereof for the model.

Example 3: Assess the interdependences of the variables.

During interviews, experts are asked about interdependences. Error source: physicians do not think in these terms, it does not fit within their mental model of their task execution (Van der Gaag & Helsper 2002). These researchers substitute asking about interdependences (the technology) with the notion of causality, compatible with physician’s reasoning, which results in questions like “What could cause this symptom?” and “What manifestations could this disease have?” This follows our guidelines ‘invisible technology’ and ‘user compatibility’.

To check the interdependences, the developer will sometimes present the expert with a small piece of the network. Error source: experts cannot understand the network structure. As Van der Gaag en Helsper (2002) remarked: “Unfortunately, however, the experts often appeared to misinterpret the structure.” In their solution they applied the guidelines ‘user compatibility’ and ‘invisible technology’. Physicians are very familiar with case descriptions. These researchers therefore replaced network structures (the technology) with case descriptions (reasoning compatible with physicians’ mental model), e.g.: “Suppose that you have a patient with a high fever, and you have made an assessment of the patient’s muscle aches. Can knowledge of the presence or absence of a headache change your assessment?” Schreiber et al. (2000) and Scott et al. (1991) have suggested a similar approach.

Example 4: Assess the probabilities.

Users are provided with probabilities to be assessed in probabilistic notation, for instance $Pr(\mathrm{Invasion} = \mathrm{T2} \mid \mathrm{Shape} = \mathrm{polypoid}, \mathrm{Length} < 5\mathrm{cm})$. Error source: this format is not understood by the user. Moreover, van der Gaag et al. (2002) reports about their experiences: “Especially the meaning of what is represented on either side of the conditioning bar appeared to be confusing, and in fact remained to be so during successive interviews.” Other methods used here are probability-scale methods, gamble-like methods, and probability-wheel methods. Error source: those representations were shown to be not well-understood by the user either. Van der Gaag et al. (2002) tested the first two of these methods. Regarding the probability scale, users commented that they felt uncomfortable working with it, that there was very little to go by. User answers were also biased by the ‘spacing effect’: they placed their marks unconsciously to that they looked attractively spaced. The gambling method was also ineffective and very demanding. Users found that the specific lotteries were very hard to conceive of because of the rare or unethical situations they presented. The gambling method does a little better job at ‘hiding technology’, but it is not compatible with the users’ mental model of their usual tasks and experiences. The authors remark: “... the gambling appeared to be rather demanding on the experts. Apparently, it deviated substantially from their usual cognitive processes.” Also, both methods were found to be time-consuming, which runs counter to our ‘efficient method’ guideline. A brief overview of the probability-wheel method was provided by Renooij (2001). She stated that this method tends to be time-consuming, and is not suitable for assessing very large or very small probabilities as the users cannot perceive the difference. The solution proposed by van der Gaag et al. (1999, 2002) is to design a new method, based on transcribing probabilities in case-like terms (guidelines: ‘user compatibility’ and ‘hiding technology’), accompanied by a scale in common, well-understood reasoning terms as *certain*, *probable* and *fifty-fifty* and their associated approximate percentages (guideline 'natural language'). This method proved to be efficient as well, enabling the researchers to elicitate 150-175 probability estimates per hour.

Still, physicians feel uncomfortable with the use of probabilities in numerical form (von Winterfeldt & Edwards, 1986). To reduce the number of probabilities to be assessed, Helsper et al. (2004) suggest an extra step just before assessing probabilities, namely to first establish the qualitative pattern of interaction for each causal mechanism. These patterns can then be used as constraints on the precise probabilities to be obtained. By substituting natural reasoning for math concepts, this follows the 'invisible technology' guideline as well as the 'efficient system' heuristic because not only does the task become more natural and thus much easier to carry out, fewer of the exact probabilities will have to be elicited.

5 PHASE 3: DATA ENTRY

Surreptitious laughter sounds from the room. The users have just spotted the probabilistic network underlying their DSS. Their expressions show disbelief and lack of understanding, even though we have just explained what a probabilistic network is and does and all have indicated their understanding of our explanation.

In phase 3, the developer mainly interacts with the end-user. Examples exist of data-entry modules that have been designed on user-centered principles (e.g. Long et al., 1996; Hernando et al., 2001) but clear heuristic guidelines have not been identified.

Example 1. A data-entry module is needed for a user to provide evidence.

The developer supplies the user with data entry based on the network representation. Error sources: (1) the user does not understand the network representation; (2) the variable names used will not be understood by the user as precisely defined stochastic variables but, at best, as concepts. For the first problem, apply 'invisible technology' and 'natural language' guidelines. In Tabachneck-Schijf et al., (submitted), we describe a questionnaire representation we designed. According to Gliner & Morgan (2000), questionnaires are particularly useful for quantitative research, especially in comparative and descriptive approaches. Moreover, with a questionnaire one can specify a choice of possible answers corresponding with the values in the network. The questionnaire was implemented in our application as keywords with associated pull-down variable value boxes. Open questions should be avoided as this can introduce more translation errors, e.g. the developer is faced with user language, or the answer is not among the values in the network. In addition, open answers require keystroke entries, much slower than clicking, which violates our 'efficient system' guideline. In Hernando et al. (2001), the guidelines 'hiding technology' and 'user compatibility' were also mainly used. Diabetic users needed to enter data at each mealtime on blood values and meal contents. Their solution was similar to ours, a data-entry module consisting of pull-down boxes to choose values. The boxes were arranged horizontally on a time axis, which agreed with the users' mental model of timeflow in a day.

For the second problem, misunderstanding stochastic values as concepts, Tabachneck-Schijf et al., (submitted) applied the 'user compatibility' and 'preserve precision' guidelines. Their solution was to ask the users to define their own terms that, in their professional language, covered the precise definition of the stochastic variables.

Example 2: A physical means for entering evidence has to be provided.

The developer constructs a program on a computer or hands the user a form to be filled out. Error source: neither means of entry may be suitable to the users' work environment or the work flow. Solutions: Apply the 'user compatibility' guideline. Consult the user as to the circumstances under which they need to carry out the data entry; discuss ways to circumvent problems. In a case where the users needed to collect evidence in situ in a pig unit, we decided to put the data-entry module on a personal digital assistant, which can easily be carried into a pig unit and be covered by sturdy, well-closed plastic bags which can be discarded so that the device cannot be contaminated (Tabachneck-Schijf et al., submitted).

Example 3: The data-entry questions need to be ordered.

The user is confronted with an ordering that is constructed by the developer from the model's viewpoint. Error source: this ordering may not fit the user's workflow or mental model of the task. This will slow down users, even if they can reply to questions at random, because they will have to search for the next question to answer. Again, the 'user compatibility' guideline can be applied: consult the user. For instance, in Tabachneck-Schijf (submitted), we needed small groups of up to five questions. The users were given cards with the questions printed on them. They were asked to group cards together, then to sort the groups in an order that fit in with their normal workflow, to next order the questions in each group similarly, and then to write down their grouping rationale. Two orderings ensued; by examining the frequency by which two questions were grouped together, we arrived at clusters which best fit the orderings and also fit our limited interface space. In Kappen et al, 2002, the developers developed a type of questionnaire. The questions of the actual questionnaire part are grouped, but it is not clear what the researchers used as a basis for the ordering the data-entry part.

To further improve the efficiency of the system, Tabachneck-Schijf et al. (submitted) also attempted to minimize the number of clicks the users had to carry out, as well as keystroke input. Keystrokes were limited to entries that could be carried out either before or after the in-situ visit to the pig unit, where the PDA could be handled more easily without its protective covering.

6 CONCLUSIONS

In this paper, we have begun to identify a systematic approach to improve those representations on the interface of a probabilistic DSS that are meant for the domain expert and for the end-user. To this purpose, we have presented five heuristic guidelines we identified as being particularly effective in guiding our own representation designs. Most of these guidelines are meant to (1) increase awareness of the differences between the developer and the non-mathematical user or domain expert; (2) show the necessity of aligning representations intended for the user to the users' mental model; and (3) support the need to translate user information into model information and vice versa, before and after user-developer interactions have taken place. One guideline underscores the need to make the application efficient if the end-user is a demanding user. The crux of our method is to look at each interaction representation and all interaction information and see whether they violate one of our heuristic guidelines. If so, the representation needs to be altered, in the vast majority of cases by interacting with a domain expert or an end-user. We feel that following this advice can prevent many knowledge-transfer errors.

These guidelines can be applied during the entire development process of a DSS, from knowledge elicitation to explaining and justifying the outcomes of the probabilistic model. Four of the guidelines originate from commonly known human-computer interaction theory. One guideline, preserving precision, was coined by us and is specific to mathematical modeling. It concerns the problematic difference between rather hazy user concepts and precisely defined stochastic variables, and targets the translation from user's model to design model and vice versa.

To help us to systematically identify the different moments where better interactions with the user may result in considerable improvements to the model's contents and use, we divided the development process into five phases. In each phase several interactions are distinguished. The application itself and the type(s) of users determine which interactions are needed; these may differ from the ones we specified. The interactions mentioned in this paper reflect the experiences of our research group in developing (bio)medical DSSs.

We illustrate the use of our systematic approach of applying the guidelines in each phase by explaining the cause of the knowledge transfer errors and giving examples of user-centered solutions. In essence, knowledge transfer errors are caused by the large differences between what is in the head of the developer and of the user. Training and commonly executed tasks do not overlap and result in knowledge bases that are in general completely different. As incoming information immediately mixes with and is interpreted by the existing knowledge base this is as if different types of professions are wearing different glasses. When looking at the same information they 'see' something different. For instance, what either 'sees' in a medical record and a classical probabilistic network representation is very different.

We hope that this paper will initiate a fruitful discussion about the positive results of involving the user in a *meaningful* interaction with the developer throughout the entire development process. Applying the heuristic guidelines may well lead to a further dissemination of probabilistic decision support systems.

With our growing awareness of the issues in knowledge transfer, and our successes thus far, we will certainly continue to firm up and apply our systematic approach, also for interactions that occur within phases 2, 4 and 5.

Acknowledgements

This research was (partly) supported by the Netherlands Organisation for Scientific Research (NWO). The discussions with Linda van der Gaag have been invaluable to the development of this paper. We also thank our domain experts and veterinary users for gladly supplying all the information we requested.

References

Anderson, J.R (2004). Cognitive Psychology and its implications, 6th ed. Worth Publishers, New York, NY.

Gaag, L.C. van der, Helsper, E.M. (2002). Experiences with modelling issues in building probabilistic networks. In A. Gmez-Prez & V.R. Benjamins (Eds.), *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, Proceedings of EKAW 2002*. Springer-Verlag, Berlin, Germany, pp. 21-26.

Gaag, L.C. van der, Renooij, S., Witteman, C.L.M., Aleman, B., Taal, B.G. (1999). How to elicit many probabilities. In K.B. Laskey & H. Prade (Eds.) *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, California, pp. 647-654.

Gaag, L.C. van der, Renooij, S., Witteman, C.L.M., Aleman, B.M.P., Taal, B.G. (2002). Probabilities for a probabilistic network: a case study in oesophageal cancer. *Artificial Intelligence in Medicine*, vol 25, pp. 123-148.

Gliner, J.A., Morgan, G.A. (2000). *Research methods in* *applied settings: an intergrated approach to design and analysis*. Lawrence Erlbaum Assoc., Mahwah, N.J.

Helsper, E.M., Gaag, L.C. van der, Groenendaal, F. (2004). Designing a procedure for the acquisition of probability constraints for Probabilistic networks. In: E. Motta, N.R. Shadbolt, A. Stutt, & N. Gibbins (Eds.). *Developing Knowledge in the Age of the Semantic Web, Proceedings EKAW 2004*. Springer-Verlag, Heidelberg, pp. 280 - 292.

Hernando, M.E., Gomez, E.J., Del Pozo, F., (2001). Educational tool for diabetic patients based on causal probabilistic networks. In: S. Quaglini, P. Barahona, & S. Andreassen (Eds.): Aime, LNAI, pp. 203-206. Springer-Verlag Berlin, Germany.

Kappen H.J., Wiegerinck W.A.J.J., Braak E. ter (2002). Decision support for medical diagnosis In: J. Meij (Ed.) *Dealing with the data flood. Mining data, text and multimedia*. STT/Beweton, The Hague, The Netherlands, pp. 111-121

Lacave, C., Diez, F.J. (2002). A review of explanation methods for Bayesian networks. *The Knowledge Engineering Review archive*, Volume 17, pp. 107-127

Long W.J., Fraser H.S.F., Naimi, S. (1996). A web interface for the Heart Disease Program. In: *Proceedings of the AMIA Fall Symposium*. Hanley and Belfus, Washington, DC, pp. 762-766.

Mayhew, D. J. (1992). *Principles and Guidelines in Software User Interface Design*, Prentice Hall, Englewood Cliffs, NJ.

McKendrick, I. J. , Gettinby, G., Gu, Y. , Reid, S.W.J., Revie, C.W. (2000). Using a Bayesian belief network to aid differential diagnosis of tropical bovine diseases. *Preventive Veterinary Medicine*,. vol. 47, pp. 141-156.

Nielsen, J. (1993). *Usability Developing*. Academic Press, San Diego, CA.

Norman, D. (1998). *The Design of Everyday Things*. New York: Basic Books.

Ogunyemi, O., Rice P., Clarke J., Matheny, M., investigators (2006). TraumaSCAN-web project. *http://dsg.harvard.edu/ oogunyem/traumascan/*.

Onisko, A., Druzdzel, M.J., Wasyluk, H. (1999). A Bayesian Network Model for Diagnosis of Liver Disorders. Technical report CBMI-99-27,Center for Biomedical Informatics, University of Pittsburgh.

Renooij, S. (2001). *Qualitative Approaches to Quantifying Probabilistic Networks*. Ph.D. Thesis, Institute for Information and Computing Sciences, Utrecht University, The Netherlands.

Schreiber,G., Akkermans, H., Anjewierden, A., Hoog, R. de, Shadbolt, N., Velde, W. van de, Wielinga, B. (2000). *Knowledge Engineering and management: The CommonKADS methodology*. MIT Press, Cambridge, MA.

Scott, A.C., Clayton, J.E., Gibson, E.L. (1991). *A practical guide to knowledge acquisition*. Addison-Wesley, Reading, MA.

Tabachneck-Schijf, H.J.M., Geenen, P.L., Van der Gaag, L.C., Schrage, M.M., Van IJzendoorn, A. Loeffen, W.L. A., Elbers A.R.W. (Submitted). Towards user-centred interfaces for probabilistic decision-support systems: a case study.

Tabachneck, H.J.M. (1992) (currently Tabachneck-Schijf). *Computational differences in mental representations: Effects of mode of data presentation on reasoning and understanding*. Doctoral Dissertation. Carnegie Mellon University, Pittsburgh, PA.

Tognazzini, B. First principles of interaction design. *http://www.asktog.com/basics/firstPrinciples.html*

Winterfeldt D. von, Edwards W. *Decision Analysis and Behavioral Research* (1986). Cambridge University Press, Cambridge, UK.