ABSTRACT

Providing expert system users with informative explanations is today considered almost as important as giving them correct solutions. Yet, a majority of explanation systems were limited to the inspection of a more or less understandable trace of fired rules. This was detrimental to the quality of explanations since informations were given in a way that depended highly on the implementation rather than on the expertise itself. The only exception to this was the use of canned text where instances of possible explanations had to be anticipated. This paper describes a method for automatically assembling explanations expressed in the expert own terms which is yet more general than mere canned text techniques. The method is based on the constitution of an explanatory knowledge base written in a special purpose language and allowing the assembly of explanations by means of a full fledged reasoning conducted on the expert knowledge base as well as on session traces.
A Study of the Knowledge Required for Explanation in Expert Systems

C. Millet
EUROSOFT Systèmes
Route de Tréguier
22300 Lannion, France
millet@lannion.cnet.fr

M. Gilloux
Centre National d'Etudes des Télécommunications
department LAA/SLC/AIA
Route de Tréguastel, B.P. 40
22301 Lannion Cedex, France
gilloux@lannion.cnet.fr

ABSTRACT

Providing expert system users with informative explanations is today considered almost as important as giving them correct solutions. Yet, a majority of explanation systems were limited to the inspection of a more or less understandable trace of fired rules. This was detrimental to the quality of explanations since informations were given in a way that depended highly on the implementation rather than on the expertise itself. The only exception to this was the use of canned text where instances of possible explanations had to be anticipated. This paper describes a method for automatically assembling explanations expressed in the expert own terms which is yet more general than mere canned text techniques. The method is based on the constitution of an explanatory knowledge base written in a special purpose language and allowing the assembly of explanations by means of a full fledged reasoning conducted on the expert knowledge base as well as on session traces.

1. INTRODUCTION

The ability of expert systems (ES) to give explanations of their results and of the reasoning leading to those results is considered as one of the main advantages of these systems, as compared to usual programs. In rule-based ES, explanations are often confined to a trace of the program execution. A trace is a record of fired rules. It may also include the data which allows these firings, cast into some readable form, preferably in natural language. In some approaches, a distinction has been made between why and how explanations which correspond respectively to the reasons that led the program to choose a particular rule and the results obtained by the firing of that rule. The need for why-not explanations to justify the non-firing of a rule was also suggested.

* This work was done under a CIFRE contract at the Centre National d'Etudes des Télécommunications, dept. SLC/AIA, Lannion, France.

2. CURRENT APPROACHES TO EXPLANATION

Two different approaches to explanation have been proposed: tracing the rules fired to reach the solution, sometimes making their text more readable, or employing "canned text" which consists in anticipating questions and writing the correct answers in advance. Some approaches deal with the problem in different ways. Many of them only focus on the form of the answer. For example, Weiner [1] studies the structure of an explanation and how the explanation should be given so that it will be clear and concise. Eriksson [2] tries to provide clear answers by transforming a proof tree, excluding some non-informative paths and reducing some others. Wallis and Shortliffe [3] insist on a model of the user in order to provide explanations tailored to his level of understanding.
Another research perspective focuses on the information content of explanations itself, independently of their form. Hasling [4] wrote the Neomycin system to improve Mycin and provide explanations which would be more than just the text of rules by using meta-rules that can describe the expert strategic knowledge. Clancey [5] also developed the Guidon system to teach the knowledge of Mycin to medical students. While writing this tutorial system, it was discovered that the knowledge of Mycin was not sufficient and not enough explicitly written to help in teaching.

In his XPLAIN system, Swartout [6] used an "automatic writer" to produce object systems in which some kind of deep model of the expertise would be compiled into a shallow form, suitable to efficient problem-solving and still linked to the deep model in order to be able to answer question about the justification of some result. This method certainly makes available the knowledge needed for generating good explanations. Still, it is very demanding for the expert or knowledge engineer since it requires the availability of a theory of the domain and a suitable language to describe it.

One of the most difficult problems in the explanation domain is to answer why-not questions concerning facts that were not inferred by the ES [7] [8].

To give more substance to the answers and make them more homogeneous, Kassel [9] proposes a "conceptual model" of the domain which explicitly represents the objects on which the system is reasoning and the concepts structuring its knowledge of problem solving. This model is then used by the explanation system to think about the system reasoning. In a similar manner, Dieng et al. [10] describe a frame-based representation to express the explanatory knowledge of a system and its different levels.

Our criticism to these approaches of explanation is that a majority of them rely on system terms rather than terms relevant to the problem domain. Exceptions are the cases of canned-text, which is too ad hoc in most situations, and of the use of a domain theory, which is not always available, at least at the onset. However, it is clearly desirable that a single expertise, would give always the same solution to the same problem and also the same explanation to the same question whatever the language used to implement it. One way to achieve this goal is to build explanations by classifying and exploiting particular forms taken by the expertise.

3. NECESSARY KNOWLEDGE

We studied an ES knowledge base [11] to find how we would have best explained the knowledge encoded into its rules. We discovered that the way of coding the expert knowledge had a precise and useful meaning that is needed for assembling informative explanations. We call the knowledge about how the coding of the expertise is related to its meaning the necessary knowledge. Doing this, we tried to identify various types of knowledge patterns that would not be too specific to the implementation language but could also be found in other systems. In order to overcome the danger of being to specific to the chosen language, we considered other systems, in different domains as, for example, McDermott R1 [12] whose success and complexity are well known.

3.1 Missing knowledge

Obviously, as for any other system, a program can only rely on its knowledge to provide explanations. For several reasons, the knowledge found in the system base rules is often not sufficient to generate explanations. An ES is a computational system designed to meet time and storage requirements. Information which takes room and time to process but does not improve the quality of solutions is usually not included. For example, causal relations like "if A then B and if B then C" may be written as "if A then C" without any loss of accuracy but it is obvious that a level of detail which is significant to the explanation process is lost here. Would this level of detail be absent, the corresponding explanations will never be given back later. We call this type of information, i.e. the information omitted for not being useful in the search of a solution, the missing knowledge.

A well-known example, pointed at by Clancey [5], is featured in Mycin where a rule like:

\[
\text{If the patient is less than 8,} \\
\text{then do not prescribe tetracycline}
\]

can be found. One can see a causal relation between both parts of the rule, but these assertions are part of a more detailed causal chain developed by Clancey, (see figure 1), which binds them together.

\[
\begin{align*}
\text{Tetracycline in youngster} & \\
\downarrow & \\
\text{chelation of the drug in growing bones} & \\
\downarrow & \\
\text{teeth discoloration} & \\
\downarrow & \\
\text{do not administer tetracycline}
\end{align*}
\]

**Figure 1.** A causal chain

3.2 Implicit knowledge

The other sort of knowledge necessary for the explanation exists but is scattered throughout the system. It is the knowledge embedded in the implementation of the expertise. It may also be implicit to the expert mind. The expert may not be fully aware of all the information he puts in the system and especially of the presuppositions that constrain the rules of expertise. Therefore, we have to consider that the way something is said is as important as what it says, i.e. that the way rules are encoded using the provided language is significant. Then one has to discover the meaning of the form and how it contributes to the process of explanation.
We call the knowledge embedded in the implementation the implicit knowledge.

Implicit knowledge may also be divided into two kinds. One kind is bound to the inference engine. As such, it is independent from the knowledge base and reflects only the strategy followed by the ES. The other class depends on the expert or knowledge engineer and on his analysis of the expertise. This hidden knowledge has a general form which depends on each domain.

3.2.1 The reasoning strategy - Most rule based ES consist of two parts: a knowledge base and an inference engine. This must be taken into account when considering the generation of explanations, because the way in which rules are chosen or applied may have a relevant meaning. For example, the possible types of questions are highly influenced by the reasoning process, e.g. data driven or goal driven. A system working with backward chaining [7] [8] will give negative (why not) explanations more readily than a system working with forward chaining [12]. In the first case, for example, to explain why a fact could not be found, it is possible to start with this fact and recursively find the rules which could have led to it until the process is blocked by negated primitive facts. At this point, all the facts that should be true for the fact of interest to be inferred are identified. The explanation consists in presenting these facts or some well-chosen subset of them.

3.2.2 Clause order - The manner in which the condition in a rule is evaluated influences widely the way the rule is written. Generally, as soon as a condition term is found to be false, it is no use to go further and evaluate the next one, if the condition part is a conjunctive expression. Then the order of the terms in the condition part of a rule is not arbitrary. It may have a meaning by itself. Meaningful relations may exist between the terms. If we have a rule:

\[ \text{If } A \text{ and } B \text{ and } C \text{ then } D \]

and if we know that:

\[ \neg A \rightarrow \neg B \]

the reason for A being the first clause in the rule may just be a programming artifact used to speed up the inference process. For example, if A takes less time to compute than B, or saves from asking something to the user, it is not necessary to evaluate the rest of the rule whenever A is false. Clancey [5] calls A a screening clause. In such a case, A should not be mentioned in an explanation concerning the rule.

For example, in R1 each rule is associated to a context and can be fired only if this context is active. Each rule begins for instance with something equivalent to:

IF: THE MOST CURRENT ACTIVE CONTEXT IS CHECKING VOLTAGE AND FREQUENCY

3.2.3 The choice of the rule - The inference engine must select one rule when several may fire. How it does it is very important for the explanation process. Different techniques may be used to choose among rules in competition. This can be done, for example, after the evaluation of the condition parts of every rule. Then a strategy is employed to choose among the candidate rules. Or, rules may have priorities, the system then searches by decreasing priority, or the system may fire the first (in some specified sense) rule whose condition part evaluates to true. Thus, when writing the rule base, it is necessary to bear in mind how the choice is done since the order of rules may influence the way one is chosen for application instead of another. Saying that a rule was chosen because it was before the others in the knowledge base is obviously not an adequate explanation. Then, it must be recorded that its condition part was true and that the conditions of the preceding rules were false. With a set of concurrent rules \( R_1, \ldots, R_n \), such that each rule \( R_i \) has a condition part \( C_i = C_i, \ldots, C_i, \ldots, C_i \), when the order in the rule base is important, one explanation of \( R_k \) should use the set of conditions:

\[ C'_k = C_k \text{ and } \neg C_{k-1} \text{ and } \neg C_{k-2} \ldots \text{ and } \neg C_1 \]

3.2.4 The firing context - The result of the reasoning may be considered as a path from the data to the solution. At each step a rule is fired. The condition part of the rule is necessarily true. More generally speaking, without studying a particular problem, a rule is applied in a wider context than just its condition set. When a rule is fired, the system is in a state with some precise meaning. It is possible to find some properties which must be true before a rule may be applied.

The task-driven aspect of R1 is a good illustration of this. At each step a context is active and defines which task is being pursued. When a context is active, some facts are necessarily true. Some control rules make a context active according to some fact. For example, the rule:

CHECK-VOLTAGE-AND-FREQUENCY-1

IF: THE MOST CURRENT ACTIVE CONTEXT IS CHECKING VOLTAGE AND FREQUENCY

AND THERE IS A COMPONENT THAT REQUIRES ONE VOLTAGE OR FREQUENCY

AND THERE IS ANOTHER COMPONENT THAT REQUIRES A DIFFERENT VOLTAGE OR FREQUENCY

THEN ENTER THE CONTEXT OF FIXING VOLTAGE AND FREQUENCY MISMATCHES

activates the context

FIXING VOLTAGE AND FREQUENCY MISMATCHES
Whenever a rule is fired in this context, there are two different components that require one voltage or frequency. This fact is always true when the rules of this context can be applied.

3.2.5 Relations between terms - Knowledge can also be hidden behind the terms, objects or concepts featured in the rules. Terms may be tied by relations which often are not explicitly defined, even if they are known by the expert. For example, he may know that if the object A exists then B can not be there, without any need to write it. But to the question "why not B?", one should answer that A and B are not compatible. Such relations must also be made explicit.

The knowledge of concepts depends on the domain expert and his manner to see and represent things. The examples given by McDermott in his paper on R1 are not sufficient to know the exact relations between terms and their meaning or importance. The given names only hint at the semantics of terms and furthermore they can only make sense for a human reader.

3.2.6 Rule patterns - Rules are written using primitives of some language. They mention facts in a domain and relations between terms. They may use predicates or functions working on domain variables for example. Still, they may have a more general meaning, being in fact an abstraction of the condition to consider. Furthermore, the meaning of the base function may depend on the variables it manipulates and be different according to their relations with others.

We discovered that significant rule patterns may be identified in the studied expert systems, i.e. that we could write a general rule in which only the names of objects or attributes would be changed to generate other rules. A general explanation exists for this kind of rule and it can be customized to each particular rule by replacing variables. To find whether a rule fits a general form we look for the pattern of clauses in the condition or action parts. For example, we may say that rules following the pattern:

\[ \text{IF } \text{Cond1} \ X \ Y \ \text{and} \ \text{Cond2} \ X \ \text{THEN} \ \text{Action1} \ Z \]

deal with such and such issue.

3.2.7 Rules sequences - It has often been advertised that building an expert system is just a matter of writing down modular rules independently from each other. Alas, in a production system rule are often not independent. Rules may influence the evaluation of other rules and cannot be considered apart from the rest of the system. We already saw that priorities have a great importance. Some rules often apply only after or before some others. It is often a matter of implementation which forces to separate a chunk of knowledge which should fit only one rule into many. Then it may happen that a set of dependent rules should be explained as a whole, as shown in :

\[ R_1, R_2, R_3 \]

where rules \( R_1, R_2 \) and \( R_3 \) always apply in this order equivalently to a single rule \( R \).

4. THE DOMAIN

The illustration we have chosen is a linguistic analyzer [11] written with an ES shell called IROISE. It was elected because of its use of stereotyped reasoning strategies encoded under complex forms.

The analyzer contains a knowledge base, encoded as rules and known facts, for example the lexicon and the sentence to be analyzed, and an interpreter of these rules.

The language IROISE is somewhat reminiscent of OPS5 [13]. The rules fit the usual mold

\[ \text{If Condition Then Action.} \]

They apply to objects which represent the factual knowledge of the system. These objects are collections of attribute-value pairs. A condition is a list of object patterns which use variables as well as arbitrary Lisp expressions. The action may add, modify or remove some objects. The inference engine follows a deterministic recognize-act cycle, i.e. it looks for the rule to fire in relation with the state of the base, applies it, and starts again. An agenda scheme is used to focus on relevant rules according to their partitioning into non overlapping rule packets. At any given time only the packets in the agenda are active and only the rules associated to them may fire.

In the parser, the main objects are syntactic structures which represent the parse tree and predicative description of the sentence to be analyzed. The parsing strategy is deterministic due to the use of a lookahead window of three words or structures \( S_1, S_2 \) and \( S_3 \). \( S_0 \) is the current structure on which the analyzer is currently working. Whenever a syntactic structure is completed or provisionally abandoned, the system may read the following words of the sentence and slide the window along the sentence.

IROISE rules packets were used to represent the parser states and relative rule priorities within one state. Rules are evaluated according to these priorities within the set.

5. THE EXPLANATION SYSTEM

In our system, we focus more especially on the problem of gathering the necessary information to provide enlightening explanations. Thus, we exclude from this discussion user
models, query interpretation, the rhetorical structure of explanations, natural language generation components, etc.

The implementation follows two axes, according to the distinction made above between missing and implicit knowledge. Yet, these directions are kept complementary.

We studied the linguistic analyzer to see what was needed to succeed in providing satisfying explanations. It was decided not to change its structure to get better explanations, but to study it and improve it from outside. Thus we added explanatory knowledge to the system. Still, we believe that this approach can be applied to other systems as well. Also, this can be done during the design of the ES, while the knowledge engineer works with the expert to create the knowledge base and affects the manner in which rules are encoded, making the knowledge in them more explicit, thus more explainable.

5.1 An explanatory base

As a great part of the knowledge needed for explanation is missing, because of being of no use in the search of a solution, it must be found somewhere else and/or be added explicitly. Of course we do not want to undermine the performance of the expert system.

The method consists in building another knowledge base limited to explanatory purposes on top of the already existing rules and facts data bases without changing the reasoning behavior of the ES. This knowledge base is planned to encode all causal chains, domain term relations, presuppositions, rule patterns, etc., existing in the rule base such as conceived by the expert.

In our system all this information is represented as in the ground expert system through objects and explanatory rules dealing with these objects. The expert may write what he needs to assemble explanations provided that it fits the objects and rules paradigm.

There are two components: 1) an explanatory data base which represents the expert system to be explained and 2) the explanatory rules and facts which encode all the implicit or missing information and allow to infer explanatory facts such as the equivalence between a particular rule and some causal chain.

5.2 Explanatory meta-rules

At the onset, all we have is the ES conceived by a knowledge engineer who wrote the rules representing the expertise. For that purpose he used the production system language. He must then give the meaning of what he wrote in some other language or, better, using the same one as for expert rules. This second solution was chosen so as to keep some coherence to the system and exempt from learning a new formalism. This was also made possible by the ability of the rule language to represent the explanatory knowledge base.

The explanatory rules are in fact meta-rules, working at a different level just above the expert rules. These meta-rules are used only when an explanation is needed and do not change the expert system progress. They work on objects representing the knowledge base of expert rules (the rules of the initial ES are called expert rules as opposed to explanatory rules), a data base of explanatory facts and an object representation of the execution trace. Expert rules are encoded in such a way that the explanatory rules may manipulate them as objects.

We illustrate below our approach using the lexical analyzer. For each particular rule or set of rules we had to find their precise meaning, the reason for their existence, looking how they fit within the models we gave above. The rules were classified according to the identified types of necessary knowledge. Then the explanatory rules were written as any other expert system would be.

5.2.1 Processing of rule patterns - Studying the parser rules, we noticed that rules dealing with ambiguity had a similar form and in fact performed similar operations. This is the case of what we called previously rule patterns. Yet, the recognition of these patterns was not straightforward since some primitives, which in others rules only meant the search for some fact, were here characteristic of the detection and resolution of an ambiguity in the context of others clauses.

In the condition part of these rules, a word with several possible lexical interpretations must be found. Looking at the preceding or following words, it is possible to decide for an appropriate category for the word. In the action part, the rule removes the irrelevant categories. For example, the following rule tries to detect a lexical ambiguity between pronoun and determiner.

\[
\text{Rule Det/Pronoun-Ambiguity} \\
(P\text{Act VAct NAct}) 20 \\
\text{If } S1 \text{ is marked as a Pronoun and a } \\
\text{Determiner} \\
S2 \text{ is marked as a Noun or an Adjective} \\
\text{Then} \\
\text{Unmark } S1 \text{ as a Pronoun}
\]

An explanatory rule should reveal that this kind of expert rule deals with ambiguity so as to mention it in an explanation. The substance of this explanatory rule is:
If \( R \) is a rule
\[ \begin{align*}
R & \text{ has as condition part:} \\
X1 & \text{ is marked as a Y and a Z} \\
X2 & \text{ is marked as a W} \\
R & \text{ has as action part:} \\
\text{Unmark X1 as Z} \\
Y & \text{ and Z are not compatible}
\end{align*} \]

Then

The word X1 had two possible categories Y and Z.
The rule R treats ambiguity, it determines that X1 is a Y because Y and Z are not compatible and X is followed by W.

The condition part of this rule refers to the conditions and actions which constitute the expert rules. The objects of the explanatory base, as for example "Y and Z are not compatible", are also referred to. They are made of knowledge added by the expert, but not useful for the reasoning in the expert rules. When the explanation is given, parameters like Z, X, Y are bound to particular values and explanatory objects are added to the data base of facts, like those representing the detection of an ambiguity resolution rule. If necessary, explanatory rules may also use objects they have created to perform a complex chain of reasoning.

5.2.2 Processing of the choice of the rule - The rules in the analyzer are classified according to rule packets corresponding to different tasks and priorities. When the inference engine is looking for a rule to fire, all problems are active in the decreasing order of priorities. Each rule belongs to a packet which is related to its firing context. This packet must be active so that the first clause of the condition part may be true. For some packets, we observed that the rule with the lowest priority had an empty condition part. This rule is applied each time it is evaluated. It is a default rule that corresponds to the non-firing of the preceding ones. When one such rule gives a value to a fact, it is no use to give the rule-condition part to explain this fact. The explanation would otherwise be: "This fact has this value because it is true". We would rather want to know why the preceding rules could not be applied. Thus, an explanatory rule like the one below should verify whether a rule is a default rule.

\[ \begin{align*}
\text{IF R is a rule} \\
\text{R has as condition part:} \\
S0 & \text{ has as active packet FA} \\
\text{it is the only condition} \\
\text{Then} \\
\text{Search explanations for the rules belonging to FA with a higher priority.}
\end{align*} \]

5.2.3 Processing of the application context - As we said above, a rule must belong to an active packet to be fired. Packets are made active by the action parts of the rules, which put the name of the packet in the list of active packets or remove it from this list. It is possible to determine that when a packet is active, some facts are known true at that moment. They are at least those which allowed the firing of the rule which made this packet active. For some rules, the only action is the activation of a packet. Thus we can wrote an explanatory rule which finds the rule that activates a particular packet.

\[ \begin{align*}
\text{IF R is a rule} \\
\text{R has as action part:} \\
\text{Make the packet FA active} \\
\text{It is the only action} \\
\text{Then} \\
\text{Facts of the condition part are true when a rule belonging to FA is applying.}
\end{align*} \]

In more complex cases, a particular packet may be made active by different rules but these rules share a common set of conditions which are thus the meaning of the packet.

5.2.4 Processing explanatory rules - When the explanatory system is working, objects built by meta-rules are used lately to provide an explanation. This is because an explanatory rule may apply to several expert rules or several explanatory rules may apply to a single expert rule, giving a more argumented explanation.

The explanatory expert system works whenever an explanation is required, possibly at any stage of the analysis. It is also possible to ask a question before a problem is tackled. In this case, the system would work in absolute mode, without knowing the values of the domain facts. Thus the process starts with the creation of a set of objects which encode the particular type of explanation needed. It stops when all the possible explanatory rules have been fired and then must give an answer.

While developing meta-rules for discovering hidden knowledge we felt the need for the knowledge that drives the dialogue with the user (i.e. for the demand and the presentation of the explanation) to be made explicit in rules. The discussion of such rules is left outside the scope of this article.

5.3 Why making complicated what can be said simply?

To show the improvement in the answer given by IROISE, thanks to the method, we give below two sample explanations of a same inference step. The (French) sentence to be analyzed, "le chat mange la souris" features several lexical ambiguities. At the beginning the working window contains three words "le", "chat" and "mange". In the lexicon the system found two possible categories for the word "le" : determiner and pronoun. The analysis begins and after cycle 4, we see that "le" is still a determiner and no more a pronoun.

To the explanation query "Why is "le" a determiner ? (or why is it not any more a pronoun?)" the explanation provided by IROISE would be:
At cycle 3, the value of the attribute CATEGORY of the object SUBST..2 was DET because:

1. the rule Det/Pronoun-Ambiguity, fired at cycle 3, says:
2. The current context is LERLON,
3. the problem to solve is Rules20,
4. there is one STRUCTURE (STRUCT..15) of which the attribute NAME [4] is equal to S0 and of which the attribute ACTIVSET [5] verifies (INC 'FACT) or verifies (INC 'VACT) or verifies (INC 'NACT),
5. there is one STRUCTURE ?VAR1 (STRUCT..1) of which the attribute NAME [6] is equal to S1 or is equal to S2 or is equal to S3 and of which the attribute FOLLOW [7] is ?T2 (i.e. S2),
6. then
7. remove the SUBSTRUCTURE ?SS11,
8. modify the SUBSTRUCTURE ?SS12 with as new attribute CATEGORY (ADD ?T 'BEGIN GN) and
9. the new problem to solve is FILEBUFFER

Using the method we have described, the explanation is:

The word "le" has two possible interpretations, i.e. as a pronoun or as a determiner. The rule Det/Pronoun-Ambiguity deals with ambiguity, it determines that "le" is a determiner because determiner and pronoun are not compatible and "le" is followed by the noun "chat".

6. CONCLUSION

Explanation systems can be improved along different axes [14] [15]. The method described here focuses on the problem of encoding and searching for the information relevant to an explanation. It is necessary to provide explanations cast into domain terms rather than obscured by implementation details. There remain however a number of important problems to be solved.

When too much information is available, the useful and necessary data must be filtered so that only the most interesting is given. Providing too many details is as bad as not saying enough. The clarification may be done according to the user or not. Considering the reasoning, there are several places where factual information may be eliminated, without knowing its meaning, as for example the initial facts provided by the user. But the best way to do that is to know the user knowledge. Thus it is needed to define a user model [16] [17] and then apply it to obtain answers tailored to the user. This is a difficult problem and several solutions have been proposed. After the definition of the user knowledge level, the search for information can be oriented so that it corresponds to this representation. Data may be classified according to different knowledge levels, and explanations must be neither too simple nor too complicated.

The last step is the final touch of the explanation. It consists in generating natural language text, presenting the explanatory data found. The form of an explanation is very important to make it understandable. In particular, the vocabulary used and the structure of the sentences must be chosen carefully.

REFERENCES